
XLab/Xlib 使用手册

发布 1.0

2015 年 02 月 10 日

Contents

1 简介	1
1.1 下载	2
1.2 安装运行	2
1.3 登录	3
1.4 主界面	5
1.4.1 工具栏	5
脚本运行	5
帮助	7
查看log	7
关于	8
1.4.2 ODPS Table列表	8
1.4.3 窗口管理器	9
1.4.4 工作界面	9
1.5 数据表	10
1.5.1 数据表属性	11
1.5.2 数据表操作	12
导出数据	13
统计图	13
显示/隐藏列	16
文本显示	16
2 基本概念	19
2.1 表和分区	19
2.2 XLib稠密矩阵表	20
2.3 Xlib稀疏矩阵表	22
2.4 三元组表	24
2.5 索引三元组表	25
2.6 KV表	25
2.7 索引KV表	27
2.8 表达式	28

2.9 脚本语言	28
2.9.1 字符串	29
2.9.2 标识符的命名	29
2.9.3 变量	30
2.9.4 逻辑行与物理行	30
2.9.5 控制流	31
2.9.6 if语句	31
2.9.7 while语句	32
2.9.8 for循环	32
2.9.9 break语句	32
2.9.10 continue语句	33
2.9.11 函数	33
3 表的操作	35
3.1 函数	35
3.1.1 判断表是否存在	36
3.1.2 创建表	36
3.1.3 删除表	37
3.1.4 删除表分区	37
3.1.5 显示指定工程下的所有表名	37
3.1.6 设置全局生命周期	38
3.1.7 获取全局生命周期	38
3.1.8 获取表名	38
3.1.9 获取表注释	39
3.1.10 获取表的Project名	39
3.1.11 获取表所有者	39
3.1.12 获取表的生命周期	40
3.1.13 获取表创建时间	40
3.1.14 获取表最后修改时间	41
3.1.15 判断表是否为虚拟视图	41
3.1.16 获取所有列名	41
3.1.17 获取所有列注释	42
3.1.18 获取所有列类型	42
3.1.19 通过列名获取列类型	42
3.1.20 获取表总记录数	43
3.1.21 获取表指定分区的记录数	43
3.1.22 获取表所有分区	44
4 基本统计	45
4.1 全表基本统计	46
4.1.1 界面	46
4.1.2 函数	47

4.2 单列基本统计量	48
4.2.1 界面	48
4.2.2 函数	50
4.3 直方图	51
4.3.1 界面	51
4.3.2 函数	56
4.4 协方差	56
4.4.1 界面	56
4.4.2 函数	57
4.5 相关矩阵	57
4.5.1 界面	57
4.5.2 函数	58
4.6 Top100	58
4.6.1 界面	58
4.6.2 函数	59
4.7 bottom100	60
4.7.1 界面	60
4.7.2 函数	61
4.8 频率	61
4.8.1 界面	61
4.8.2 函数	62
4.9 百分位	63
4.9.1 界面	63
4.9.2 函数	64
4.10 累积分布图	65
4.10.1 界面	65
4.10.2 函数	66
4.11 概率密度图	66
4.11.1 界面	67
4.11.2 函数	68
4.12 全表统计汇总	68
4.12.1 界面	69
4.12.2 函数	70
4.13 分位数	70
4.13.1 界面	70
4.13.2 函数	72
4.14 排名和分位	73
4.14.1 函数	73
4.14.2 界面	74
4.15 按条件统计对比	76
4.15.1 界面	76
4.15.2 函数	80

4.16 按分区统计对比	81
4.16.1 界面	81
4.16.2 函数	83
4.17 分组聚合函数	84
4.17.1 界面	84
4.17.2 函数	86
4.18 窗口聚合函数	86
4.18.1 界面	86
4.18.2 函数	88
4.19 扩展直方图	89
4.19.1 界面	89
4.20 交叉表	93
4.20.1 界面	93
4.20.2 函数	95
4.21 对比交叉表	95
4.21.1 界面	95
4.22 排行榜	97
4.22.1 界面	97
4.22.2 函数	99
4.23 散布图矩阵	100
4.24 按行统计	103
4.24.1 函数	104
按行统计	104
5 统计分析	107
5.1 对应分析	107
5.1.1 界面	107
5.1.2 函数	108
5.2 多维对应分析	110
5.2.1 界面	110
5.2.2 函数	111
5.3 多重共线性	114
5.3.1 函数	114
5.4 主成分分析	116
5.4.1 函数	116
5.4.2 界面	117
6 数据处理	121
6.1 数据过滤	121
6.1.1 界面	121
6.1.2 函数	123
6.2 连续变量分组	124

6.2.1 界面	124
6.2.2 函数	128
6.3 唯一结果集	128
6.3.1 界面	128
6.3.2 函数	129
6.4 前N条记录	129
6.4.1 界面	130
6.4.2 函数	130
6.5 随机采样	131
6.5.1 界面	131
6.5.2 函数	131
6.6 加权采样	132
6.6.1 界面	132
6.6.2 函数	133
6.7 数据拆分	133
6.7.1 界面	133
6.7.2 函数	137
6.8 追加ID列	138
6.8.1 函数	138
6.9 多表列合并	138
6.9.1 函数	138
6.10 多表行合并	139
6.10.1 函数	139
6.11 排序	139
6.11.1 函数	139
6.12 信息值	140
6.12.1 界面	140
6.12.2 函数	144
6.13 变量转换	145
6.13.1 界面	145
6.13.2 函数	147
6.14 行列转换(行转列)	148
6.14.1 函数	148
6.14.2 界面	148
6.15 行列转换(列转行)	150
6.15.1 函数	150
6.15.2 界面	151
6.16 缺失值填充	152
6.16.1 界面	152
6.16.2 函数	160
6.17 归一化	161
6.17.1 界面	161

6.17.2 函数	165
6.18 标准化	166
6.18.1 界面	166
6.18.2 函数	166
6.19 分箱	166
6.19.1 界面	167
6.19.2 函数	174
6.20 数据生成	175
6.20.1 函数	175
6.20.2 界面	177
 7 格式转换	 181
7.1 转换图	182
7.2 普通表转XLib稠密矩阵表	182
7.3 普通表转XLib稀疏矩阵表	182
7.4 XLib稠密矩阵表转普通表	183
7.5 XLib稀疏矩阵表转普通表	183
7.6 XLib稀疏矩阵表转XLib稠密矩阵表	184
7.7 XLib稠密矩阵表转XLib稀疏矩阵表	184
7.8 索引三元组表转稀疏矩阵表	184
7.9 XLib稀疏矩阵表转索引三元组表	185
7.10 索引三元组表+行map或列map表转三元组表	185
7.11 获取三元组表的行map表或列map表	186
7.12 三元组表转索引三元组表+行map或列map表	186
7.13 三元组表+行map或列map表转索引三元组表	187
7.14 KV表转稠密矩阵表	188
7.15 KV表转稀疏矩阵表	189
7.16 索引KV表转稠密矩阵表	189
7.17 索引KV表转稀疏矩阵表	190
7.18 矩阵表转索引KV表	190
7.19 矩阵表转KV表	191
7.20 KV表转三元组表	192
 8 分类预测	 193
8.1 随机森林	193
8.1.1 函数	194
训练	194
预测	195
是否为随机森林模型	196
加载模型	196
导出模型	197
8.1.2 界面	198

训练	198
预测	204
模型显示	207
导出模型	211
定义模板	212
8.2 逻辑回归	214
8.2.1 函数	214
二分类训练	215
二分类预测	216
二分类加载模型	217
二分类逐步回归训练	218
二分类逐步回归预测	219
二分类逐步回归加载模型	219
多分类训练	220
多分类预测	221
多分类加载模型	222
8.2.2 界面	222
训练	222
预测	225
8.3 线性支持向量机(Linear SVM)	226
8.3.1 函数	226
训练	227
预测	228
加载模型	229
8.3.2 界面	229
训练	229
预测	231
8.4 非线性支持向量机(Nonlinear SVM)	233
8.4.1 函数	233
训练	234
预测	236
加载模型	237
8.4.2 界面	237
训练	237
预测	240
8.5 朴素贝叶斯(Naive Bayes)	242
8.5.1 函数	243
训练	243
预测	243
加载模型	244
8.5.2 界面	244
训练	244

预测	248
8.6 贝叶斯判别(Bayes)	251
8.6.1 函数	252
训练	252
预测	252
加载模型	253
8.6.2 界面	253
训练	253
预测	256
8.7 费希尔判别(Fisher)	260
8.7.1 函数	261
训练	261
预测	261
加载模型	262
8.7.2 界面	262
训练	262
预测	265
8.8 马氏距离判别(MDistance)	269
8.8.1 函数	270
训练	270
预测	270
加载模型	271
8.8.2 界面	271
训练	271
预测	274
8.9 决策树C5.0	278
8.9.1 函数	278
训练	279
预测	282
是否为C5.0模型	282
加载模型	283
导出模型	283
8.9.2 界面	284
训练	284
预测	289
打开模型	292
导出模型	296
8.10 梯度渐近树	297
8.10.1 函数	297
训练	298
预测	299
加载模型	299

判断模型	299
8.10.2 界面	300
训练	301
预测	304
模型显示	306
8.11 评估	307
8.11.1 界面	307
8.11.2 打开混淆矩阵或概率评估表	313
8.11.3 计算正确率	316
8.11.4 计算混淆矩阵	317
8.11.5 概率评估	318
8.11.6 加载混淆矩阵	319
8.11.7 加载概率评估数据	320
9 回归分析	321
9.1 线性回归	321
9.1.1 函数	321
训练	322
预测	322
加载模型	323
判断模型	323
9.1.2 界面	323
训练	323
预测	326
9.2 梯度渐近回归树	326
9.2.1 函数	327
训练	327
预测	328
加载模型	328
判断模型	329
9.2.2 界面	329
训练	329
预测	332
模型显示	334
9.3 线性支持向量回归 (Linear SVR)	335
9.3.1 函数	335
训练	336
预测	337
加载模型	338
9.3.2 界面	338
训练	338
预测	340

10 聚类分析	343
10.1 KMeans	343
10.1.1 函数	344
10.1.2 界面	345
10.1.3 例子	346
10.2 Canopy	350
10.2.1 函数	350
10.3 EM聚类	351
10.3.1 函数	351
10.3.2 界面	352
11 关联分析	357
11.1 函数	357
11.2 界面	358
11.3 例子	361
12 推荐算法	369
12.1 eTREC	369
12.1.1 函数	369
12.2 协同过滤-SVD	371
12.2.1 函数	372
训练	372
预测	372
推荐	373
加载模型	373
是否是SVD模型	374
12.3 协同过滤-ALS	374
12.3.1 函数	374
训练	374
预测	375
推荐	375
加载模型	376
是否是ALS模型	376
13 矩阵计算	377
13.1 函数	378
13.1.1 获取矩阵行数	379
13.1.2 获取矩阵列数	379
13.1.3 获取矩阵类型	379
13.1.4 获取矩阵非零元素数目	380
13.1.5 矩阵转置	380
13.1.6 产生随机矩阵	381
13.1.7 向量点积	381

13.1.8 矩阵的svd分解	382
13.1.9 求矩阵的迹	382
13.1.10 向量或者矩阵的范数	383
13.1.11 存取矩阵到表	383
14 自然语言处理算法	385
14.1 阿里分词 (AliWS)	385
14.1.1 函数	385
分词	386
14.2 文本归一化 (Normalize)	387
14.2.1 函数	388
归一化	388
14.3 过滤噪音 (FilterNoise)	389
14.3.1 函数	389
过滤噪音	389
14.4 分割单词串	390
14.4.1 函数	391
分割单词串为三元组 (splitWordsToTriple)	391
分割单词串为多行 (splitWordsToMultiRows)	391
14.5 文本特征选择 (FeatureSelect)	392
14.5.1 函数	392
卡方检验	392
信息增益	393
14.6 词频统计 (wordcount)	393
14.6.1 函数	393
词频统计	394
14.7 权重计算 (Weight)	394
14.7.1 函数	394
TF-IDF	395
14.8 Word2Vec	395
14.8.1 函数	395
训练	396
15 有向图相关计算	399
15.1 函数	400
15.1.1 获取有向图的节点数目	400
15.1.2 获取有向图的边的数目	401
15.1.3 保存为邻接矩阵形式	401
15.1.4 保存为三元组 (边的集合) 的形式	401
15.2 链接分析算法	402
15.2.1 PageRank算法	402
16 张量模型	405

16.1 GigaTensor	405
16.1.1 函数	405
训练	405
预测	406
加载模型	407
是否是GigaTensor模型	407
16.2 DinTucker	407
16.2.1 函数	407
训练	408
预测	409
加载模型	409
是否是DinTuckerTensor模型	409
17 常见问题	411
17.1 XLab/XLib使用说明	411
17.2 表的操作	412
17.3 数据处理	413
17.4 随机森林	414
17.5 逻辑回归	415
17.6 线性回归	416
17.7 GBRT	416
17.8 关联规则	417
17.9 矩阵计算	417
17.10 格式转换	418
17.11 基本统计	418
17.12 SVM	418
17.13 聚类	418
17.14 分类评估	419
17.15 排序	419
18 附录	421
18.1 附录 A 表达式支持的操作符、函数以及相应的优先级	421
18.1.1 表达式运算优先级	421
18.1.2 表达式字符串的注意事项	422
18.1.3 表达式支持的字符串操作和函数	422
18.1.4 正则表达式和sql正则表达式区别	422

第 1 章

简介

ODPS XLab/XLib可以帮助用户轻松处理海量数据，包含了统计、机器学习、矩阵等常用计算功能。

- XLab是ODPS提供的客户端。无论您是否有大数据分析的基础，都可以通过XLab图形界面，轻松上手；XLab还提供了脚本编辑执行功能，灵活方便、帮您成为大数据分析的高手。
- XLib是XLab的后台分布式算法库。可以通过XLab或ODPS客户端调用；由于二者使用相同的函数定义，XLab上的函数命令和脚本可以在ODPS客户端上直接执行。

XLab能做什么呢？XLab的主要操作均是基于ODPS的 `odps_table`，即输入数据和输出数据均存储在ODPS的表中。它支持 [基本统计](#)、[统计分析](#)、[数据处理](#)、[格式转换](#)、[分类预测](#)、[回归分析](#)、[聚类分析](#)、[关联分析](#)、[矩阵计算](#)、[有向图相关计算](#)等。

- 基本统计包含 [全表基本统计](#) 及 [单列基本统计量](#)（总个数，有效值个数，缺失值个数，和，平方和，立方和，最小值，最大值，极差，均值，方差，标准差，变异系数，标准误差，偏度，峰度，二阶矩，三阶矩，四阶矩，二阶中心距，三阶中心距，四阶中心距等）、[频率](#)、[直方图](#)、[交叉表](#)、[排行榜](#)、[百分位](#)、[散布图矩阵](#)、[相关矩阵](#)、[Top100](#)、[Top100](#)、[bottom100](#)、[分位数](#)、[排名和分位](#)、[全表统计汇总](#)、[按条件统计对比](#)、[按分区统计对比](#)、[分组聚合函数](#)、[扩展直方图](#)、[对比交叉表](#)。
- 统计分析包括：[对应分析](#)、[多维对应分析](#)、[多重共线性](#) 和 [主成分分析](#)。
- 数据处理包括：[数据过滤](#)、[连续变量分组](#)、[唯一结果集](#)、[前N条记录](#)、[随机采样](#)、[加权采样](#)、[数据拆分](#)、[追加ID列](#)、[多表列合并](#)、[多表行合并](#)、[排序](#)、[信息值](#)、[变量转换](#)、[缺失值填充](#)、[归一化](#)、[标准化](#)、[分箱](#)、[数据生成](#)。
- 格式转换包括：[ODPS 表和分区](#)、[Xlib稀疏矩阵表](#)、[Xlib稠密矩阵表](#)、[三元组表](#)、[索引三元组表](#)、[KV表](#)、[索引KV表](#)间的相互转换。

- 分类预测包括: 随机森林、逻辑回归(二分类回归、多分类回归和逐步回归)、线性支持向量机(Linear SVM)、朴素贝叶斯(Naive Bayes)、贝叶斯判别(Bayes)、费希尔判别(Fisher)、马氏距离判别(MDistance)。
- 回归分析包括: 线性回归 和 梯度渐近回归树。
- 聚类分析主要是 odps_xlib_cluster_kmeans。
- 推荐算法主要是 eTREC。
- 关联分析是指关联式规则(Association Rules, AR)。
- 矩阵计算主要支持这些运算: 矩阵构造, 获取矩阵信息(类型、行数、列数和非零元数目), 打印矩阵基本信息(矩阵类型、行数、列数和非零元数目), 加减乘除运算, 向量点积, 矩阵和向量范数, 矩阵的迹, 矩阵转置, 矩阵的svd分解。
- 有向图主要是: 有向图的构造, 获取有向图的信息, PageRank算法。

1.1 下载

XLab的下载请访问XLab的主页: [XLab](#)。

注解: XLab用户群: 955167455, 欢迎加入讨论交流。

1.2 安装运行

解压XLab.zip, 能看到如下目录及文件:

```
lib/ start.bat XLab.jar
```

lib目录包含XLab依赖的库文件, start.bat是Windows下的可执行文件, XLab.jar是XLab的jar包, 可在Windows和Linux下执行。

安装环境:

- Windows或Linux 操作系统;
- JDK 1.6版本。

运行方式:

- Windows用户, 可点击start.bat或XLab.jar, 进入登录界面;
- Linux用户, 执行java -jar XLab.jar, 进入登录界面。

运行后, 会增加以下目录及文件:

log/ history/ login.cfg

log目录包含XLab运行的日志文件, history目录包含临时表和job信息, login.cfg是登录配置文件。

1.3 登录

运行后, 会出现登录界面, 如下图:



用户需要填写以下登录信息:

END PONIT=<阿里云的URL (必选)>

ACCESS ID=<阿里云账号相关信息 (必选)>

ACCESS KEY=<阿里云账号相关信息 (必选)>

默认Project=<默认项目空间名称, 配置此项后不必特殊指定数据导入、导出表所在的项目空间 (必选)>

关联Project=<关联项目空间名称, 是指同一END PONIT下除默认Project之外的Project, 如果填写此项, 则XLab的Table列表会自动在表前面加

全局生命周期=<单位为天, 空或者-1为不设置生命周期。

此变量用来设置全局的生命周期, 当此变量被设置后,

所有的输出表(调用sql函数除外)都会被设置上生命周期,

生命周期的时间为此变量的值。 (可选) >

不同的 END POINT可以连接到不同的ODPS Service, 输入完成后点击登录按钮或者输入回车键, 后台验证完成后界面跳转到主界面。其中Project Name为默认project, 即之后的操作都会运行在此project下, 关联project可以设置多个, 用换行符(回车)分割, 用户可以点击选择保存登录信息, 用以下一次登录, 登录信息会以明文形式保存于执行目录下的login.cfg文件, 请谨慎选择。

Odps生命周期使用见odps sql文档中修改表的生命周期属性章节, xlib中全局生命周期的定义是如果被设置为有效, 则所有的xlib输出表(sql函数的输出表除外)都会被设置上生命周期, 值为全局生命周期的值。通俗的讲, 此时的全局生命周期跟odps sql创建表中的lifecycle关键字等价。

Odps对lifecycle有一些系统管理, 在系统上总共有三个选项, 如下:

- optional: 默认, 不强制使用生命周期
- Mandatory: 强制用户在建表时指定生命周期
- Inherit: 如果用户在建表时没有指定, 取project中的odps.sql.lifecycle.value

全局生命周期中的值为两种:

- -1: 全局生命周期不起效。也就是Odps lifecycle选项为optional时xlib输出表(除sql函数外)无生命周期, Mandatory时报错, Inherit时取project中的odps.sql.lifecycle.value。
- 大于0: xlib的输出表(除sql函数外), 生命周期被设置上全局生命周期的值

全局生命周期值的修改和获取, 可以使用如下函数:

指定全局xlib产出表的生命周期, 此函数指定了所有xlib产出表的默认生命周期。在project属性中lifecycle为mandatory时, 必须设置全局生命周期。

```
def setGlobalLifeCycle(lifeCycle) :
```

参数:

- lifeCycle: 生命周期, 单位为天, Long型。值为Table.INVALIDGLOBALIFCYCLE时, xlib产出表不设置生命周期。

示例:

```
Table.setGlobalLifeCycle(37231)
```

得到全局xlib产出表的生命周期

```
def getGlobalLifeCycle() :
```

返回值:

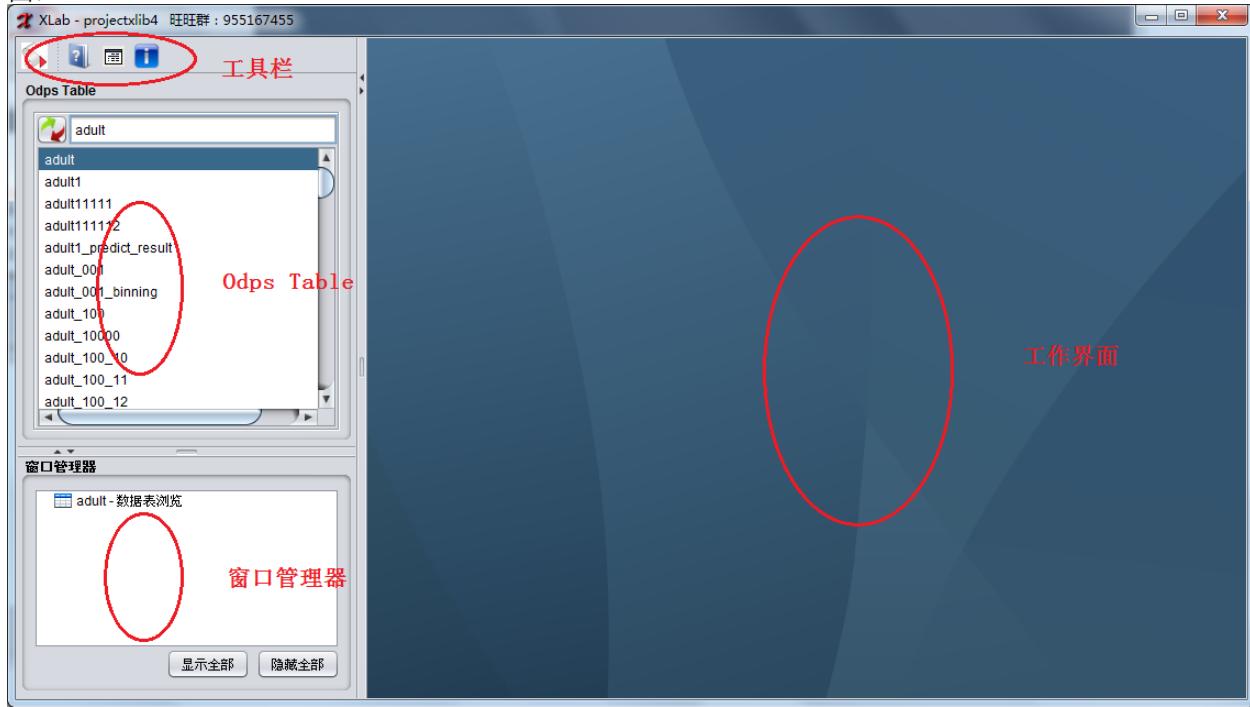
- 返回全局生命周期的值。

示例:

```
Table.getGlobalLifeCycle()
```

1.4 主界面

登录之后，进入主界面。主界面分为四个区域：工具栏，ODPS Table列表，窗口管理器和工作界面。如下图：



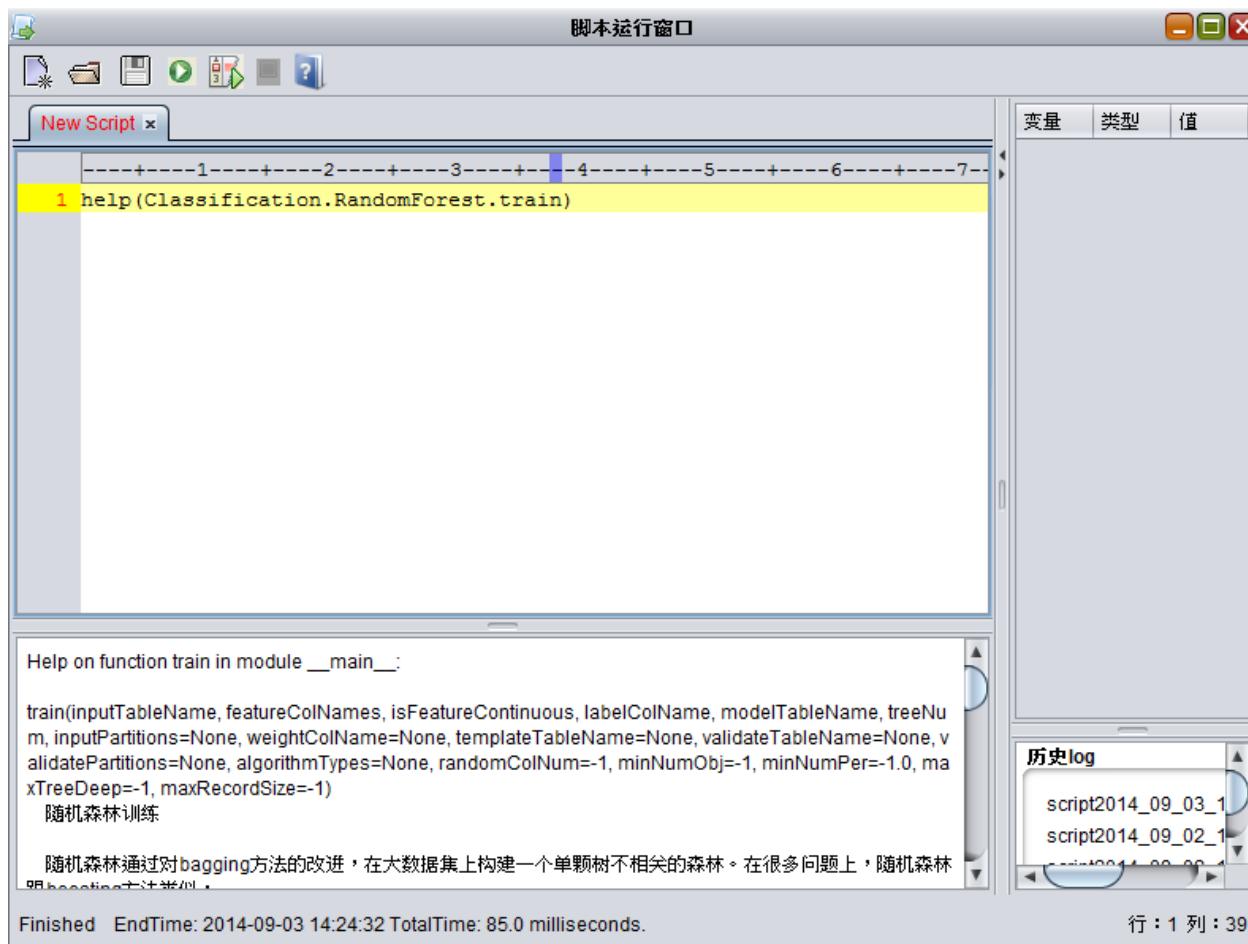
1.4.1 工具栏

工具栏包含XLab的各项应用工具：

- 脚本运行
- 帮助
- 查看log
- 关于

脚本运行

XLab提供了一个脚本语言的基础编程环境，支持Python脚本语法。用户可以通过脚本运行窗口实现脚本编辑和执行。如下图：



补全以及函数提示快捷键 : Ctrl+p

脚本运行工具栏提供以下几个功能:

- 新建脚本
- 打开脚本
- 保存脚本
- 运行脚本
- 运行选中脚本
- 停止运行

XLab脚本除了支持调用XLib的函数, 还支持直接调用ODPS SQL, 返回SQL运行结果。ODPS SQL的详细说明, 可参考其官网。

调用ODPS SQL的函数为sql(str), 可以通过help(sql)查看使用帮助。

```
def sql(str):
```

参数:

- str: 字符串

返回:

- 计算结果

示例1:

```
sql("create table sql_table(col double)")
```

示例1的结果为生成一张表: sql_table

示例2:

```
print sql('select * from dual')
```

示例2的结果:

```
"f1"  
0
```

注意事项:

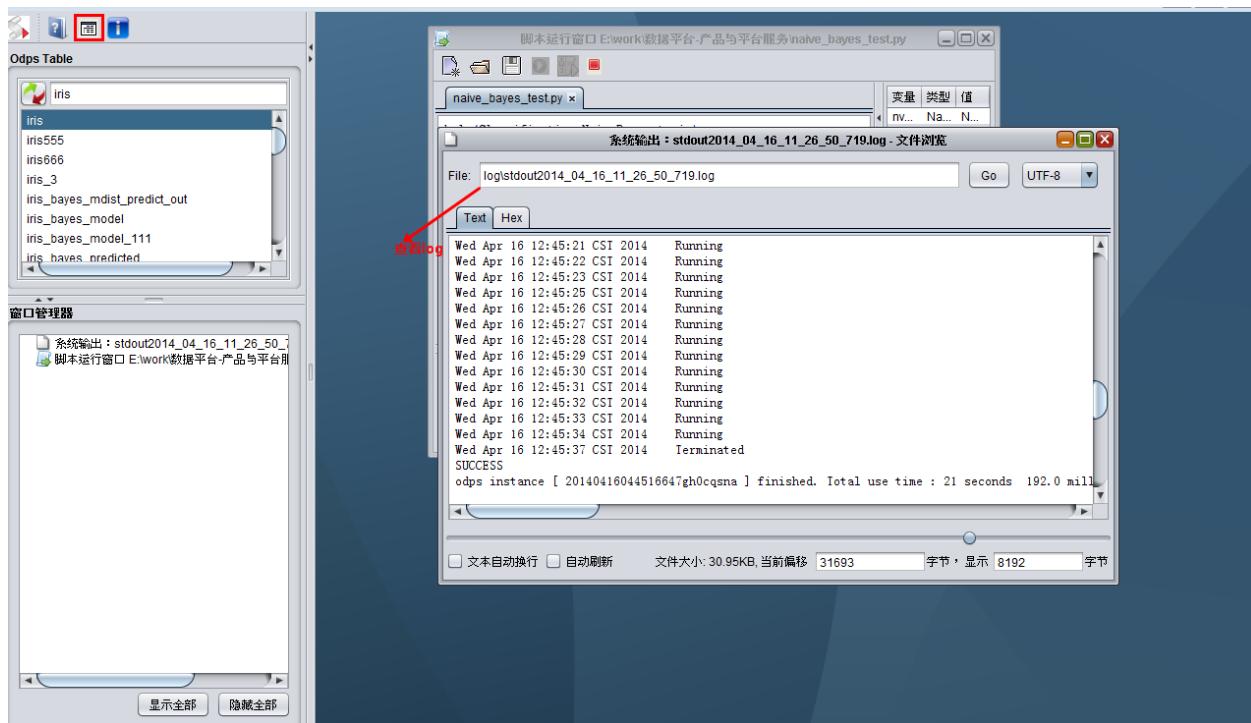
- 函数计算结果需要通过执行命令print sql(str)打印出来

帮助

如果用户对XLab不熟悉, 可点击”帮助”按钮, 可打开XLab使用手册。

查看log

XLab提供查看log的功能, 当用户执行任务时, 点击”查看log”按钮, 可以查看执行log, 如下图:



关于

关于： 提供XLab版本等信息。

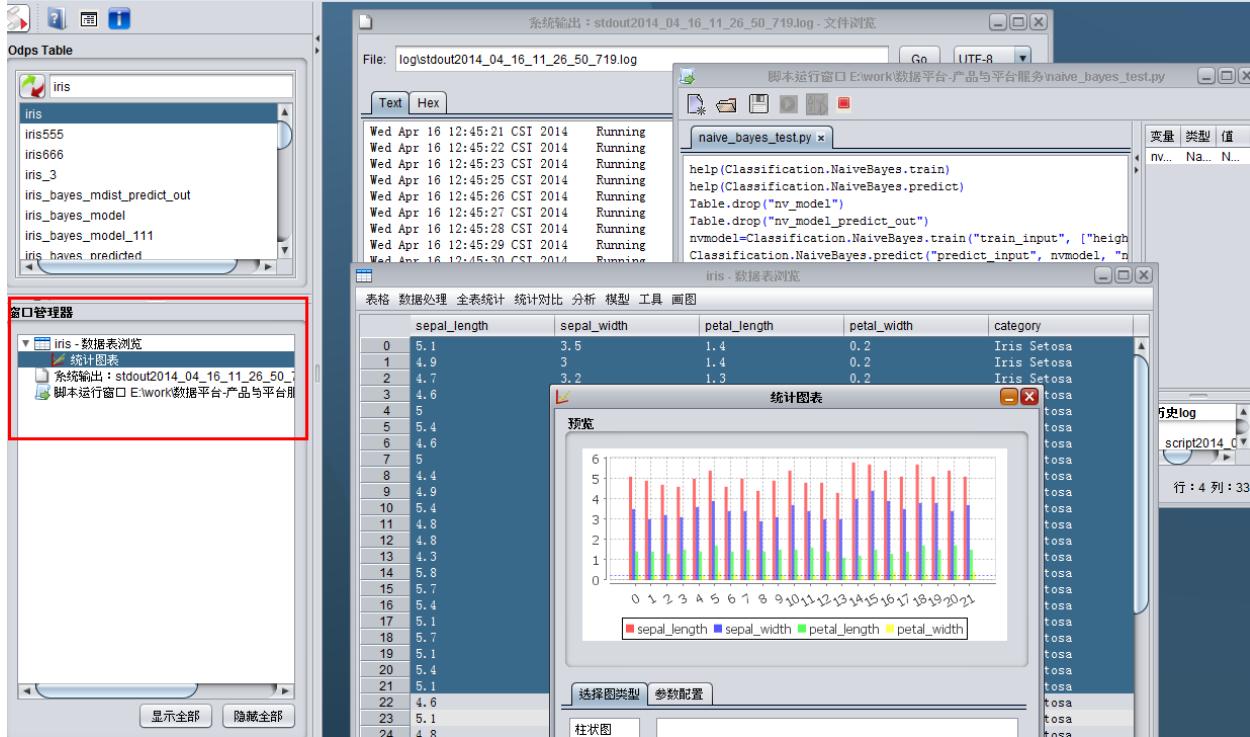
1.4.2 ODPS Table列表

ODPS Table列表中展示默认Project和关联Project下面的Table列表，用户也可在搜索框中输入关键词，搜索相关Table，如下图：



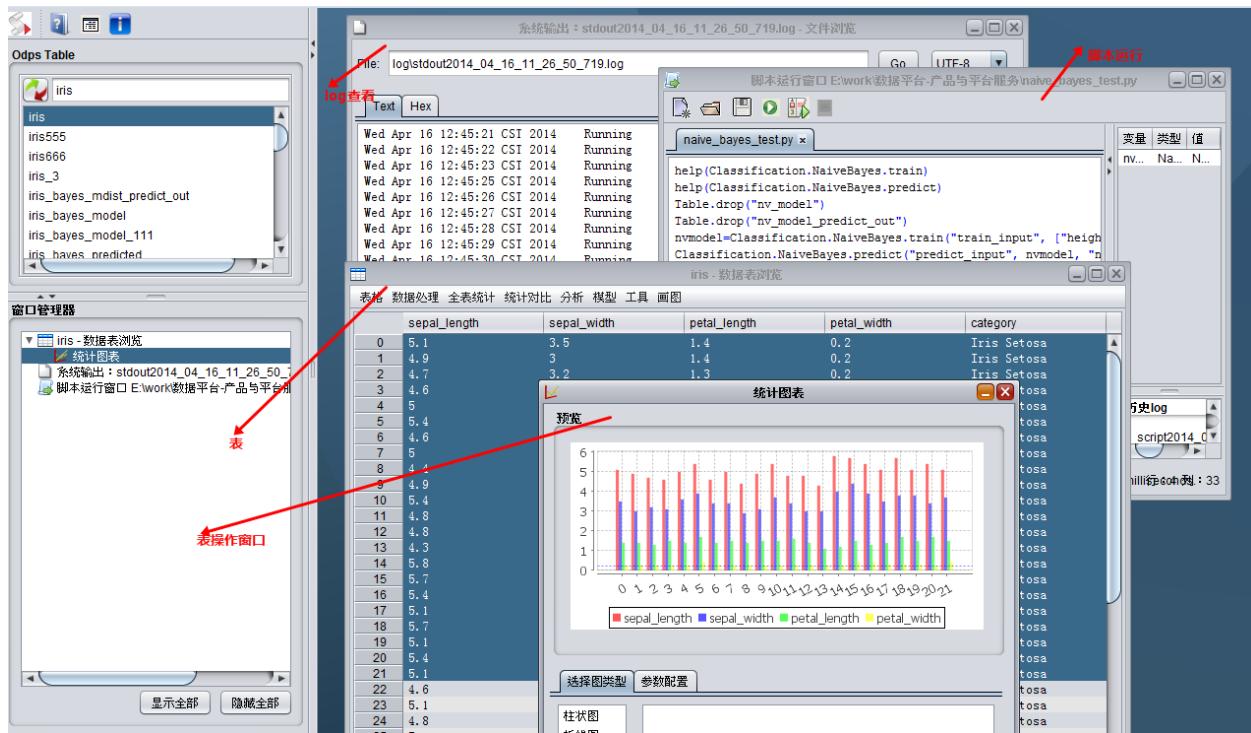
1.4.3 窗口管理器

窗口管理器包括所有在工作桌面上运行的窗口列表。中间树状结构显示XLAB内部各个窗体及其对应关系。如图中的两个数据表格窗口在树中有父子节点关系，则表示子节点对应窗口数据由父节点操作后生成。用户点击树节点可以显示隐藏对应窗口，也可以点击下方显示隐藏全部窗口。如下图：



1.4.4 工作界面

工作界面用于容纳XLab各类窗口，包括表，脚本，log等，如下图：



1.5 数据表

XLab的主要操作均是基于数据表ODPS Table的，即输入数据和输出数据均存储在ODPS Table中。因此，XLab的主要功能的展现是伴随着数据表的，比如数据表属性查看，数据表操作(导出、统计、显示)等，如下图：

iris - 数据表浏览

	length	sepal_width	petal_length	petal_width	category
1	4.5	3.5	1.4	0.2	Iris Setosa
2	4.3	3.0	1.4	0.2	Iris Setosa
3	4.6	3.2	1.3	0.2	Iris Setosa
4	5.0	3.1	1.5	0.2	Iris Setosa
5	5.4	3.6	1.4	0.2	Iris Setosa
6	4.6	3.9	1.7	0.4	Iris Setosa
7	5.0	3.4	1.4	0.3	Iris Setosa
8	4.4	2.9	1.4	0.2	Iris Setosa
9	4.9	3.1	1.5	0.1	Iris Setosa
10	5.4	3.7	1.5	0.2	Iris Setosa
11	4.8	3.4	1.6	0.2	Iris Setosa
12	4.8	3.0	1.4	0.1	Iris Setosa
13	4.3	3.0	1.1	0.1	Iris Setosa
14	5.8	4.0	1.2	0.2	Iris Setosa
15	5.7	4.4	1.5	0.4	Iris Setosa
16	5.4	3.9	1.3	0.4	Iris Setosa
17	5.1	3.5	1.4	0.3	Iris Setosa
18	5.7	3.8	1.7	0.3	Iris Setosa
19	5.1	3.8	1.5	0.3	Iris Setosa
20	5.4	3.4	1.7	0.2	Iris Setosa
21	5.1	3.7	1.5	0.4	Iris Setosa
22	4.6	3.6	1.0	0.2	Iris Setosa
23	5.1	3.3	1.7	0.5	Iris Setosa
24	4.8	3.4	1.9	0.2	Iris Setosa
25	5.0	3.0	1.6	0.2	Iris Setosa
26	5.0	3.4	1.6	0.4	Iris Setosa
27	5.2	3.5	1.5	0.2	Iris Setosa
28	5.2	3.4	1.4	0.2	Iris Setosa
29	5.1	3.0	1.5	0.1	Iris Setosa

1.5.1 数据表属性

当用户需要了解数据表属性时，在ODPS Table列表选中表名，右键选择”属性”，如属性的信息包括以下的信息：数据列定义中可以看到表的列名，列类型和列对应的注释。

分区的定义和分区的列表，用户可以点击左下角的”刷新行数”按钮得到分区和总表的行数信息，或者点击右下角”打开”按钮进行数据表浏览，如下图：



1.5.2 数据表操作

当用户需要对数据进行表格浏览和操作时, ODPS Table列表双击表名, 进入数据表浏览界面。表下方显示当前显示的数据信息, 最多显示10000行, 左下角可以按分区(partition)显示数据, 默认所有分区(Whole Table)。

- 无分区的表, 分区(partition)为Whole Table, 比如表iris, 如下图:

	sepal_length	sepal_width	petal_length	petal_width	category
0	5.1	3.5	1.4	0.2	Iris Setosa
1	4.9	3	1.4	0.2	Iris Setosa
2	4.7	3.2	1.3	0.2	Iris Setosa
3	4.6	3.1	1.5	0.2	Iris Setosa
4	5	3.6	1.4	0.2	Iris Setosa
5	5.4	3.9	1.7	0.4	Iris Setosa
6	4.6	3.4	1.4	0.3	Iris Setosa
7	5	3.4	1.5	0.2	Iris Setosa
8	4.4	2.9	1.4	0.2	Iris Setosa
9	4.9	3.1	1.5	0.1	Iris Setosa
10	5.4	3.7	1.5	0.2	Iris Setosa
11	4.8	3.4	1.6	0.2	Iris Setosa
12	4.8	3	1.4	0.1	Iris Setosa
13	4.3	3	1.1	0.1	Iris Setosa
14	5.8	4	1.2	0.2	Iris Setosa
15	5.7	4.4	1.5	0.4	Iris Setosa
16	5.4	3.9	1.3	0.4	Iris Setosa
17	5.1	3.5	1.4	0.3	Iris Setosa
18	5.7	3.8	1.7	0.3	Iris Setosa
19	5.1	3.8	1.5	0.3	Iris Setosa
20	5.4	3.4	1.7	0.2	Iris Setosa
21	5.1	3.7	1.5	0.4	Iris Setosa
22	4.6	3.6	1	0.2	Iris Setosa
23	5.1	3.3	1.7	0.5	Iris Setosa
24	4.8	3.4	1.9	0.2	Iris Setosa
25	5	3	1.6	0.2	Iris Setosa
26	5	3.4	1.6	0.4	Iris Setosa
27	5.2	3.5	1.5	0.2	Iris Setosa
28	5.2	3.4	1.4	0.2	Iris Setosa

- 有分区的表，默认分区(partition)为Whole Table，比如表iris_bayes_model，可选择分区：
pt=000000查看数据，如下图：

	vallabel	restlabel	namefeature	mean	inv0	inv1	inv2	inv3
0	Iris Se...	10.8376...	sepal_l...	5.00599...	19.0306...	-12.555...	-4.8438...	-2.5686...
1	Iris Se...	10.8376...	sepal_w...	3.41800...	-12.555...	15.8318...	1.53218...	-4.9917...
2	Iris Se...	10.8376...	petal_l...	1.464	-4.8438...	1.53218...	38.2544...	-16.044...
3	Iris Se...	10.8376...	petal_w...	0.24399...	-2.5686...	-4.9917...	-16.044...	102.280...
4	Iris Ve...	8.67710...	sepal_l...	5.936	9.50276...	-3.6762...	-8.6317...	6.45450...
5	Iris Ve...	8.67710...	sepal_w...	2.77000...	-3.6762...	19.7109...	2.11602...	-19.480...
6	Iris Ve...	8.67710...	petal_l...	4.26	-8.6317...	2.11602...	19.8037...	-26.937...
7	Iris Ve...	8.67710...	petal_w...	1.32599...	6.45450...	-19.480...	-26.937...	87.2447...
8	Iris Vi...	6.72983...	sepal_l...	6.58799...	10.5338...	-3.4797...	-9.9601...	1.78815...
9	Iris Vi...	6.72983...	sepal_w...	2.97399...	-3.4797...	15.8754...	1.10268...	-8.4728...
10	Iris Vi...	6.72983...	petal_l...	5.552	-9.9601...	1.10268...	13.4058...	-2.8909...
11	Iris Vi...	6.72983...	petal_w...	2.026	1.78815...	-8.4728...	-2.8909...	19.3140...

导出数据

当用户想导出表格中的数据，可以点击菜单：表格→导出数据。导出后的数据表格变为可修改模式(类似excel)，用户可以针对表格上的数据进行直接修改，复制等，如下图：

数据导出表 (导出前 150 行, 总共 150 行)				
基本统计 画图				
sepal_length	sepal_width	petal_length	petal_width	category
5.1	3.5	1.4	0.2	Iris Setosa
4.9	3	1.4	0.2	Iris Setosa
4.7	3.2	1.3	0.2	Iris Setosa
4.6	3.1	1.5	0.2	Iris Setosa
5	3.6	1.4	0.2	Iris Setosa
5.4	3.9	1.7	0.4	Iris Setosa
4.6	3.4	1.4	0.3	Iris Setosa
5	3.4	1.5	0.2	Iris Setosa
4.4	2.9	1.4	0.2	Iris Setosa
4.9	3.1	1.5	0.1	Iris Setosa
5.4	3.7	1.5	0.2	Iris Setosa
4.8	3.4	1.6	0.2	Iris Setosa
4.8	3	1.4	0.1	Iris Setosa
4.3	3	1.1	0.1	Iris Setosa
5.8	4	1.2	0.2	Iris Setosa
5.7	4.4	1.5	0.4	Iris Setosa
5.4	3.9	1.3	0.4	Iris Setosa
5.1	3.5	1.4	0.3	Iris Setosa
5.7	3.8	1.7	0.3	Iris Setosa
5.1	3.8	1.5	0.3	Iris Setosa
5.4	3.4	1.7	0.2	Iris Setosa
5.1	3.7	1.5	0.4	Iris Setosa
4.6	3.6	1	0.2	Iris Setosa
5.1	3.3	1.7	0.5	Iris Setosa
4.8	3.4	1.9	0.2	Iris Setosa
5	3	1.6	0.2	Iris Setosa
5	3.4	1.6	0.4	Iris Setosa
5.2	3.5	1.5	0.2	Iris Setosa
5.2	3.4	1.4	0.2	Iris Setosa
4.7	3.2	1.6	0.2	Iris Setosa
4.8	3.1	1.6	0.2	Iris Setosa
5.4	3.4	1.5	0.4	Iris Setosa
5.2	4.1	1.5	0.1	Iris Setosa
5.5	4.2	1.4	0.2	Iris Setosa

统计图

在数据表浏览或者数据导出表中选中要展示的数据，点击画图→统计图，可以使用统计图功能，如下图：

iris - 数据表浏览

	sepal_length	sepal_width	统计图	length	petal_width	category
0	5.1	3.5	1.4	0.2	Iris Setosa	
1	4.9	3	1.4	0.2	Iris Setosa	
2	4.7	3.2	1.3	0.2	Iris Setosa	
3	4.6	3.1	1.5	0.2	Iris Setosa	
4	5	3.6	1.4	0.2	Iris Setosa	
5	5.4	3.9	1.7	0.4	Iris Setosa	
6	4.6	3.4	1.4	0.3	Iris Setosa	
7	5	3.4	1.5	0.2	Iris Setosa	
8	4.4	2.9	1.4	0.2	Iris Setosa	
9	4.9	3.1	1.5	0.1	Iris Setosa	
10	5.4	3.7	1.5	0.2	Iris Setosa	
11	4.8	3.4	1.6	0.2	Iris Setosa	
12	4.8	3	1.4	0.1	Iris Setosa	
13	4.3	3	1.1	0.1	Iris Setosa	
14	5.8	4	1.2	0.2	Iris Setosa	
15	5.7	4.4	1.5	0.4	Iris Setosa	
16	5.4	3.9	1.3	0.4	Iris Setosa	
17	5.1	3.5	1.4	0.3	Iris Setosa	
18	5.7	3.8	1.7	0.3	Iris Setosa	
19	5.1	3.8	1.5	0.3	Iris Setosa	
20	5.4	3.4	1.7	0.2	Iris Setosa	
21	5.1	3.7	1.5	0.4	Iris Setosa	
22	4.6	3.6	1	0.2	Iris Setosa	
23	5.1	3.3	1.7	0.5	Iris Setosa	
24	4.8	3.4	1.9	0.2	Iris Setosa	
25	5	3	1.6	0.2	Iris Setosa	
26	5	3.4	1.6	0.4	Iris Setosa	
27	5.2	3.5	1.5	0.2	Iris Setosa	
28	5.2	3.4	1.4	0.2	Iris Setosa	
...	

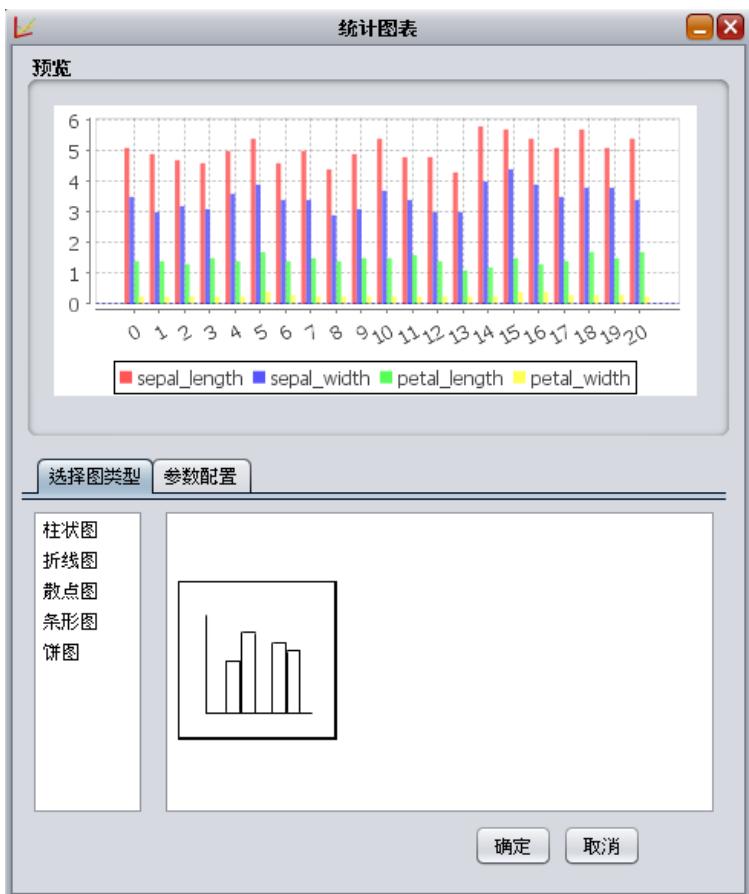
Whole Table 位置 数据大小: -- 行, 当前显示前 50 行 (最多10000行) Go!

或者:

数据导出表 (导出前 150 行, 总共 150 行)

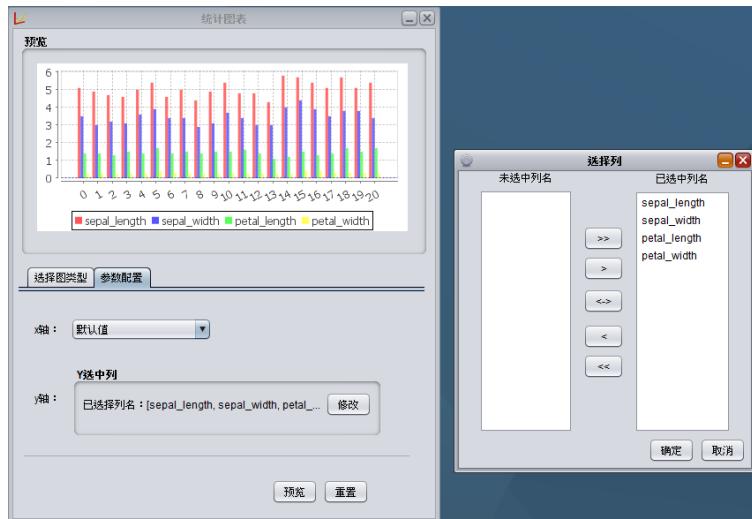
	sepal_len	统计图	sepal_width	petal_length	petal_width	category
5.1			3.5	1.4	0.2	Iris Setosa
4.9			3	1.4	0.2	Iris Setosa
4.7			3.2	1.3	0.2	Iris Setosa
4.6			3.1	1.5	0.2	Iris Setosa
5			3.6	1.4	0.2	Iris Setosa
5.4			3.9	1.7	0.4	Iris Setosa
4.6			3.4	1.4	0.3	Iris Setosa
5			3.4	1.5	0.2	Iris Setosa
4.4			2.9	1.4	0.2	Iris Setosa
4.9			3.1	1.5	0.1	Iris Setosa
5.1			3.7	1.5	0.2	Iris Setosa
4.8			3.4	1.6	0.2	Iris Setosa
4.8			3	1.4	0.1	Iris Setosa
4.3			3	1.1	0.1	Iris Setosa
5.8			4	1.2	0.2	Iris Setosa
5.7			4.4	1.5	0.4	Iris Setosa
5.4			3.9	1.3	0.4	Iris Setosa
5.1			3.5	1.4	0.3	Iris Setosa
5.7			3.8	1.7	0.3	Iris Setosa
5.1			3.8	1.5	0.3	Iris Setosa
5.4			3.4	1.7	0.2	Iris Setosa
5.1			3.7	1.5	0.4	Iris Setosa
4.6			3.6	1	0.2	Iris Setosa
5.1			3.3	1.7	0.5	Iris Setosa
4.8			3.4	1.9	0.2	Iris Setosa
5			3	1.6	0.2	Iris Setosa
5.4			3.4	1.6	0.4	Iris Setosa
5.2			3.5	1.5	0.2	Iris Setosa
5.2			3.4	1.4	0.2	Iris Setosa
4.7			3.2	1.6	0.2	Iris Setosa
4.8			3.1	1.6	0.2	Iris Setosa
5.4			3.4	1.5	0.4	Iris Setosa
5.2			4.1	1.5	0.1	Iris Setosa
5.5			4.2	1.4	0.2	Iris Setosa

打开统计图界面, 如下图:



使用选择图类型选项页可以分别选择柱状图，折线图，散点图，条形图或者饼图，每种不同类型的图下有一种或者多种样式。点击确定按钮可以弹出当前预览的大图表示。

在参数配置选项页下可以分别选择x轴或者y轴的参考列，如下图：



选中对应的列之后，点击预览可以看到所选择参数和所选择图类型对应的统计图的预览。

统计图遵守以下规则：

- 不支持空值和NaN等非法值。
- 所有的y轴必须为数值类型(整形、浮点型)。
- 当x轴有重复值时, 取最后一个值。
- 默认情况下, x轴使用行号来代替。
- 饼图为一列, 分类x轴所选值, 值为y轴对应值。
- 柱状图, 折线图, 条形图中, 分类值为列名, x轴和y轴对应图中x和y。
- 散点图中如果x轴选中的为非数值类型的数据, 则用行号来代替。

显示/隐藏列

当用户想隐藏某些列, 可以点击菜单: 表格→显示/隐藏 列, 选择隐藏的列, 确定。也可以在数据表上选中一列, 右键→隐藏列, 隐藏选中的列。

当用户想显示某些隐藏的列, 可以点击菜单: 表格→显示/隐藏 列, 选择隐藏的列, 确定, 如下图:



文本显示

用户也可用文本显示数据表的内容, 但不可编辑, 可以点击菜单: 表格→文本显示, 如下图:

iris - 数据表浏览

表格 数据处理 全表统计 统计对比 分析 模型 工具

	sepal_length	sepal_width	petal_length	petal_width	category
0	5.1	3.5	1.4	0.2	Iris Setosa
1	4.9	3	1.4	0.2	Iris Setosa
2	4.7				Iris Setosa
3	4.6				Iris Setosa
4	5				Iris Setosa
5	5.4	3.0	1.4	0.2	Iris Setosa
6	4.6	3.2	1.3	0.2	Iris Setosa
7	5	3.1	1.5	0.2	Iris Setosa
8	4.4	3.6	1.4	0.2	Iris Setosa
9	4.9	3.9	1.7	0.4	Iris Setosa
10	5.4	3.4	1.4	0.3	Iris Setosa
11	4.8	3.4	1.5	0.2	Iris Setosa
12	4.8	2.9	1.4	0.2	Iris Setosa
13	4.3	3.1	1.5	0.1	Iris Setosa
14	5.8	3.7	1.5	0.2	Iris Setosa
15	5.7	3.4	1.6	0.2	Iris Setosa
16	5.4	4.8	3.0	1.4	Iris Setosa
17	5.1	4.3	3.0	1.1	Iris Setosa
18	5.7	5.8	4.0	1.2	Iris Setosa
19	5.1	5.7	4.4	1.5	Iris Setosa
20	5.4	5.4	3.9	1.3	Iris Setosa
21	5.1	5.1	3.5	1.4	Iris Setosa
22	4.6	5.7	3.8	1.7	Iris Setosa
23	5.1	5.1	3.8	1.5	Iris Setosa
24	4.8	5.4	3.4	1.7	Iris Setosa
25	5	5.1	3.7	1.5	Iris Setosa
26	5	4.6	3.6	1.0	Iris Setosa
27	5.2	5.1	3.3	1.7	Iris Setosa
28	5.2				Iris Setosa

第 2 章

基本概念

XLab涉及这些概念：表(table)和分区(partition)、XLib稠密矩阵表、XLib稀疏矩阵表、三元组表、索引三元组表、KV表、索引KV表、表达式和脚本语言。本章节将简要介绍这些概念。

XLab之所以提供XLib稠密矩阵表、XLib稀疏矩阵表的原因是ODPS表有列的限制（1024列），对于列大于1024的数据ODPS表没有办法表示，因此自定义了XLib稀疏矩阵表和XLib稠密矩阵表这样的数据结构来解决这个问题。XLab提供许多算法的输入支持XLib稀疏矩阵表和XLib稠密矩阵表，比如：逻辑回归，GBRT，SVM等，这样用户就不必担心如何处理特征(feature)特别多的问题。同样，我们也可以对列大于1024的矩阵进行计算。

那如何生成XLib稀疏矩阵表和XLib稠密矩阵表呢？我们需要借助KV表、索引KV表和三元组表、索引三元组表来将用户的数据组织，然后转换为XLib稀疏矩阵表和XLib稠密矩阵表。因此XLab又定义了KV表、索引KV表和三元组表、索引三元组表的表示形式。这些表间的相互转换不需要用户做额外工作，XLab提供了相应转换工具。这个转换工具可以通过界面和函数操作，详见“格式转换”章节。

2.1 表和分区

XLab/XLib中提到的表(table)和分区(partition)均是指ODPS的表(table)和分区(partition)。

示例，表(Table)table_4x4，4行4列，如下图：

table_4x4 - 数据表浏览

表格 数据处理 全表描述 统计 分析 模型 工具 画图

	x0	x1	x2	x3
0	13.58	0	0	49.6
1	22.31	0	12.31	47.8
2	28.82	0	16.18	0
3	0	3.54	0	43.2

Whole Table ▾ 位置 数据大小: 4 行, 当前显示前 50 行 (最多10000行) GO!

2.2 XLib稠密矩阵表

稠密矩阵是指非0元素占所有元素比例较大的矩阵。XLib稠密矩阵表是指存储稠密矩阵的0dps表。我们用四个分区来对其进行定义和描述XLib稠密矩阵表。XLib稠密矩阵表，包括四个分区：

- matrixconf: 矩阵类型为1, 代表稠密矩阵;
- matrixdata: 按先行后列存储全部数据;
- matrixrow: 空;
- matrixcol: 空。

示例, XLib稠密矩阵表table2dense_matrix_4x4(与表table_4x4对应):

- matrixconf: 矩阵类型为1, 代表稠密矩阵, 如下图:

table2dense_matrix_4x4

表格 数据处理 全表描述 统计 分析 模型 工具 画图

类型: 稠密矩阵
行数
列数
元素总数

	val
0	1
1	4
2	4
3	16

part=matrixconf ▾ 位置 数据大小: 20 行, 当前显示前 50 行 (最多10000行) GO!

- matrixdata: 按先行后列存储全部数据, 如下图:

table2dense_matrix_4x4

表格 数据处理 全表描述 统计 分析 模型 工具 画图

	val
0	13.58
1	0
2	0
3	49.6
4	22.31
5	0
6	12.31
7	47.8
8	28.82
9	0
10	16.18
11	0
12	0
13	3.54
14	0
15	43.2

part=matrixdata ▾ 位置 数据大小: 20 行, 当前显示前 50 行 (最多10000行) GO!

- matrixrow: 空, 如下图:

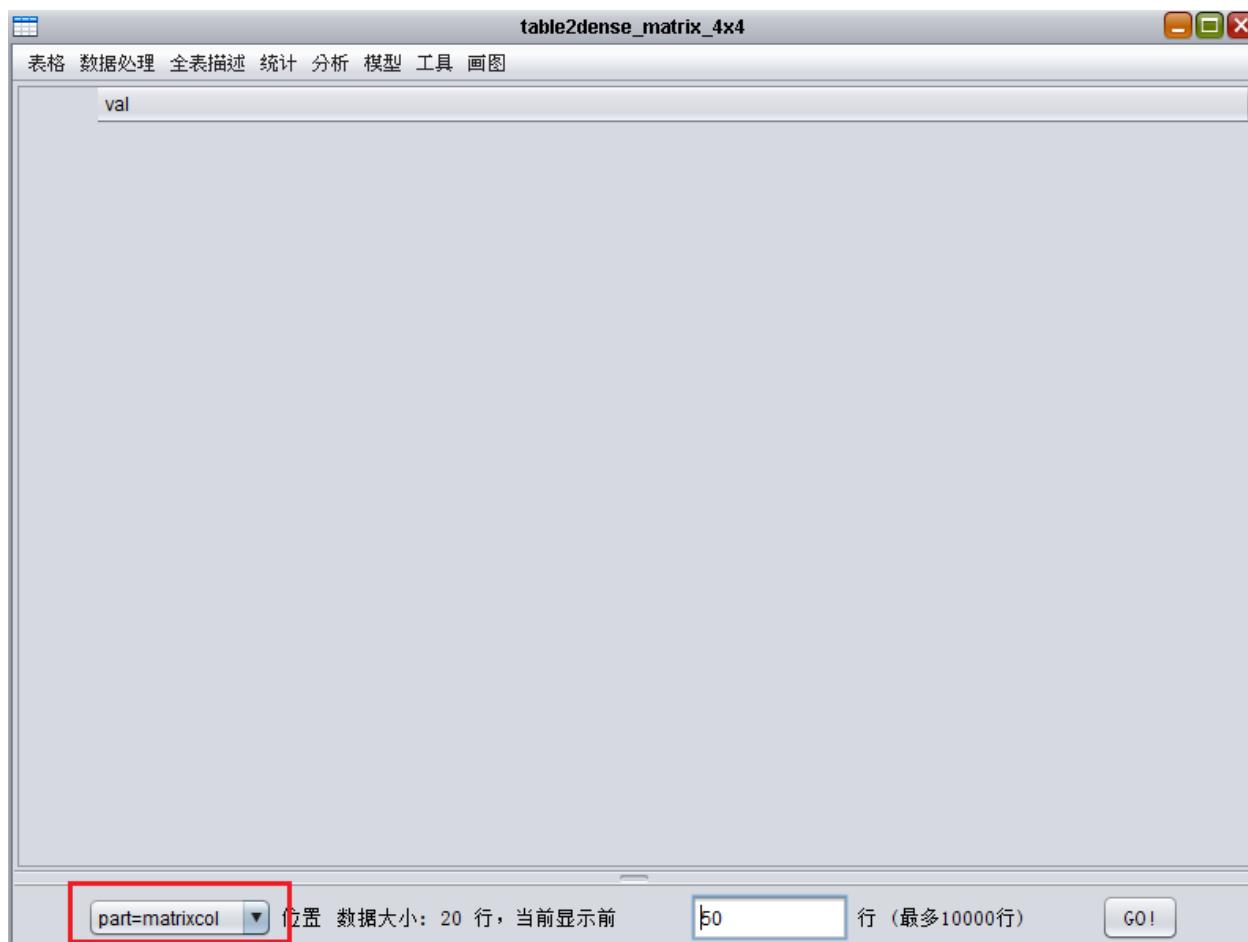
table2dense_matrix_4x4

表格 数据处理 全表描述 统计 分析 模型 工具 画图

	val
0	
1	
2	
3	

part=matrixrow ▾ 位置 数据大小: 20 行, 当前显示前 50 行 (最多10000行) GO!

- matrixcol: 空, 如下图:



2.3 Xlib稀疏矩阵表

稀疏矩阵是指非零元素的个数远远小于矩阵元素的总数，并且非零元素的分布没有规律的矩阵。XLib稀疏矩阵表是指存储稀疏矩阵的0dps表。XLib稀疏矩阵表包括四个分区：

- matrixconf: 矩阵类型为3, 代表稀疏矩阵;
- matrixdata: 按先行后列存储全部数据;
- matrixrow: 存储每一行的开头在matrixdata文件里的偏移;
- matrixcol: 存储matrixdata文件里每个非零元素的列指标。

示例, table2sparse_matrix_4x4(与表table_4x4对应):

- matrixconf: 矩阵类型为3, 代表稀疏矩阵, 如下图:

table2sparse_matrix_4x4

表格 数据处理 全表描述 统计 分析 模型 工具 画图

	val
0	3
1	4
2	4
3	9

part=matrixconf ▾ 位置 数据大小: 27 行, 当前显示前 50 行 (最多10000行) GO!

- matrixdata: 按先行后列存储全部数据, 如下图:

table2sparse_matrix_4x4

表格 数据处理 全表描述 统计 分析 模型 工具 画图

	val
0	13.58
1	49.6
2	22.31
3	12.31
4	47.8
5	28.82
6	16.18
7	3.54
8	43.2

part=matrixdata ▾ 位置 数据大小: 27 行, 当前显示前 50 行 (最多10000行) GO!

- matrixrow: 存储每一行的开头在matrixdata文件里的偏移, 如下图:

table2sparse_matrix_4x4

表格 数据处理 全表描述 统计 分析 模型 工具 画图

	val
0	0
1	2
2	5
3	7
4	9

part=matrixrow ▾ 位置 数据大小: 27 行, 当前显示前 50 行 (最多10000行) GO!

- matrixcol: 存储matrixdata文件里每个非零元素的列指标, 如下图:

table2sparse_matrix_4x4

表格 数据处理 全表描述 统计 分析 模型 工具 画图

	val
0	0
1	3
2	0
3	2
4	3
5	0
6	2
7	1
8	3

part=matrixcol ▾ 位置 数据大小: 27 行, 当前显示前 50 行 (最多10000行) GO!

2.4 三元组表

三元组表是用三元组的形式表示稀疏矩阵，三元组表一共有三列，分别表示行，列，值。其中值的类型为 Double或Long。例如表示(用户，电影，打分)，数据应该是(“小明”，“肖申克”，9)。

示例，三元组表sparse_matrix_4x4_triple(与XLib稀疏矩阵表table2sparse_matrix_4x4)，存储的是稀疏矩阵的非零元素的行名、列名及值，如下图：

sparse_matrix_4x4_triple - 数据表浏览

表格 数据处理 全表描述 统计 分析 模型 工具 画图

	row_index_row	col_index_col	value
0	行1	列0	22.31
1	行1	列2	12.31
2	行1	列3	47.8
3	行2	列0	28.82
4	行2	列2	16.18
5	行3	列1	3.54
6	行3	列3	43.2
7	行0	列0	13.58
8	行0	列3	49.6

Whole Table ▾ 位置 数据大小: 9 行, 当前显示前 50 行 (最多10000行) GO!

行名与行下标的映射表rowmap，它共两列，存储行下标与对应的行名，如下图：

rowmap - 数据表浏览

表格 数据处理 全表描述 统计 分析 模型 工具 画图

	row_index	row_name
0	0	行0
1	1	行1
2	2	行2
3	3	行3

Whole Table ▾ 位置 数据大小: 4 行, 当前显示前 50 行 (最多10000行) GO!

列名与列下标的映射表colmap, 它共两列, 存储列下标与对应的列名, 如下图:

The screenshot shows a software interface titled "colmap - 数据表浏览". The main area displays a table with two columns: "col_index" and "col_name". The "col_index" column contains values 0, 1, 2, and 3. The "col_name" column contains values "列0", "列1", "列2", and "列3". Below the table, there is a toolbar with buttons for "Whole Table", "位置" (Position), "数据大小: 4 行, 当前显示前" (Data size: 4 rows, current display before), a numeric input field "50", a button "行 (最多10000行)" (Rows (up to 10000)), and a "GO!" button.

col_index	col_name
0	列0
1	列1
2	列2
3	列3

2.5 索引三元组表

索引三元组表是三元组表的一种特殊形式, 如果三元组表中行和列是索引, 则是索引三元组表。索引三元组表的类型是(Long, Long, Double|Long), 其中行索引和列索引的小标都是从0开始的。例如(用户id, 电影id, 打分), 数据格式是(1, 12, 9)。

示例, 索引三元组表sparse_martix_4x4_index_triple(与XLib稀疏矩阵表table2sparse_martix_4x4相对应), 存储的是稀疏矩阵的非零元素的行下标、列下表以及非零值, 如下图:

The screenshot shows a software interface titled "sparse_martix_4x4_index_triple - 数据表浏览". The main area displays a table with three columns: "row_index", "col_index", and "value". The "row_index" column contains values 0, 1, 2, 3, 4, 5, 6, 7, and 8. The "col_index" column contains values 0, 3, 0, 2, 1, 3, 0, 2, 1, and 3. The "value" column contains values 13.58, 49.6, 22.31, 12.31, 47.8, 28.82, 16.18, 3.54, and 43.2. Below the table, there is a toolbar with buttons for "Whole Table", "位置" (Position), "数据大小: 9 行, 当前显示前" (Data size: 9 rows, current display before), a numeric input field "50", a button "行 (最多10000行)" (Rows (up to 10000)), and a "GO!" button.

row_index	col_index	value
0	0	13.58
1	3	49.6
2	0	22.31
3	2	12.31
4	1	47.8
5	2	28.82
6	2	16.18
7	1	3.54
8	3	43.2

2.6 KV表

KV表是表里每个数据字段格式都是key: value, key: value...的形式, 其中key, value之间的分隔符和多对KV之间的分隔符可以自定义。key是指列名, value是指存储在该行key列的元素值。

- KV表的列可为单列, 也可为多列, 同一行, 各列的KV对不能重复。
- 对于一个KV表, 需要有相对应的map表才能完整的表示这个数据。

map表有以下特征:

- 由两列组成, 一列为string类型, 表示列名。另一列为bigint类型, 表示列号。

- 和kv表成对出现。
- map表中的列名对应于KV表中的key。

示例1: KV对

```
col1:1 col2:2 col3:3
```

- col1、col2、col3是key, 代表列名。
- 1、2、3是value, 代表key对应的值。

示例2: KV表

KV表为两列kvstr1, kvstr2的示例, 如下图:

	kvstr1	kvstr2
0	col1:0	
1	col1:1,col2:2	col3:3
2	col3:1,col2:1	col1:2
3	col4:2,col5:1	col3:2

选中单元格的数据(col1:1, col2:2)说明:

- KV对是由, 分割;
- col1:1代表第1行col1列的值为1;
- col2:2代表第1行col2列的值为2。

和KV表中例子对应的map表为:

	kvkey	kvidx
0	col1	0
1	col2	1
2	col3	2
3	col4	3
4	col5	4

KV表加上map表即可表示一个完整的矩阵, 组合KV表和map表中的两个例子, 矩阵为:

col1	col2	col3	col4	col5
0	0	0	0	0
1	2	3	0	0
2	1	1	0	0
0	0	2	2	1

示例3, KV表dense_martix_4x4_kv(与XLib稠密矩阵表table2dense_martix_4x4和列下标列名映射表colmap相对应), 如下图:

dense_martix_4x4_kv - 数据表浏览	
表格 数据处理 全表描述 统计 分析 模型 工具 画图	
col0	
0	列0:13.580000000000001,列1:0,列2:0,列3:49.600000000000014,
1	列0:22.309999999999987,列1:0,列2:12.310000000000005,列3:47.79999999999972,
2	列0:28.820000000000003,列1:0,列2:16.179999999999997,列3:0,
3	列0:0,列1:3.540000000000004,列2:0,列3:43.200000000000028,

选中单元格元素: 列0:13.580000000000001,列1:0,列2:0,列3:49.600000000000014, 代表第0行的四列元素, KV对间用, 分割。列0:13.580000000000001代表一个KV对, key为列0, value为13.580000000000001, 这个KV对代表第0行的列0列的值为13.580000000000001。

示例4, KV表sparse_martix_4x4_kv(与XLib稀疏矩阵表table2sparse_martix_4x4和列下标列名映射表colmap相对应), 如下图:

sparse_martix_4x4_kv - 数据表浏览	
表格 数据处理 全表描述 统计 分析 模型 工具 画图	
col0	
0	列0:13.580000000000001,列3:49.600000000000014,
1	列0:22.309999999999987,列2:12.310000000000005,列3:47.79999999999972,
2	列0:28.820000000000003,列2:16.179999999999997,
3	列0:0,列1:3.540000000000004,列2:0,列3:43.200000000000028,

2.7 索引KV表

KV表里如果key是索引id, 则是索引KV表。索引KV表不需要对应map表。

示例1, 索引KV对:

1:ab 2:bc 3:cd

- 1、2、3是key, 代表列下标。
- ab、bc、cd是value, 代表key对应的值。

示例2, 索引KV表dense_martix_4x4_kv_index(与XLib稠密矩阵表table2dense_martix_4x4相对应), 如下图:

dense_martix_4x4_kv_index - 数据表浏览	
表格 数据处理 全表描述 统计 分析 模型 工具 画图	
col0	
0	0:13.580000000000001, 1:0, 2:0, 3:49.600000000000014,
1	0:22.30999999999987, 1:0, 2:12.310000000000005, 3:47.79999999999972,
2	0:28.820000000000003, 1:0, 2:16.17999999999997, 3:0,
3	0:0, 1:3.540000000000004, 2:0, 3:43.200000000000028,

选中单元格元素: 0:13.580000000000001, 1:0, 2:0, 3:49.600000000000014, 代表第0行的四列元素, KV对间用, 分割。 0:13.580000000000001代表一个KV对, key为0, value为13.580000000000001, 这个KV对代表第0行的第0列的值为13.580000000000001。

示例3, 索引KV表sparse_martix_4x4_kv_index (与XLib稀疏矩阵表table2sparse_martix_4x4对应), 如下图:

sparse_martix_4x4_kv_index - 数据表浏览	
表格 数据处理 全表描述 统计 分析 模型 工具 画图	
col0	
0	0:13.580000000000001, 3:49.600000000000014,
1	0:22.30999999999987, 2:12.310000000000005, 3:47.79999999999972,
2	0:28.820000000000003, 2:16.17999999999997,
3	1:3.540000000000004, 3:43.200000000000028,

2.8 表达式

Xlab提供了一些表达式与函数, 方便用户在脚本和界面中使用, 包括:

IN, RLIKE, LIKE, +, -, *, /, %, ^, =, ==, !=, <>, IS NULL, IS NOT NULL , <, <=, >, >=, &&, AND, ||, OR, !, NOT, +, -, &, |, <<, >>, reverse(), len()

表达式及函数的使用说明, 详见附录。

2.9 脚本语言

我们使用Python作为脚本语言, 详细说明可参考Python的简明教程。

这里我们摘要列出最基本的内容, 方便用户快速入门。

注意事项:

- 任何在#符号右面的内容都是注释。

2.9.1 字符串

- 使用单引号(‘)

你可以用单引号指示字符串，就如同’ Quote me on this’ 这样。所有的空白，即空格和制表符都照原样保留。

- 使用双引号(”)

在双引号中的字符串与单引号中的字符串的使用完全相同，例如” What’s your name?” 。

- 使用三引号(‘’ 或 “”)

利用三引号，你可以指示一个多行的字符串。你可以在三引号中自由的使用单引号和双引号。

示例：

```
'''This is a multi-line string. This is the first line.  
This is the second line.  
"What's your name?", I asked.  
He said "Bond, James Bond."  
'''
```

- 转义符 假设你想要在一个字符串中包含一个单引号(‘)，那么你该怎么指示这个字符串？例如，这个字符串是What’s your name?。你肯定不会用’ What’s your name?’ 来指示它，因为Python会弄不明白这个字符串从何处开始，何处结束。所以，你需要指明单引号而不是字符串的结尾。可以通过 转义符 来完成这个任务。你用’ 来指示单引号——注意这个反斜杠。现在你可以把字符串表示为’ What’s your name?’ 。
- 另一个表示这个特别的字符串的方法是” What’s your name?” ，即用双引号。类似地，要在双引号字符串中使用双引号本身的时候，也可以借助于转义符。另外，你可以用转义符\来指示反斜杠本身。
- 在一个字符串中，行末的单独一个反斜杠表示字符串在下一行继续，而不是开始一个新的行。

示例：

```
"This is the first sentence. \  
This is the second sentence."
```

等价于

```
"This is the first sentence. This is the second sentence."
```

2.9.2 标识符的命名

变量是标识符的例子。 标识符 是用来标识 某样东西 的名字。在命名标识符的时候，你要遵循这些规则：

- 标识符的第一个字符必须是字母表中的字母(大写或小写)或者一个下划线(‘ _ ’)。

- 标识符名称的其他部分可以由字母(大写或小写)、下划线(‘_’)或数字(0-9)组成。
- 标识符名称是对大小写敏感的。例如, myname和myName不是一个标识符。注意前者中的小写n和后者中的大写N。
- 有效 标识符名称的例子有i、__my_name、name_23和a1b2_c3。
- 无效 标识符名称的例子有2things、this is spaced out和my-name。

2.9.3 变量

我们看一下如何使用变量和字面意义上的常量。

示例:

```
# Filename : var.py
i = 5
print i
i = i + 1
print i

s = '''This is a multi-line string.
This is the second line.'''
print s
```

输出

```
5
6
This is a multi-line string.
This is the second line.
```

下面来说明一下这个程序如何工作。首先我们使用赋值运算符(=)把一个字面意义上的常数5赋给变量i。这一行称为一个语句。语句声明需要做某件事情，在这个地方我们把变量名i与值5连接在一起。接下来，我们用print语句打印i的值，就是把变量的值打印在屏幕上。然后我们对i中存储的值加1，再把它存回i。我们打印它时，得到期望的值6。

类似地，我们把一个字面意义上的字符串赋给变量s然后打印它。

注意事项:

- 使用变量时只需要给它们赋一个值，不需要声明或定义数据类型。

2.9.4 逻辑行与物理行

物理行是你在编写程序时所看见的。逻辑行是Python看见的单个语句。Python假定每个物理行 对应一个逻辑行。逻辑行的例子如print ‘Hello World’ 这样的语句——如果它本身就是一行(就像你在编辑器中看

到的那样), 那么它也是一个物理行。默认地, Python希望每行都只使用一个语句, 这样使得代码更加易读。

若要在多个物理行中写一个逻辑行, 可以使用反斜杠。

示例:

```
s = 'This is a string. \
This continues the string.'
```

注意事项:

- 如果你想要在一个物理行中使用多于一个逻辑行, 那么你需要使用分号(;)来特别地标明这种用法。分号表示一个逻辑行/语句的结束。

2.9.5 控制流

在Python中有三种控制流语句——if、for和while。

2.9.6 if语句

使用if语句的示例:

```
number = 23
guess = 12

if guess == number:
    print 'Congratulations, you guessed it.' # New block starts here
    print "(but you do not win any prizes!)" # New block ends here
elif guess < number:
    print 'No, it is a little higher than that' # Another block
    # You can do whatever you want in a block ...
else:
    print 'No, it is a little lower than that'
    # you must have guess > number to reach here
```

注意事项:

- 我们使用了缩进层次来告诉Python每个语句分别属于哪一个块。这就是为什么缩进在Python如此重要的原因。我希望你能够坚持“每个缩进层一个制表符”的规则。
- if语句在结尾处包含一个冒号——我们通过它告诉Python下面跟着一个语句块。我们在这里使用的是 elif从句, 它事实上把两个相关联的if else-if else语句合并为一个if-elif-else语句。elif和else从句都必须在逻辑行结尾处有一个冒号, 下面跟着一个相应的语句块(当然还包括正确的缩进)。

2.9.7 while语句

只要在一个条件为真的情况下, while语句允许你重复执行一块语句。while语句是所谓 循环 语句的一个例子。

示例:

```
number = 23
guess = 12
running = True

while running:
    guess += 1
    if guess == number:
        print 'Congratulations, you guessed it.'
        running = False # this causes the while loop to stop
    elif guess < number:
        print 'No, it is a little higher than that'
    else:
        print 'No, it is a little lower than that'

print 'Done'
```

2.9.8 for循环

for..in是另外一个循环语句。在C/C++中, 如果你想要写for (int i = 0; i < 5; i++), 那么用Python, 你写成for i in range(0, 5)。

示例:

```
for i in range(1, 5):
    print i
```

输出

```
1
2
3
4
```

2.9.9 break语句

break语句是用来 终止 for或while循环语句的, 即哪怕循环条件没有称为False或序列还没有被完全递归, 也停止执行循环语句。

2.9.10 continue语句

continue语句被用来告诉Python跳过当前for或while循环块中的剩余语句，然后 继续 进行下一轮循环。

2.9.11 函数

函数是重用的程序段。它们允许你给一块语句一个名称，然后你可以在你的程序的任何地方使用这个名称任意多次地运行这个语句块。这被称为 调用 函数。

函数通过def关键字定义。def关键字后跟一个函数的 标识符 名称，然后跟一对圆括号。圆括号之中可以包括一些变量名，该行以冒号结尾。接下来是一块语句，它们是函数体。

示例：

```
def sayHello():
    print 'Hello World!' # block belonging to the function

sayHello() # call the function
```

参数在函数定义的圆括号对内指定，用逗号分割。当我们调用函数的时候，我们以同样的方式提供值。

示例：

```
def printMax(a, b):
    if a > b:
        print a, 'is maximum'
    else:
        print b, 'is maximum'

printMax(3, 4) # directly give literal values

x = 5
y = 7

printMax(x, y) # give variables as arguments
```

输出

```
4 is maximum
7 is maximum
```

return语句用来从一个函数返回即跳出函数。我们也可选从函数返回一个值。

示例：

```
def maximum(x, y):
    if x > y:
        return x
    else:
```

```
    return y  
  
print maximum(2, 3)
```

输出

3

第 3 章

表的操作

在XLab中，表的操作的“表”是指ODPS表(Table)。表的操作是指在XLab的脚本编辑窗口中通过调用函数对ODPS表(Table)进行操作，支持的操作包括：判断表是否存在、创建表、删除表、显示指定工程下的所有表名、获取表名、获取表注释、获取表所有者、获取表创建时间、获取表最后修改时间、判断表是否为虚拟视图、获取所有列名、获取所有列注释、获取所有列类型、通过列名获取列类型、获取表总记录数、获取表指定分区的记录数、获取表所有分区。

上述关于表(Table)支持的函数，可用help命令查看，如下示例：

```
help(Table)
```

注解： 补全以及函数提示的快捷键：Ctrl+p。

3.1 函数

表的操作支持以下函数，具体使用方法，可用help命令查看，示例如下：

```
help(Table.exists)
help(Table.create)
help(Table.drop)
help(Table.dropPartitions)
help(Table.listTableNames)
help(Table.setGlobalLifeCycle)
help(Table.getGlobalLifeCycle)
```

```
help(Table.getTableName)
help(Table.getComment)
help(Table.getProject)
help(Table.getOwner)
help(Table.getLifeCycle)
help(Table.getCreateTime)
help(Table.getLastModifiedTime)
help(Table.isVirtualView)
help(Table.getColNames)
help(Table.getColComments)
help(Table.getColTypes)
help(Table.getColTypeByName)
help(Table.getRecordCount)
help(Table.getPartitionRecordCount)
help(Table.getPartitions)
```

3.1.1 判断表是否存在

```
def exists(tableName)
```

参数:

- tableName: 表名, 字符串

返回:

- True, 表示存在; False, 表示不存在。

示例1:

```
Table.exists("tableName")
```

3.1.2 创建表

```
def create(tableName, colNames, colTypes)
```

参数:

- tableName: 表名, 字符串
- colNames: 列名, 字符数组
- colTypes: 列类型, 字符数组

返回:

- 无。

示例1:

```
Table.create("tableName", ["col0", "col1"], ["string", "bigint"])
```

3.1.3 删除表

```
def drop(tableName)
```

参数:

- tableName: 表名, 字符串

返回:

- 无。

示例1:

```
Table.drop("tableName")
```

3.1.4 删除表分区

```
def dropPartitions(tableName, partitions):
```

参数:

- tableName: 表名, 字符串
- partitions: 分区列表, list类型

返回:

- 无

示例:

```
Table.dropPartitions("tableName", ["p=1", "p=2"])
```

3.1.5 显示指定工程下的所有表名

```
def listTableNames(projectName=None)
```

参数:

- projectName: (可选) 工程名, 字符串, 默认为当前工程

返回:

- 指定工程下的所有表名

示例1:

```
Table.listTableNames(), 显示当前project的所有表
```

3.1.6 设置全局生命周期

```
def setGlobalLifeCycle(lifeCycle)
```

参数:

- lifeCycle: 生命周期, 单位为天, Long型。值为Table.UNVALIDGLOBALLIFCYCLE时, xlib产出表不设置生命周期。

返回:

- 无

示例:

```
Table.setGlobalLifeCycle(37231)
```

3.1.7 获取全局生命周期

```
def getGlobalLifeCycle()
```

参数:

- 无

返回:

- 全局生命周期的值

示例:

```
Table.getGlobalLifeCycle()
```

3.1.8 获取表名

```
def getTableName(self)
```

参数:

- self: 不必赋值

返回:

- 表名

示例1:

```
table = Table("iris")  
  
print table.getTableName();
```

3.1.9 获取表注释

```
def getComment(self)
```

参数:

- self: 不必赋值

返回:

- 表注释

示例1:

```
table = Table("iris")  
  
print table.getComment();
```

3.1.10 获取表的Project名

```
def getProject(self)
```

参数:

- self: 不必赋值

返回:

- Project名

示例:

```
table = Table("iris")  
  
print table.getProject()
```

3.1.11 获取表所有者

```
def getOwner(self)
```

参数:

- self: 不必赋值

返回:

- 表所有者

示例1:

```
table = Table("iris")  
  
print table.getOwner();
```

3.1.12 获取表的生命周期

```
def getLifeCycle(self)
```

参数:

- self: 不必赋值

返回:

- 表的生命周期

示例:

```
table = Table("iris")  
  
print table.getProject()
```

3.1.13 获取表创建时间

```
def getCreateTime(self)
```

参数:

- self: 不必赋值

返回:

- 表创建时间

示例1:

```
table = Table("iris")  
  
print table.getCreateTime();
```

3.1.14 获取表最后修改时间

```
def getLastModifiedTime(self)
```

参数:

- self: 不必赋值

返回:

- 表最后修改时间

示例1:

```
table = Table("iris")
print table.getLastModifiedTime();
```

3.1.15 判断表是否为虚拟视图

```
def isVirtualView(self)
```

参数:

- self: 不必赋值

返回:

- 布尔值, False: 表不是虚拟视图 True: 表是虚拟视图

示例1:

```
table = Table("iris")
print table.isVirtualView();
```

3.1.16 获取所有列名

```
def getColNames(self)
```

参数:

- self: 不必赋值

返回:

- 所有列名

示例1:

```
table = Table("iris")  
  
print table.getColNames();
```

3.1.17 获取所有列注释

```
def getColComments(self)
```

参数:

- self: 不必赋值

返回:

- 所有列注释

示例1:

```
table = Table("iris")  
  
print table.getColComments();
```

3.1.18 获取所有列类型

```
def getColTypes(self)
```

参数:

- self: 不必赋值

返回:

- 所有列类型

示例1:

```
table = Table("iris")  
  
print table.getColTypes();
```

3.1.19 通过列名获取列类型

```
def getColTypeByName(self, colName)
```

参数:

- self: 不必赋值

- colName: 列名

返回:

- 列类型

示例1:

```
table = Table("iris")
print table.getTypeByName("category");
```

3.1.20 获取表总记录数

```
def getRecordCount(self)
```

参数:

- self: 不必赋值

返回:

- 表总记录数

示例1:

```
table = Table("iris")
print table.getRecordCount();
```

3.1.21 获取表指定分区的记录数

```
def getPartitionRecordCount(self, partition)
```

参数:

- self: 不必赋值
- partition: 表的分区, key value用=分割的字符串

返回:

- 表总记录数

示例1:

```
table = Table("bayes_model")
print table.getPartitionRecordCount("pt=conf");
```

3.1.22 获取表所有分区

```
def getPartitions(self)
```

参数:

- self: 不必赋值

返回:

- 表所有分区

示例1:

```
table = Table("bayes_model")  
  
print table.getPartitions();
```

第 4 章

基本统计

统计学是一门很古老的科学，一般认为其学理研究始于古希腊的亚里斯多德时代，迄今已有两千三百多年的历史。这里的基本统计主要包含计算基本统计量(全表基本统计量和单列基本统计量)、直方图、协方差、相关矩阵、top100、bottom10、频率、百分位、全表统计汇总、散布图矩阵；按条件统计对比、按条件数据分组、按分区统计对比、扩展直方图、交叉表、对比交叉表、排行榜等。

基本统计量共22种，包括：总个数，有效值个数，缺失值个数，和，平方和，立方和，最小值，最大值，极差，均值，方差，标准差，变异系数，标准误差，偏度，峰度，二阶矩，三阶矩，四阶矩，二阶中心距，三阶中心距，四阶中心距。这些统计量的英文名对应表如下所示：

- countTotal: 总个数
- count: 有效值个数
- countMissValue: 缺失值个数
- sum: 和
- sum2: 平方和
- sum3: 立方和
- min: 最小值
- max: 最大值
- range: 极差
- mean: 均值
- variance: 方差

- standardDeviation: 标准差
- cv: 变异系数
- standardError: 标准误差
- skewness: 偏度
- kurtosis: 峰度
- moment2: 二阶矩
- moment3: 三阶矩
- moment4: 四阶矩
- centralMoment2: 二阶中心距
- centralMoment3: 三阶中心距
- centralMoment4: 四阶中心距

注解： 补全以及函数提示的快捷键：Ctrl+p。

4.1 全表基本统计

全表基本统计量是指对全表数据进行基本统计。执行此统计的方式有两种：界面和函数，详细介绍见下文。

4.1.1 界面

点击菜单→全表描述→全表基本统计量，以kddcup数据为例，执行界面，如下图：



计算过程中可以点击取消按钮取消计算同时终止云端JOB。运行结果如下:

kddcup:全表基本统计量											
基本统计 画图	durationg	protocol_t...	serviceg	flagg	src_bytessg	dst_bytessg	landg	wrong_fra...	urgeng	hotg	
countI...	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898
count	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898431.0	4898
countM...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
countN...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
countP...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
countN...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
sum	2.3680...	NaN	NaN	NaN	8.9867...	5.3570...	NaN	3178.0	39.0	6092	
sum2	2.5743...	NaN	NaN	NaN	4.3414...	2.0379...	NaN	8998.0	255.0	1078	
sum3	4.9941...	NaN	NaN	NaN	4.8594...	2.3760...	NaN	26458.0	2937.0	2.54	
sum4	1.3452...	NaN	NaN	NaN	5.8883...	2.9957...	NaN	78838.0	39195.0	6.68	
min	0.0	NaN	NaN	NaN	0.0	0.0	NaN	0.0	0.0	0.0	
max	58329.0	NaN	NaN	NaN	1.3799...	1.3099...	NaN	3.0	14.0	77.0	
range	58329.0	NaN	NaN	NaN	1.3799...	1.3099...	NaN	3.0	14.0	77.0	
mean	48.342...	NaN	NaN	NaN	1834.6...	1093.6...	NaN	6.4877...	7.9617...	0.01	
variance	523206...	NaN	NaN	NaN	8.8629...	4.1604...	NaN	0.0018...	5.2057...	0.21	
standa...	723.32...	NaN	NaN	NaN	941431...	645012...	NaN	0.0428...	0.0072...	0.46	
cv	14.962...	NaN	NaN	NaN	513.14...	589.79...	NaN	66.053...	906.22...	37.7	
standa...	0.3268...	NaN	NaN	NaN	425.36...	291.43...	NaN	1.9362...	3.2599...	2.11	
skewness	26.739...	NaN	NaN	NaN	1188.9...	1807.5...	NaN	68.584...	1596.3...	50.2	
kurtosis	993.04...	NaN	NaN	NaN	153030...	353329...	NaN	4764.8...	295261...	2811	
moment2	525542...	NaN	NaN	NaN	8.8629...	4.1604...	NaN	0.0018...	5.2057...	0.22	
moment3	1.0195...	NaN	NaN	NaN	9.9204...	4.8506...	NaN	0.0054...	5.9957...	5.18	
moment4	2.7462...	NaN	NaN	NaN	1.2020...	6.1158...	NaN	0.0160...	0.0080...	136.	
centra...	523205...	NaN	NaN	NaN	8.8629...	4.1604...	NaN	0.0018...	5.2057...	0.21	
centra...	1.0119...	NaN	NaN	NaN	9.9204...	4.8506...	NaN	0.0053...	5.9957...	5.18	
centra...	2.7266...	NaN	NaN	NaN	1.2020...	6.1157...	NaN	0.0160...	0.0080...	136.	
colName	durationg	protoc...	serviceg	flagg	src_by...	dst_by...	landg	wrong_...	urgeng	hotg	

4.1.2 函数

```
def simpleSummary(inputTableName):
```

参数:

- inputTableName: 输入表名.

返回:

- SummaryResultTable.

返回值SummaryResultTable srt, 通过列名可以取出每一列的基本统计量等, 基本统计量如下:

```
src = srt.col("colName")
src.countTotal: 总个数
src.count: 有效值个数
src.countMissValue: 缺失值个数
src.sum: 和
src.sum2: 平方和
src.sum3: 立方和
src.min: 最小值
src.max: 最大值
```

```

src.range: 极差
src.mean: 均值
src.variance: 方差
src.standardDeviation: 标准差
src.cv: 变异系数
src.standardError: 标准误差
src.skewness: 偏度
src.kurtosis: 峰度
src.moment2: 二阶矩
src.moment3: 三阶矩
src.moment4: 四阶矩
src.centralMoment2: 二阶中心距
src.centralMoment3: 三阶中心距
src.centralMoment4: 四阶中心距

```

示例：

```

simpleSrt = Statistics.simpleSummary( 'kddcup' )
print simpleSrt

```

注意事项：

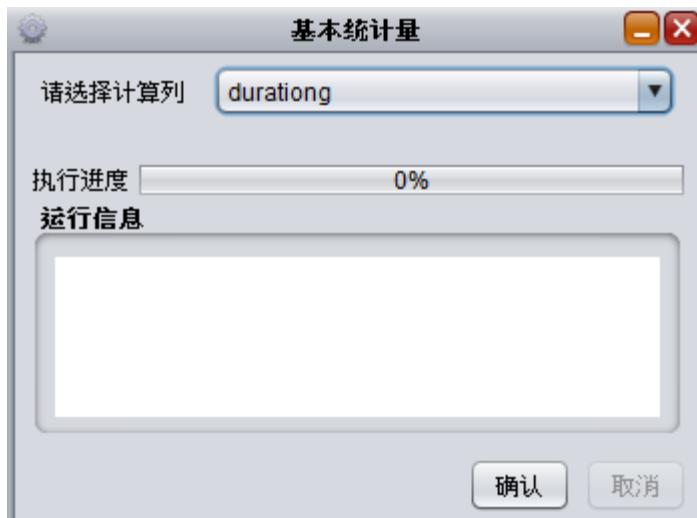
- 如果列类型是string，那么除了count, countTotal, countMissValue, countNanValue, countPositiveInfinity, countNegativeInfinity，其余都是NAN.
- 如果列类型是datetime，那么除了count, countTotal, countMissValue, countNanValue, countPositiveInfinity, countNegativeInfinity, min, max其他都是NAN.

4.2 单列基本统计量

单列基本统计量是指对单列数据进行基本统计。执行此统计的方式有两种：界面和函数，详细介绍见下文。

4.2.1 界面

点击菜单→全表描述→单列基本统计量，执行界面，如下图：



或者在表格上选取一格，右键弹出菜单选择基本统计量，如下图：

	durationng	protocol_t...	serviceg	flagg	src_bytesg	dst_bytesg	landg	wrong_fra...	urgengt	hotg
0	0	tcp	http	SF	215	45076	0	0	0	0
1	0	tcp	http	SF	162	4528	0	0	0	0
2	0	tcp	http	SF	236	1228	0	0	0	0
3	0	tcp	http	SF	233	2032	0	0	0	0
4	0	tcp	http	SF	239	486	0	0	0	0
5	0	tcp	http	SF	238	1282	0	0	0	0
6	0	tcp	http	SF	235	1337	0	0	0	0
7	0	tcp	http	SF	234	1364	0	0	0	0
8	0	tcp	http	SF	239	1295	0	0	0	0
9	0	tcp	http	SF	181	548	0	0	0	0
10	0	tcp	http	SF	184	128	0	0	0	0
11	0	tcp	http	SF	185	903	0	0	0	0
12	0	tcp	http	SF	239	129	0	0	0	0
13	0	tcp	http	SF	181	548	0	0	0	0
14	0	tcp	http	SF	236	122	0	0	0	0
15	0	tcp	http	SF	233	203	0	0	0	0
16	0	tcp	http	SF	238	128	0	0	0	0
17	0	tcp	http	SF	235	130	0	0	0	0
18	0	tcp	http	SF	234	136	0	0	0	0
19	0	tcp	http	SF	239	486	0	0	0	0
20	0	tcp	http	SF	185	903	0	0	0	0
21	0	tcp	http	SF	184	124	0	0	0	0
22	0	tcp	http	SF	181	5450	0	0	0	0
23	0	tcp	http	SF	239	1295	0	0	0	0
24	0	tcp	http	SF	236	1228	0	0	0	0
25	0	tcp	http	SF	233	2032	0	0	0	0
26	0	tcp	http	SF	239	486	0	0	0	0
27	0	tcp	http	SF	238	1282	0	0	0	0
28	0	tcp	http	SF	234	1364	0	0	0	0

点击菜单后点击确定进行计算，运行结果如下：

基本统计量@dst_bytesg	
基本统计 画图	
Statistics	dst_bytesg
countTotal	4898431.0
count	4898431.0
countMissValue	0.0
countNanValue	0.0
countPositiveInfinity	0.0
countNegativeInfinity	0.0
sum	5.357035893E9
sum2	2.03795313667757312E18
sum3	2.376054865106495E27
sum4	2.9957855644333056E36
min	0.0
max	1.309937401E9
range	1.309937401E9
mean	1093.6228137132073
variance	4.160409106797294E11
standardDeviation	645012.333742332
cv	589.7941462580723
standardError	291.4335333429357
skewness	1807.5666634006868
kurtosis	3533295.4482860127
moment2	4.160420217570837E11
moment3	4.8506447576917895E20
moment4	6.115806396850962E29
centralMoment2	4.1604082574622504E11
centralMoment3	4.850631107926556E20
centralMoment4	6.11578517777744E29
colName	dst_bytesg

4.2.2 函数

```
def simpleSummary(inputTableName, summaryColNames=None):
```

参数:

- inputTableName: 输入表名.
- summaryColNames: (可选)需要计算的列名列表

返回:

- SummaryResultTable.

返回值说明见全表统计量函数。

示例: 基本统计

```
simpleSrt = Statistics.simpleSummary('kddcup', summaryColNames=['flagg', 'countg'])
print simpleSrt
print simpleSrt.col('flagg').min
```

4.3 直方图

直方图(Histogram)又称质量分布图,柱状图,是一种统计报告图,由一系列高度不等的纵向条纹或线段表示数据分布的情况。一般用横轴表示数据类型,纵轴表示分布情况。该操作是指对单列数据显示直方图,执行方式包括两种:界面和函数。

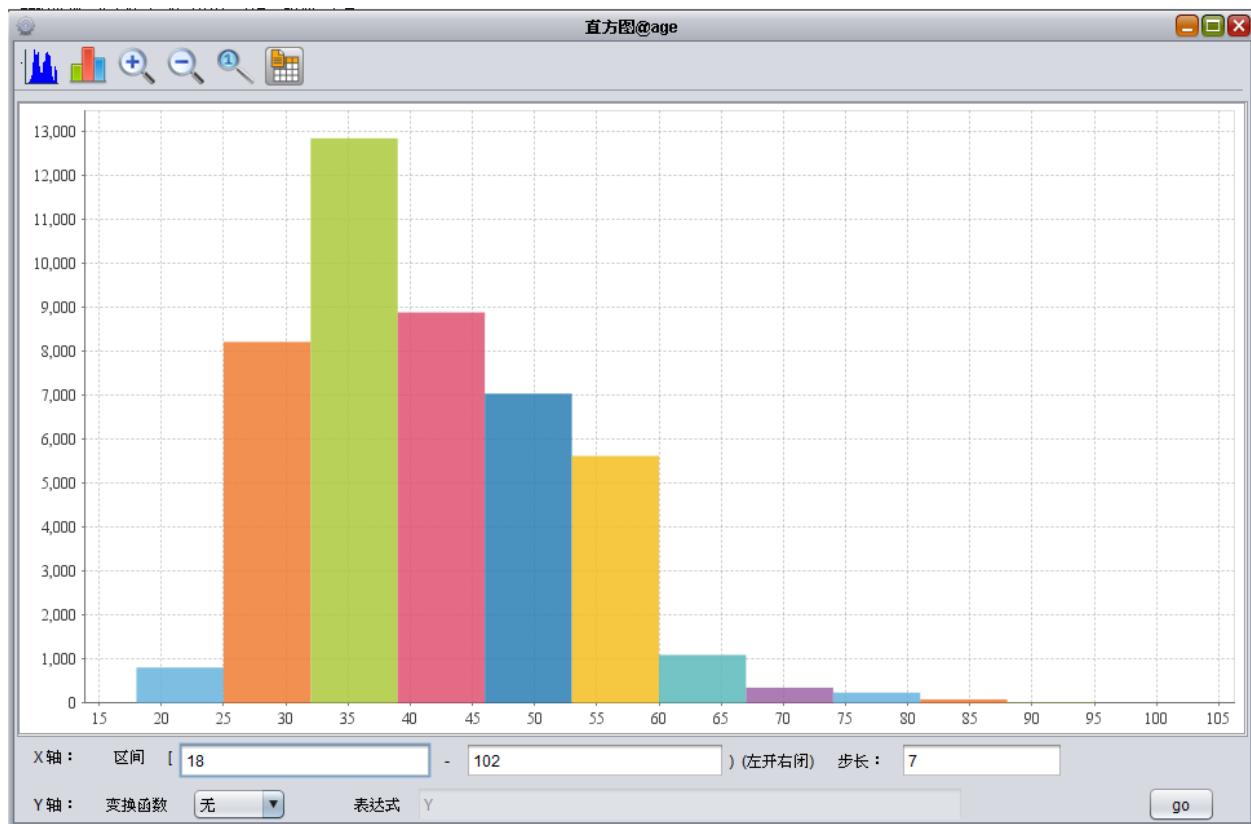
4.3.1 界面

该操作对于单列数据显示直方图,点击菜单,全表描述→直方图,显示参数窗体如下:



或者在表格上选取一列,右键弹出菜单选择直方图,参考单列基本统计量功能。

点击菜单后点击确定进行计算运行结果如下:



直方图界面分为三个部分：

上方的工具栏：



以推荐的最小步长显示直方图



以推荐的最大步长显示直方图



将当前步长减小显示直方图



将当前步长增大显示直方图



恢复原区间和步长

上述操作均针对下方的当前区间栏进行缩放操作。



查看当前直方图的数据

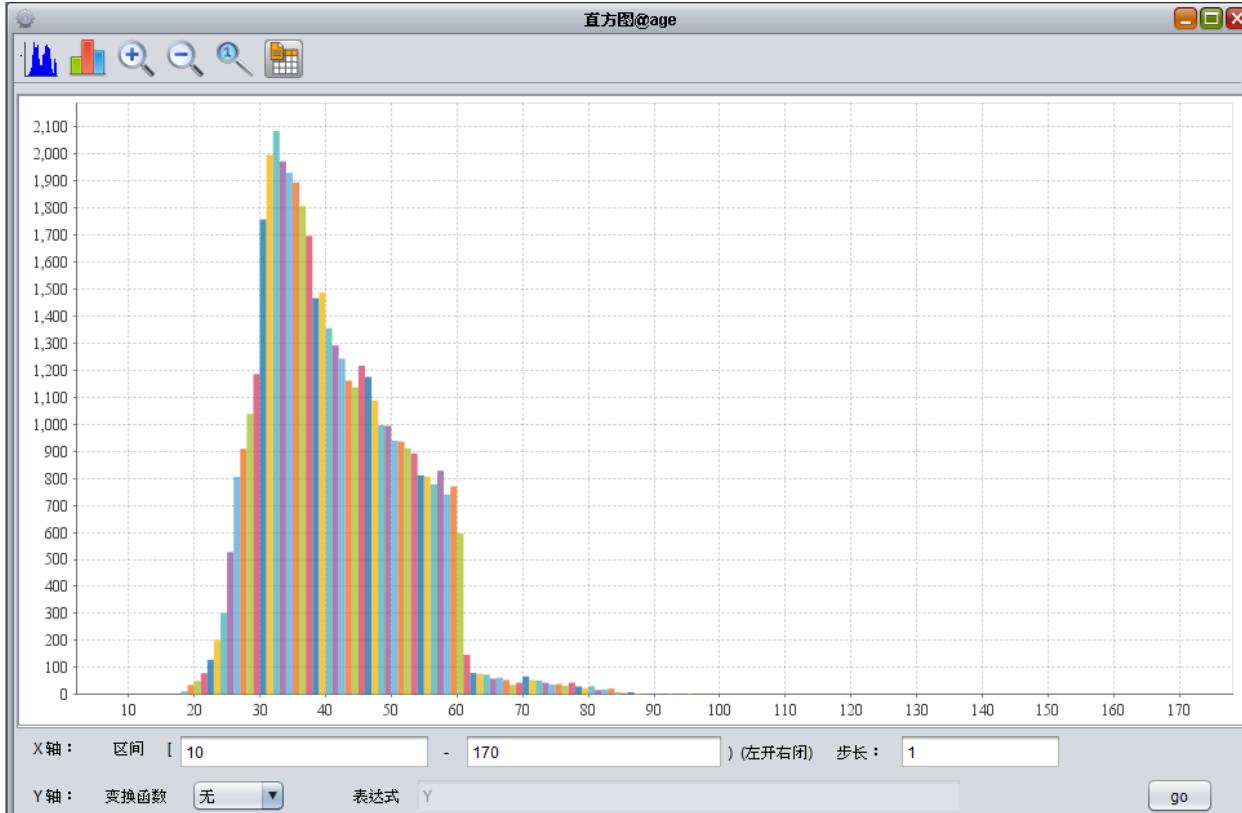
下方区域 显示直方图的信息。区间显示该数据的最大最小的区间，用户自定义的显示区间必须在此范围之内。在下方区间栏和步长栏输入对应的区间值和步长后，点击”GO”按钮显示新的直方图。(当用户输入步长值不合理时候，会自动调整后再显示)。

Y轴可以选择变换函数，默认是原值，可以选择按对数显示，或者自定义表达式显示。其中对数的公式是：

```
case when y>=0 then log(2, y+1)
else -log(2, -y+1)
end.
```

注意事项：表达式的变量必须是Y。

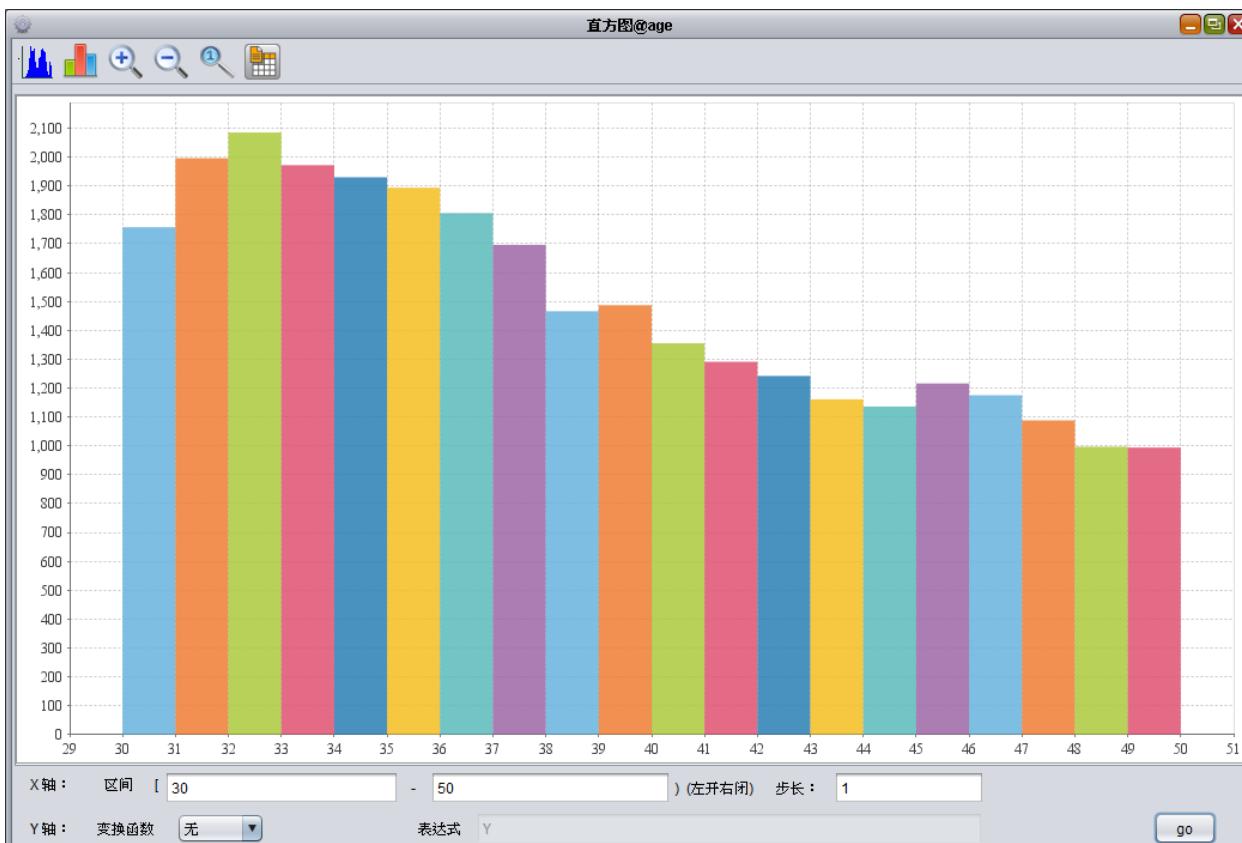
例如选择推荐最小步长，结果如下图：



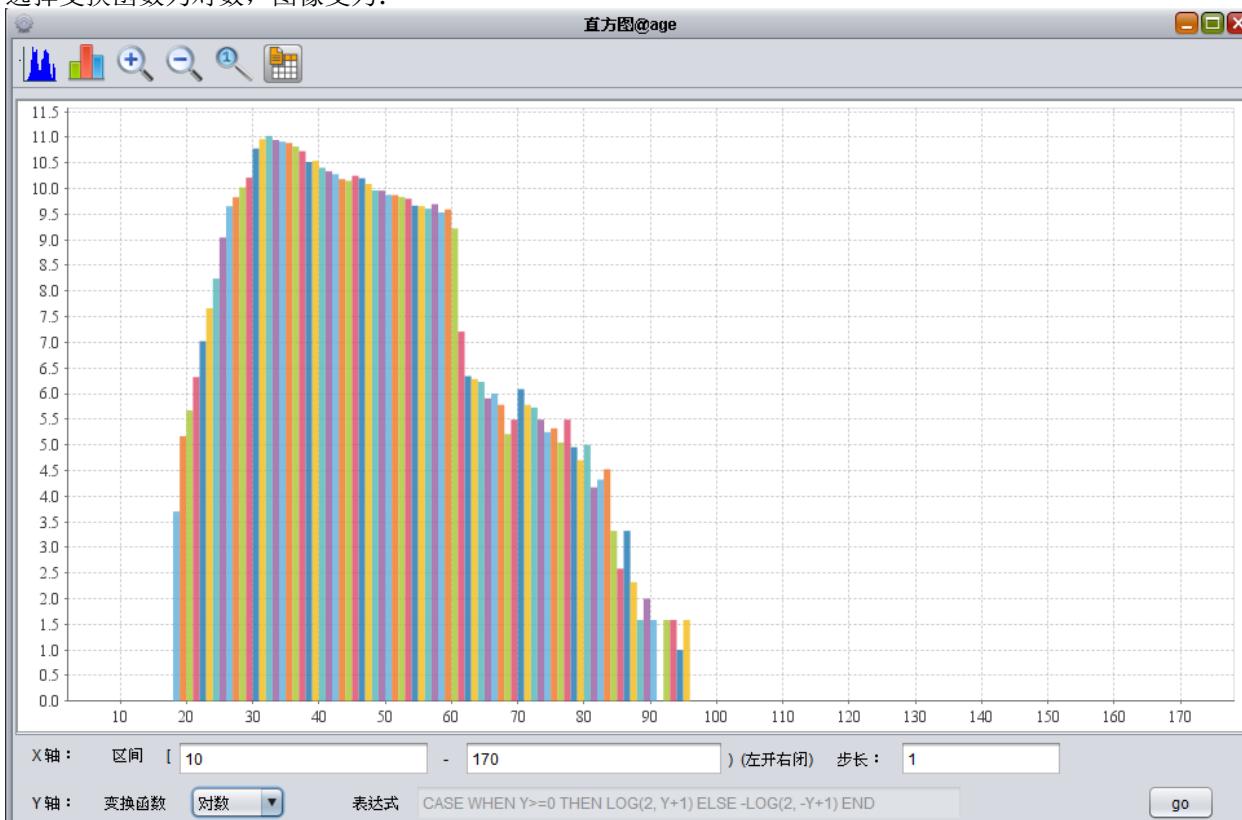
选择查看数据，显示每个区间的频数，结果如下图：

直方图数据导出表	
基本统计 画图	
HistoData_1	Data
start	10
end	108
size	98
[10, 11)	0.0
[11, 12)	0.0
[12, 13)	0.0
[13, 14)	0.0
[14, 15)	0.0
[15, 16)	0.0
[16, 17)	0.0
[17, 18)	0.0
[18, 19)	12.0
[19, 20)	35.0
[20, 21)	50.0
[21, 22)	79.0
[22, 23)	129.0
[23, 24)	202.0
[24, 25)	302.0
[25, 26)	527.0
[26, 27)	805.0
[27, 28)	909.0
[28, 29)	1038.0
[29, 30)	1185.0
[30, 31)	1757.0
[31, 32)	1996.0
[32, 33)	2085.0
[33, 34)	1972.0
[34, 35)	1930.0
[35, 36)	1894.0
[36, 37)	1806.0
[37, 38)	1696.0
[38, 39)	1466.0
[39, 40)	1487.0
[40, 41)	1355.0

选择x轴区间[30, 50), 步长是1, 变换函数无, 点击”GO”, 显示如下图:



选择变换函数为对数，图像变为：



4.3.2 函数

示例：

```
srt = Statistics.summary('kddcup', summaryColNames=['countg'])
show(srt.col('countg').getHistogram())
```

4.4 协方差

该操作对于全表数据计算协方差，协方差(Covariance)在概率论和统计学中用于衡量两个变量的总体误差。执行此统计的方式有两种：界面和函数，详细介绍见下文。

4.4.1 界面

该操作对于全表数据计算协方差。点击菜单→全表描述→协方差，显示参数窗体如下：



或者在表格上选取一列，右键弹出菜单选择协方差，参考单列基本统计量功能。点击菜单后点击确定进行计算，运行结果如下：

bank_marketing:协方差								
Column Names	age	balance	day	duration	campaign	pdays	previous	
age	112.75810...	3161.4767...	-0.805978...	-12.71171...	0.1566006...	-25.26054...	0.0315118...	
balance	3161.4767...	9270598.9...	114.09546...	16905.750...	-137.5132...	1047.3217...	116.93952...	
day	-0.805978...	114.09546...	69.263609...	-64.74031...	4.1895185...	-77.53540...	-0.991307...	
duration	-12.71171...	16905.750...	-64.74031...	66320.574...	-67.47179...	-40.34907...	0.7136535...	
campaign	0.1566006...	-137.5132...	4.1895185...	-67.47179...	9.5977333...	-27.49238...	-0.234458...	
pdays	-25.26054...	1047.3217...	-77.53540...	-40.34907...	-27.49238...	10025.765...	104.89990...	
previous	0.0315118...	116.93952...	-0.991307...	0.7136535...	-0.234458...	104.89990...	5.3058406...	

4.4.2 函数

示例：整表协方差

```
srt = Statistics.summary('kddcup')
#返回cov二维数组
cov = srt.getCov()
print cov[4][4]
```

示例：选列协方差

```
srt = Statistics.summary('kddcup', summaryColNames=['flagg', 'countg'])
#返回2*2的二维数组
cov = srt.getCov()
print cov[1][1]
```

4.5 相关矩阵

该操作对于全表数据计算相关矩阵，相关矩阵也叫相关系数矩阵，是由矩阵各列间的关系系数构成的。也就是说，相关矩阵第*i*行第*j*列的元素是原矩阵第*i*行和第*j*列的相关系数。这里的相关系数是指皮尔逊相关系数。

执行此统计的方式有两种：界面和函数，详细介绍见下文。

4.5.1 界面

该操作对于全表数据计算相关矩阵，点击菜单→全表描述→协相关矩阵，显示参数窗体如下：



或者在表格上选取一列，右键弹出菜单选择相关矩阵，参考单列基本统计量功能。点击菜单后点击确定进行计算，运行结果如下：

Column Names	age	balance	day	duration	campaign	pdays	previous
age	1	0.0977827...	-0.009120...	-0.004648...	0.0047603...	-0.023758...	0.0012883...
balance	0.0977827...	1	0.0045025...	0.0215603...	-0.014578...	0.0034353...	0.0166736...
day	-0.009120...	0.0045025...	1	-0.030206...	0.1624902...	-0.093044...	-0.051710...
duration	-0.004648...	0.0215603...	-0.030206...	1	-0.084569...	-0.001564...	0.0012030...
campaign	0.0047603...	-0.014578...	0.1624902...	-0.084569...	1	-0.088627...	-0.032855...
pdays	-0.023758...	0.0034353...	-0.093044...	-0.001564...	-0.088627...	1	0.4548196...
previous	0.0012883...	0.0166736...	-0.051710...	0.0012030...	-0.032855...	0.4548196...	1

4.5.2 函数

示例：整表相关矩阵

```
srt = Statistics.summary('kddcup')
#返回corr二维数组
corr = srt.getCorr()
print corr[1][1]
```

示例：选列相关矩阵

```
srt = Statistics.summary('kddcup', summaryColNames=['flagg', 'countg'])
#返回corr 2*2二维数组
corr = srt.getCorr()
print corr[1][1]
```

4.6 Top100

Top100此操作对于单列数据计算最大的前100值，执行此统计的方式有两种：界面和函数，详细介绍见下文。

4.6.1 界面

点击菜单→全表描述→Top100，显示参数窗体如下图：



或者在表格上选取一列，右键弹出菜单选择Top 100，参考单列基本统计量功能。点击菜单后点击确定进行计算，运行结果如下图：



4.6.2 函数

示例：

```
srt = Statistics.summary('kddcup')
print srt.col('countg').topItems
```

4.7 bottom100

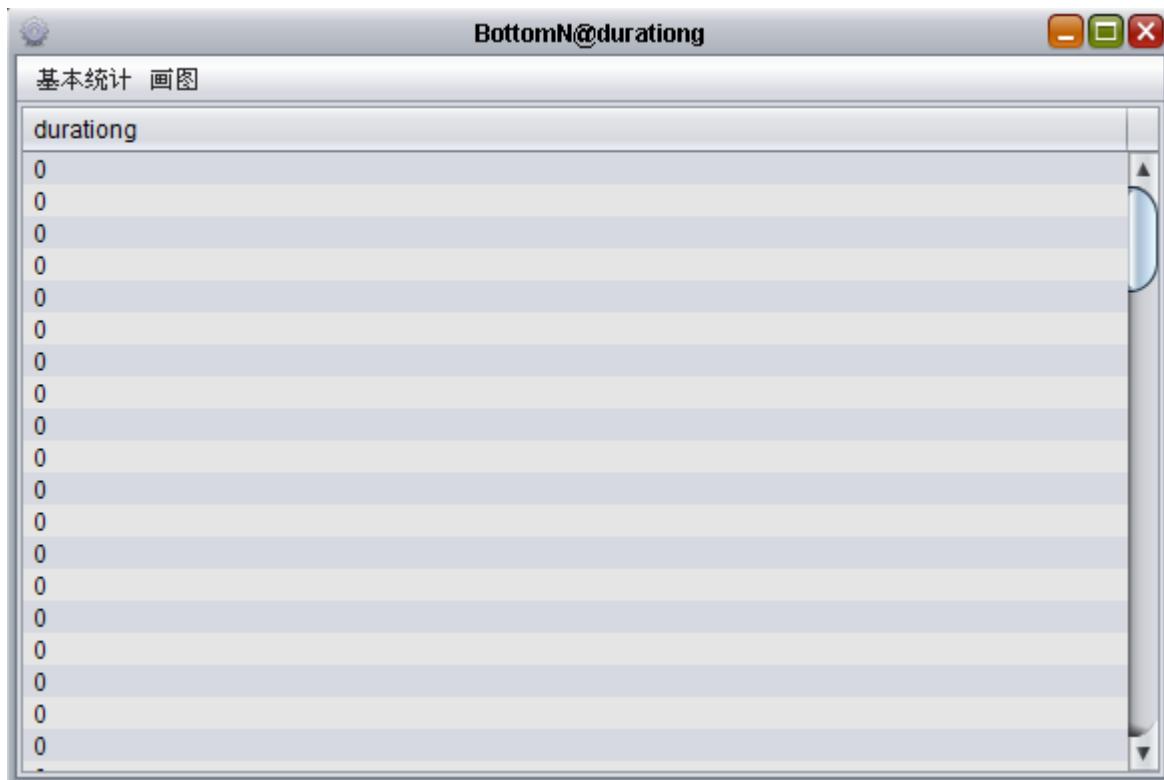
bottom100此操作对于单列数据计算最小的前100值，执行此统计的方式有两种：界面和函数，详细介绍见下文。

4.7.1 界面

点击菜单→基本统计→Bottom 100，显示参数窗体如下图：



或者在表格上选取一列，右键弹出菜单选择Bottom 100，功能参考单列基本统计量。点击菜单后点击确定进行计算，运行结果如下图：



4.7.2 函数

示例：

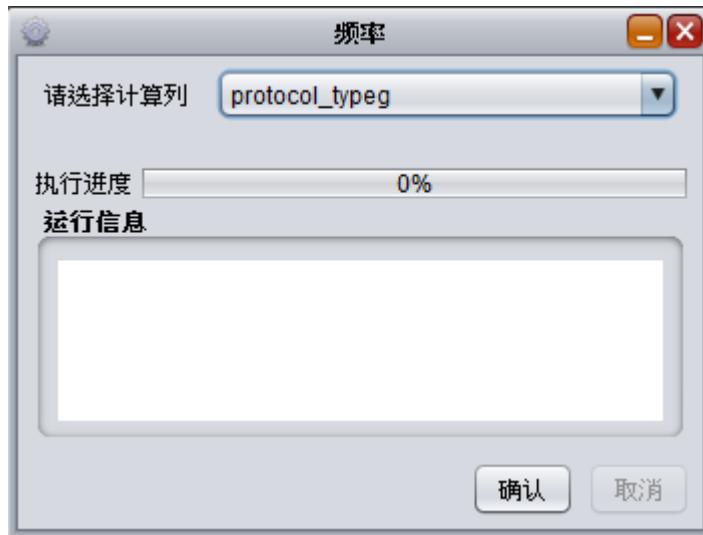
```
srt = Statistics.summary('kddcup')
print srt.col('countg').bottomItems
```

4.8 频率

频率此该操作对于单列数据计算频率，执行此统计的方式有两种：界面和函数，详细介绍见下文。

4.8.1 界面

点击菜单→基全表描述→频率，显示参数窗体如下图：



或者在表格上选取一列，右键弹出菜单选择频率，参考单列基本统计量。点击菜单后点击确定进行计算，运行结果如下图：

频率@protocol_typeg	
基本统计 画图	
Item	count
icmp	2833545
tcp	1870598
udp	194288

注意事项：

- 最多显示10000行.
- 不包括空值

4.8.2 函数

```
def freq(inputTableName, colName, outTableName, srt = None):
```

参数：

- inputTableName 输入表
- colName 列名
- outTableName 输出表名
- srt (可选) summary函数的结果.

示例：

```
Statistics.freq("bank_marketing", "job", "outTableName")
```

注意事项：

- 如果已经计算过summary的结果srt，并且这一列distinct值小于10000，可以直接从srt中取，参考全表统计汇总
- freq = srt.col('colname').getFrequencyOrderByCount() 或者 freq = srt.col('colname').getFrequencyOrderByItem()
- freq是一个list

示例2：

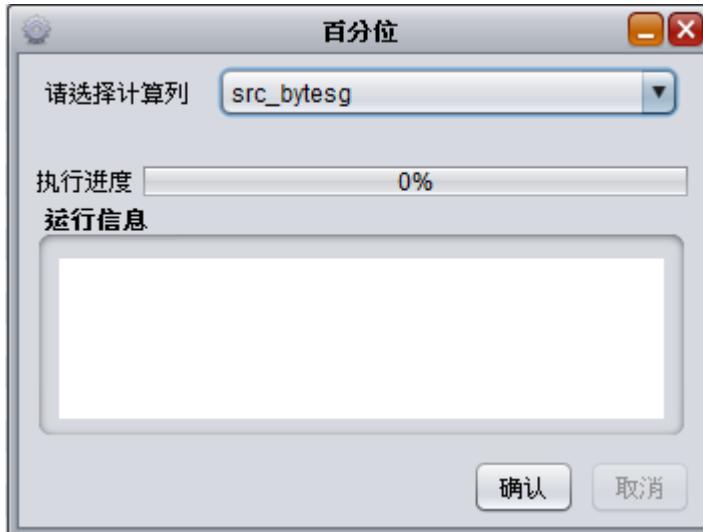
```
srt = Statistics.summary("bank_marketing")
src = srt.col("age")
freq = src.getFrequencyOrderByCount()
m = freq.size()
for k in range(0, m):
    print freq.get(k).getKey()
    print freq.get(k).getValue()
```

4.9 百分位

该操作对于单列数据计算百分位，执行此统计的方式有两种：界面和函数，详细介绍见下文。

4.9.1 界面

点击菜单：全表描述 → 百分位，显示参数窗体如下图：



或者在表格上选取一列，右键弹出菜单选择百分位，参考单列基本统计量功能。点击菜单后点击确定进行计算，运行结果如下图：

百分位(精确结果)@src_bytes	
key	value
Min	0
Lower Quartile (Q1)	45
Median	520
Upper Quartile (Q3)	1032
Max	1379963888
0%	0
1%	0
2%	0
3%	0
4%	0
5%	0
6%	0
7%	0
8%	0
9%	0
10%	0
11%	0
12%	0
13%	0
14%	0
15%	0
16%	0
17%	0
18%	0
19%	0
20%	0
21%	0
22%	0
23%	0
24%	18
25%	45
26%	105
27%	105
28%	147

4.9.2 函数

```
def percentile(inputTableName, colName, inputPartitions=None, outputTableName=None):
```

参数:

- inputTableName 输入表
- colName 列名
- outputTableName (可选)输出表名
- inputPartitions (可选)输出表名, 默认不输出到表

返回:

- Percentile

示例:

```
pct = Statistics.percentile("bank_marketing", "age")
print pct
print pct.getPercentile(33)
```

注意事项:

- 如果已经计算过summary的结果srt，并且这一列distinct值小于10000，可以直接从srt中取，参考全表统计汇总
- percentile = srt.col('colname').getPercentile()

示例2:

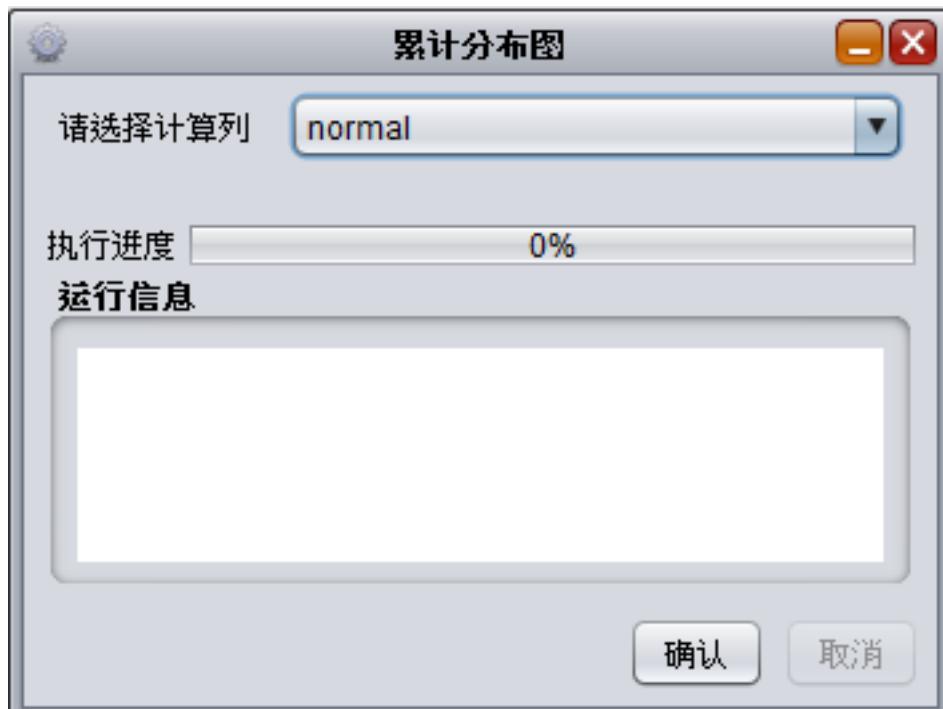
```
srt = Statistics.summary("bank_marketing")
src = srt.col("age")
pct = src.getPercentile()
```

4.10 累积分布图

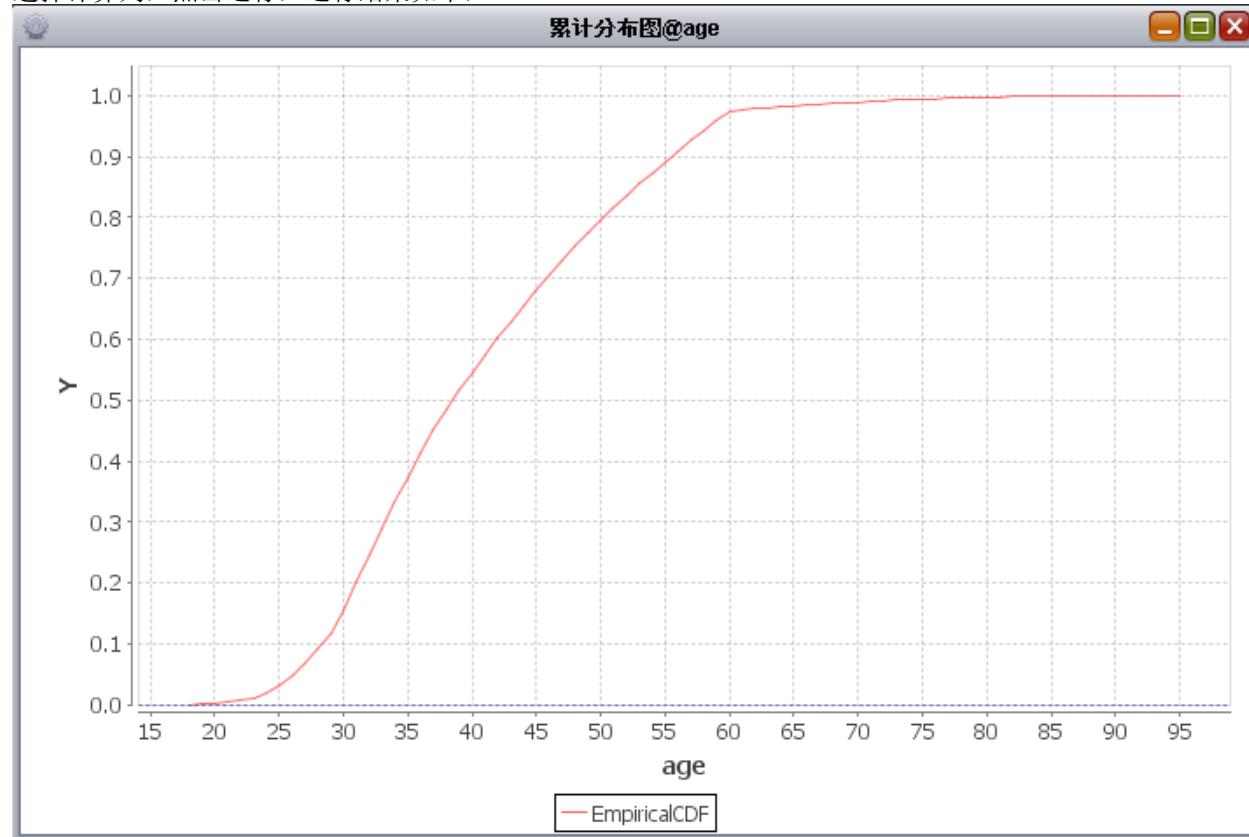
累积分布图是个随机变量的概率分布，是概率密度函数的积分。累积分布是等于或者等于某个数值的概率 $P(X \leq x)$ ，即： $F(x) = P(X \leq x)$.

4.10.1 界面

选择界面→全表描述→累积分布图，如下图：



选择计算列, 点击运行, 运行结果如下,



4.10.2 函数

```
Def empiricalCDF(srt, colname):
```

参数

- srt: summary函数的结果
- colname: 列名

示例:

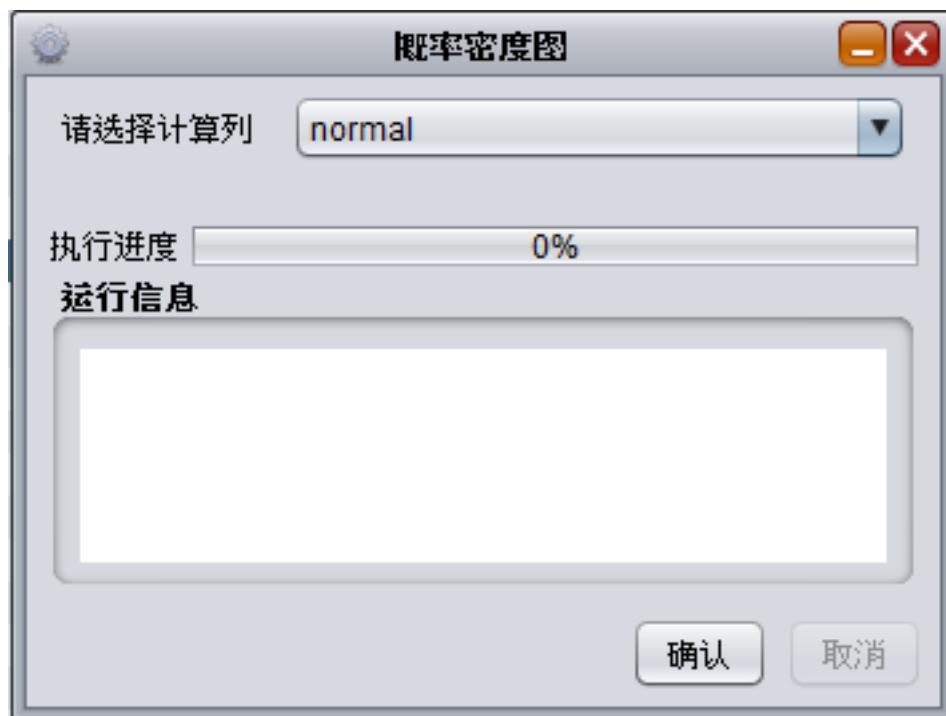
```
srt = Statistics.summary('bank_marketing')
cdf = Statistics.empiricalCDF(srt, 'age')
show(cdf)
show(cdf, 'bank_marketing@age')
```

4.11 概率密度图

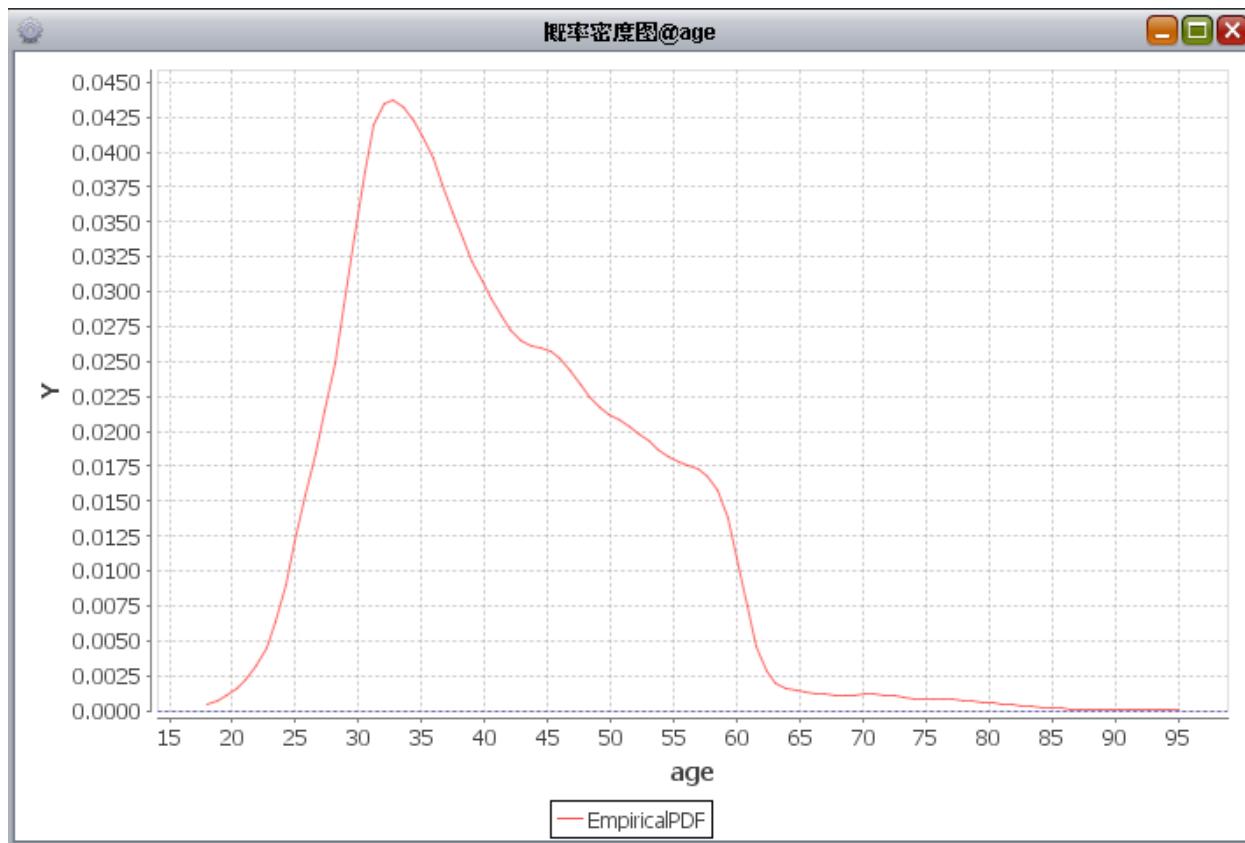
随机变量的概率密度函数, 表示瞬时幅值落在某指定范围内的概率, 因此是幅值的函数。

4.11.1 界面

选择界面->全表描述->概率密度图, 如下图:



选择计算列, 点击运行, 运行结果如下,



4.11.2 函数

```
Def empiricalPDF(srt, colname):
```

参数

- srt: summary函数的结果
- colname: 列名

示例:

```
srt = Statistics.summary('bank_marketing')
pdf = Statistics.empiricalPDF(srt, 'age')
show(pdf)
show(pdf, 'bank_marketing@age')
```

4.12 全表统计汇总

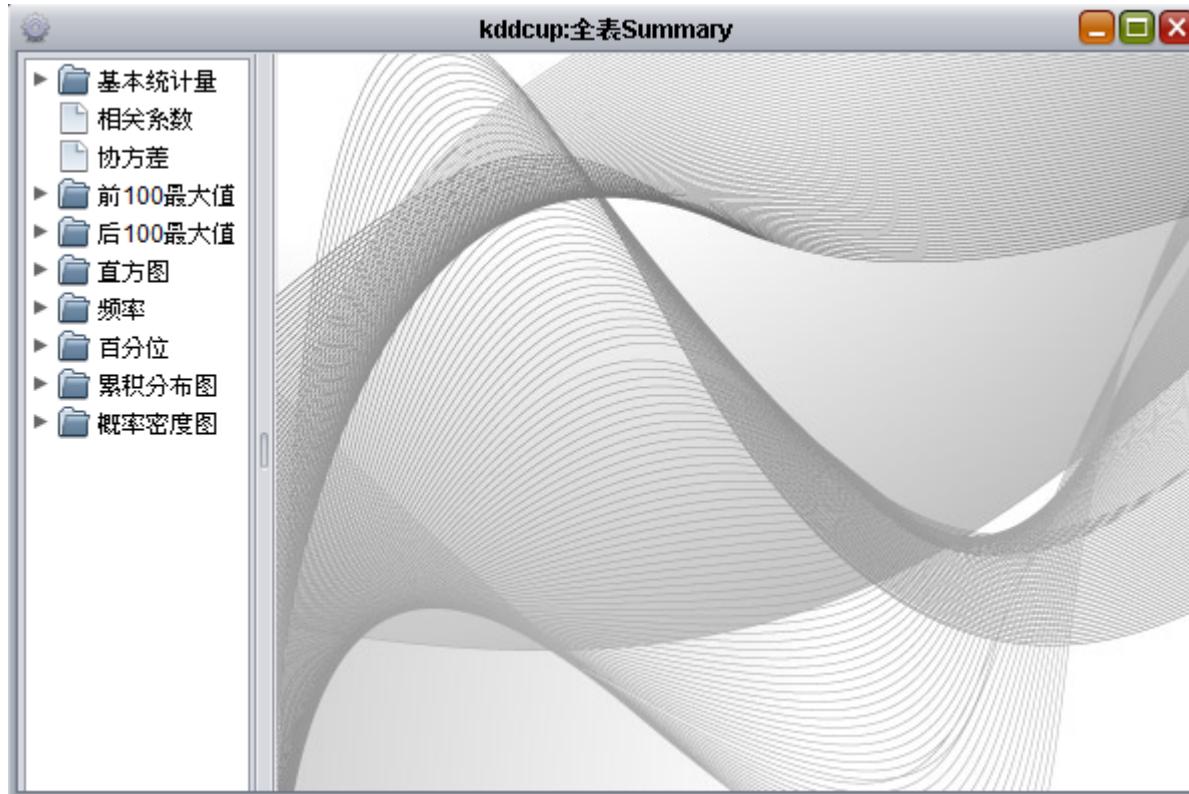
该操作对于全表数据进行基统计汇总, 包括基本统计量, 频率, 最大100个值, 最小100个值, 协方差, 相关矩阵, 频率, 直方图的计算。

4.12.1 界面

点击菜单→全表描述→全表统计汇总，显示参数窗体如下图：



计算过程中可以点击取消按钮取消计算同时终止云端JOB，运行结果如下图：



点击左侧的树状结构，可以查看相应结果。

4.12.2 函数

```
summary(inputTableName):
```

返回：

- 返回SummaryResultTable
- 给定SummaryResultTable srt, 通过列名可以取出每一列的统计量等
- src = srt.col(“colName”)
- 包括基本统计量, 统计量参考全表基本统计
- 可以取出最大的100个值

```
src.topItems
```

- 可以画出直方图

```
show(src.getHistogram())
```

- 如果这一列的不同元素个数小于10000, 可以取出每一列的频率和百分位

```
src.getFrequencyOrderByCount()
```

```
src.getPercentile()
```

- 给定SummaryResultTable srt, 可以取出协方差和相关矩阵

```
srt.getCov()
```

```
srt.getCorr()
```

示例1：

```
srt = Statistics.summary('kddcup')
print srt
print srt.col('flag').count
```

4.13 分位数

分位数提供了比百分位更为灵活的计算方式, 该操作对于单列数据计算分位数。执行此统计的方式有两种：界面和函数，详细介绍见下文。

4.13.1 界面

点击菜单：统计→分位数显示参数窗体如下图：



点击菜单后点击确定进行计算，运行结果如下图：

百分位结果@adult

	colname	quantile	value
0	age	0	17
1	age	1	17
2	age	2	18
3	age	3	19
4	age	4	19
5	age	5	19
6	age	6	20
7	age	7	20
8	age	8	21
9	age	9	21
10	age	10	22
11	age	11	22
12	age	12	23
13	age	13	23
14	age	14	23
15	age	15	24
16	age	16	24
17	age	17	24
18	age	18	25
19	age	19	25
20	age	20	26
21	age	21	26
22	age	22	26
23	age	23	27
24	age	24	27
25	age	25	28
26	age	26	28
27	age	27	28
28	age	28	29
29	age	29	29

Whole Table ▾ 位置 数据大小: -- 行, 当前显示前 50 行 (最多10000行) GO!

其中colname标识计算列名, quantile标识分位点位置, 从0开始到最大分位点数, value列标识相应分位点的值

4.13.2 函数

```
def quantile(inputTableName, selectedColName, quantileTableName=None, inputPartitions=None,
expression=None, quantileNum=100, roundMode='floor')
```

参数:

- inputTableName: 输入表名
- selectedColName: 计算分位数的列名
- quantileTableName: (可选)输出分位表名
- inputPartitions: (可选)输入表选择分区, 默认全选
- expression: (可选)过滤表达式, 默认不过滤
- quantileNum: (可选)最大分位点个数, 默认为100

- roundMode: (可选) 指定计算分位点时的算法, "floor" 为上取整、"ceil" 为下取整、"round" 为四舍五入, 默认为floor

示例:

```
Statistics.quantile("input_table", "col0")
Statistics.quantile("input_table", "col0", quantileTableName="quantile_table", quantileNum=1000, roundMode="ceil")
```

4.14 排名和分位

sort_rank指令用于多列数据同时进行排序和分位, 计算时还有分组功能(类似SQL的Group by), 并能追加ID字段。

输入格式示例:

ID(string)	data_col1(double)	data_col2(double)	data_col3(double)	Group1(string)
卖家ID例 如”张三”	特征值列1, 例 如”销售额”	特征值列2, 例如”页 面点击率”	特征值列3, 例 如”买家打分”	分类, 例 如”女装类”

输出排序表<sort_table>示例, 其实就是输入表中3个数据字段改为排名:

ID(string)	Group1(string)	data_col1(double)	data_col2(double)	data_col3(double)
“张三”	“女装类”	销售额在”女装 类”中的排名	页面点击率在”女装 类”中的排名	买家打分在”女装 类”中的排名

输出分位表<rank_table>示例, 各个分组中各个数据字段的分位值(默认为百分位)

Group1(string)	col_name(string)	Rank(bigint)	Value(double)
“女装类”	“销售额”	0	0%处分位点
“女装类”	“销售额”	1	1%处分位点
			(其他分位点)
“女装类”	“销售额”	100	100%处分位点

4.14.1 函数

help(Statistics.sortRank) 可以查看帮助信息:

```
def sortRank(inputTableName, rankTableName, quantileTableName,
    inputPartitions=None, rankColNames=None, idColNames=None, groupColNames=None,
    quantileNum=100, roundMode='floor', ties='order')
```

参数:

- inputTableName: 输入表名
- rankTableName: 输出排序表名, 其格式相当于原始表里面被排序的列里的具体数据都被替换为排名

- quantileTableName: 输出分位表名, 其格式为: ID字段, group by字段, 以及每个Rank的分位点的数值
- inputPartitions: (可选)输入表选择分区, 默认全选
- rankColNames: (可选)在输入表里指定被排序的数据字段的序号, 默认选择全部列
- idColNames: (可选)在输入表里指定ID字段的序号, 默认为空, 最大支持5个id列
- groupColNames: (可选)在输入表里指定分组字段的序号, 默认为空, 最大支持5个group列
- quantileNum: (可选)最大分位点个数, 默认为100
- roundMode: (可选)指定计算分位点时的算法, "floor" 为上取整、"ceil" 为下取整、"round" 为四舍五入, 默认为floor
- ties: (可选)指定并列时计算排名的方式, 默认为order, 排名从1开始, 例如对于序列(1, 2, 2, 5):
 - high为取上界(排名为 1, 2, 2, 4)
 - low为取下界(排名为 1, 3, 3, 4)
 - mean为取均值(排名为 1, 2.5, 2.5, 4)
 - dense排名值无间隔(排名为1, 2, 2, 3)
 - order为不并列, 随机依次排下来(排名为1, 2, 3, 4)

示例:

```
Statistics.sortRank("input_table", "rank_table", "quantile_table", rankColNames=['col0', 'col1'], groupColNames=[ 'col3' ])

# 读取表input_table进行sort_rank计算;
# 以列col3' 作为group列, 对于'col0', 'col1' 列进行计算;
# 使用默认的分位点计算方法和ties参数, 默认分位数个数为100;
# 结果存储到rank_table和rank_table中
```

4.14.2 界面

点击 统计>排名和分位, 显示参数界面:



界面显示项介绍:

- Partition筛选对应脚本参数inputPartitions
- Id, Group和Sort列分别对应脚本参数idColNames, groupColNames, rankColNames
- Ties选择框对应脚本参数ties
- RoundMode选择框对应脚本参数roundMode
- 分位数个数对应脚本参数quantileNum

- 输出rank表名即为排名表, 对应脚本参数rankTableName
- 输出分位数表名对应脚本参数quantileTableName

4.15 按条件统计对比

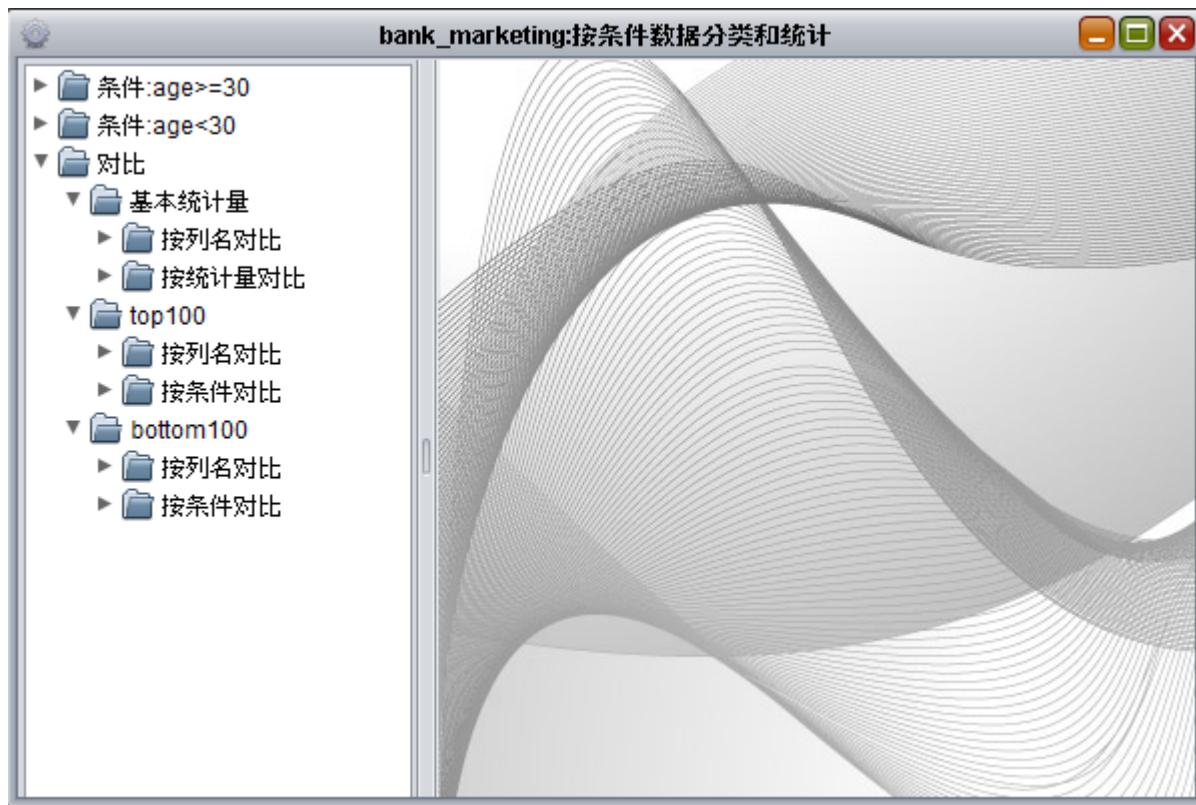
按条件统计对比, 又称作连续变量分组统计。如果想看一个变量多组过滤条件之间统计结果的对比, 可以使用按条件统计对比。

4.15.1 界面

菜单->统计->统计对比->按条件, 输入条件列表, 输入条件列表对应的输出表列表。如果有输出表列表, 那么结果中包含过滤表, 否则只有统计结果。



以UCI上Bank Marketing数据集为例，将age字段分成小于等于30岁和大于30岁两组，考虑观察每组的balance字段的统计量以及balance字段统计量的对比。运行结果，如下图：

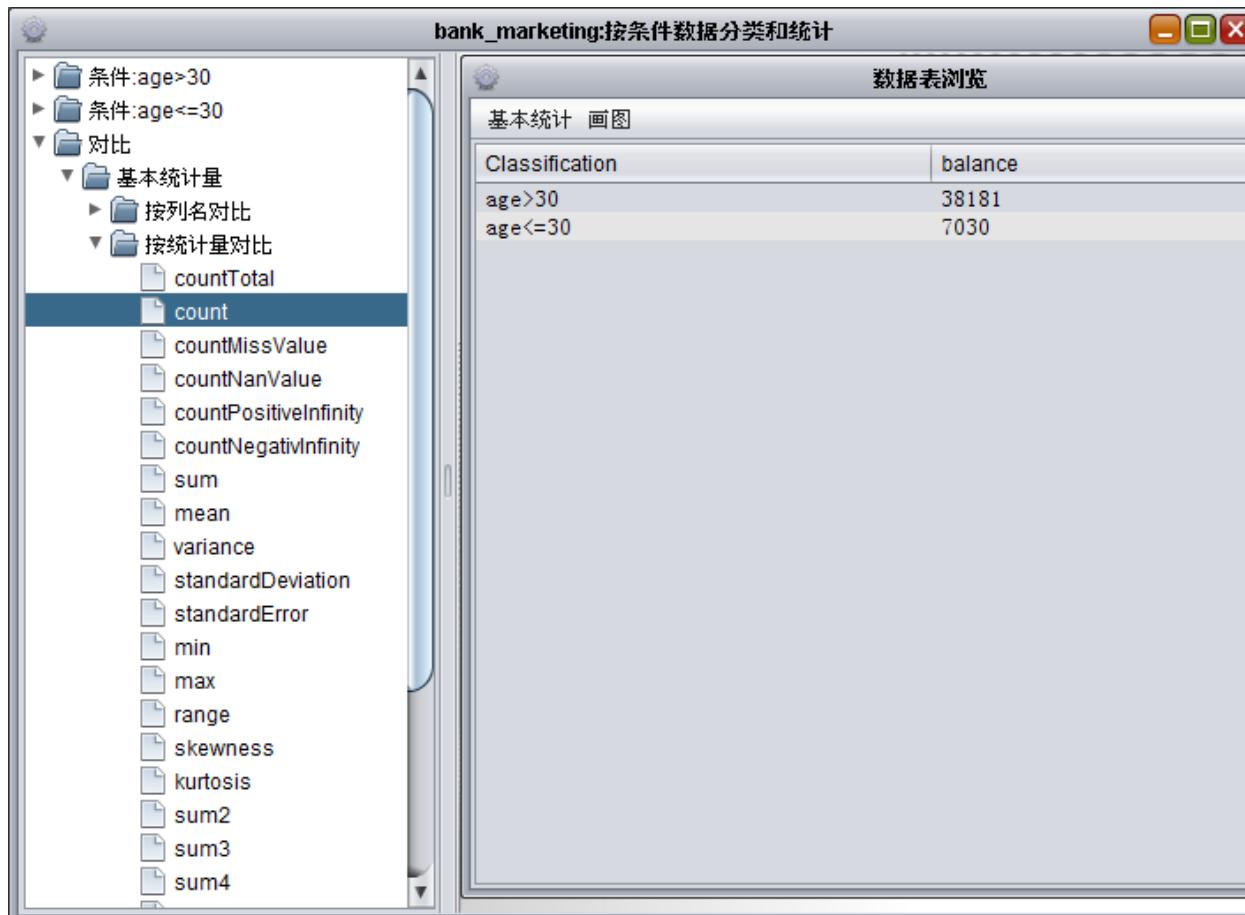


这是个树状结构，左侧是条件列表，可以看到每个条件的统计量，以及所有条件直接的对比。对比分为基本统计量的对比，top100的对比，bottom100的对比。如果选定基本统计量按列对比的结果，行显示的是两个分类条件，列显示的是基本统计量。

bank_marketing:按条件数据分类和统计

Statistics	age>=30	age<30
countTotal	39938.0	5273.0
count	39938.0	5273.0
countMissValue	0.0	0.0
countNaNValue	0.0	0.0
countPositiveInfinity	0.0	0.0
countNegativInfinity	0.0	0.0
sum	5.6387497E7	5202185.0
sum2	4.73395908041E11	2.9629773521E10
sum3	1.21211856940647E16	3.74393098375241E14
sum4	6.142892502972903E20	7.1437043238428324E18
min	-6847.0	-8019.0
max	102127.0	36252.0
range	108974.0	44271.0
mean	1411.8758325404376	986.5702636070548
variance	9860123.789199328	4646709.501938038
standardDeviation	3140.0834048157585	2155.6227642929634
cv	2.224050679559899	2.1849662855351735
standardError	15.712599010662476	29.685466326884164
skewness	8.363020971980749	5.621432413720926
kurtosis	138.9183714755229	48.17515519767856
moment2	1.1853270269943412E7	5619149.160060686
moment3	3.0350006745617456E11	7.100191511004001E10
moment4	1.5381071918906562E16	1.3547704008804918E15
centralMoment2	9859876.903431658	4645828.275026993
centralMoment3	2.5892287780191376E11	5.629135759001861E10
centralMoment4	1.3796902807867998E16	1.1045502392195314E15

如果选定基本统计量->按统计量对比->count, 那么结果显示的是所有分类条件和所有列的count值的对比。



注意事项:

- 如果想保留每个分组的数据, 那么在输出表名中输入和表达式个数相同的表名, 相应分组数据会存在相应的表中。

4.15.2 函数

```
def summary(inputTableName, inputPartitions = None, summaryColNames = None, filter = None, by = None):
```

参数:

- `inputTableName`: 输入表名.
- `inputPartitions`: (可选) 表的分区列表.
- `summaryColNames`: (可选) 需要计算的列名列表.
- `filter`: (可选) 过滤条件.
- `by`: (可选) 分组条件列表.

返回:

- 如果`by=None`, 或者`by`只有一个条件, 返回`SummaryResultTable`.

- 否则返回SummaryResultTable 列表，其中每一个代表在filter条件下，by分组条件下对应的summary结果。

示例：

```
srts = Statistics.summary("bank_marketing", by=["age>30", "age<=30"])
#打印"age>30"的统计汇总
print srts[0]
```

4.16 按分区统计对比

按分区统计对比，又称作按partition分类统计。如果想看各个分区组之间统计的对比，可以选择按partition分类统计。

4.16.1 界面

菜单->统计->统计对比->按分区，如下图：

点击”增加Partition分组”，如下图：

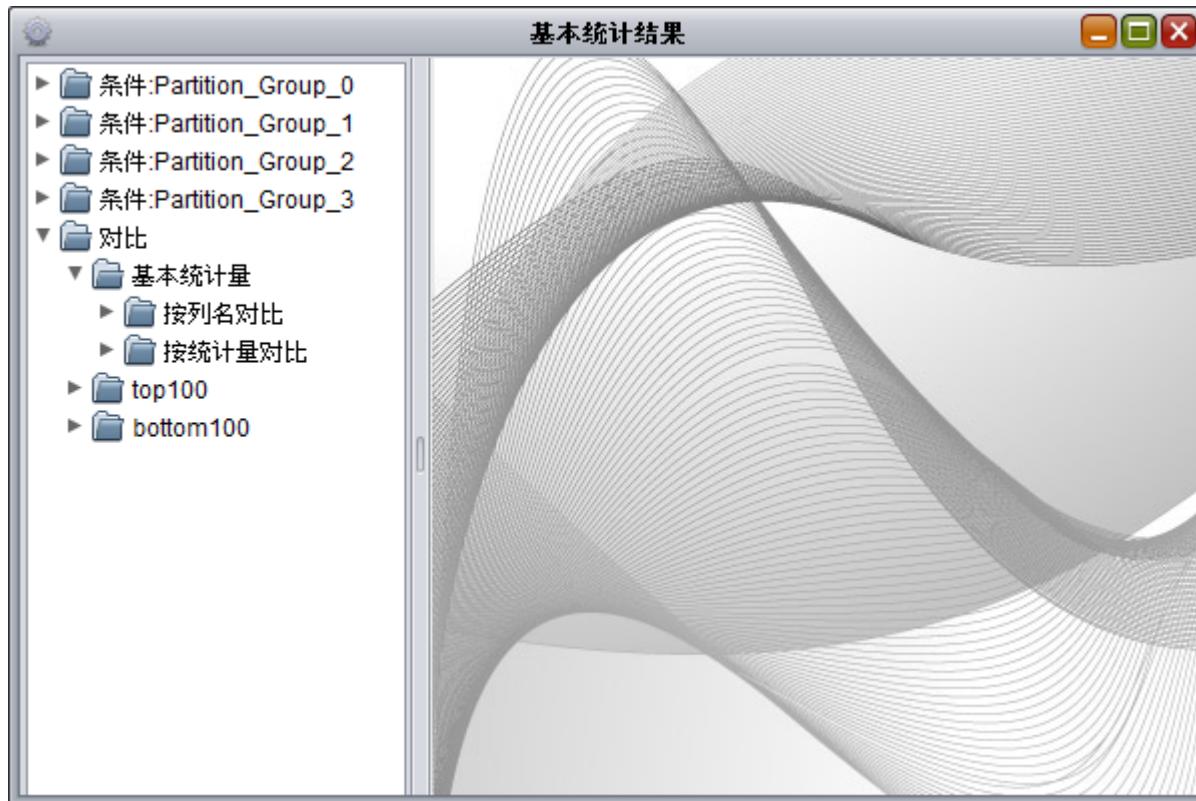
在0号分组上点编辑，如下图：

可以勾选想要的partition，或者输入表达式过滤出想要的partition，最终以勾选的为准。例如，如果想选择201306的数据，那么在过滤条件中输入pdate rlike 201306，点击选择，结果如下图：

点击确定，如下图：

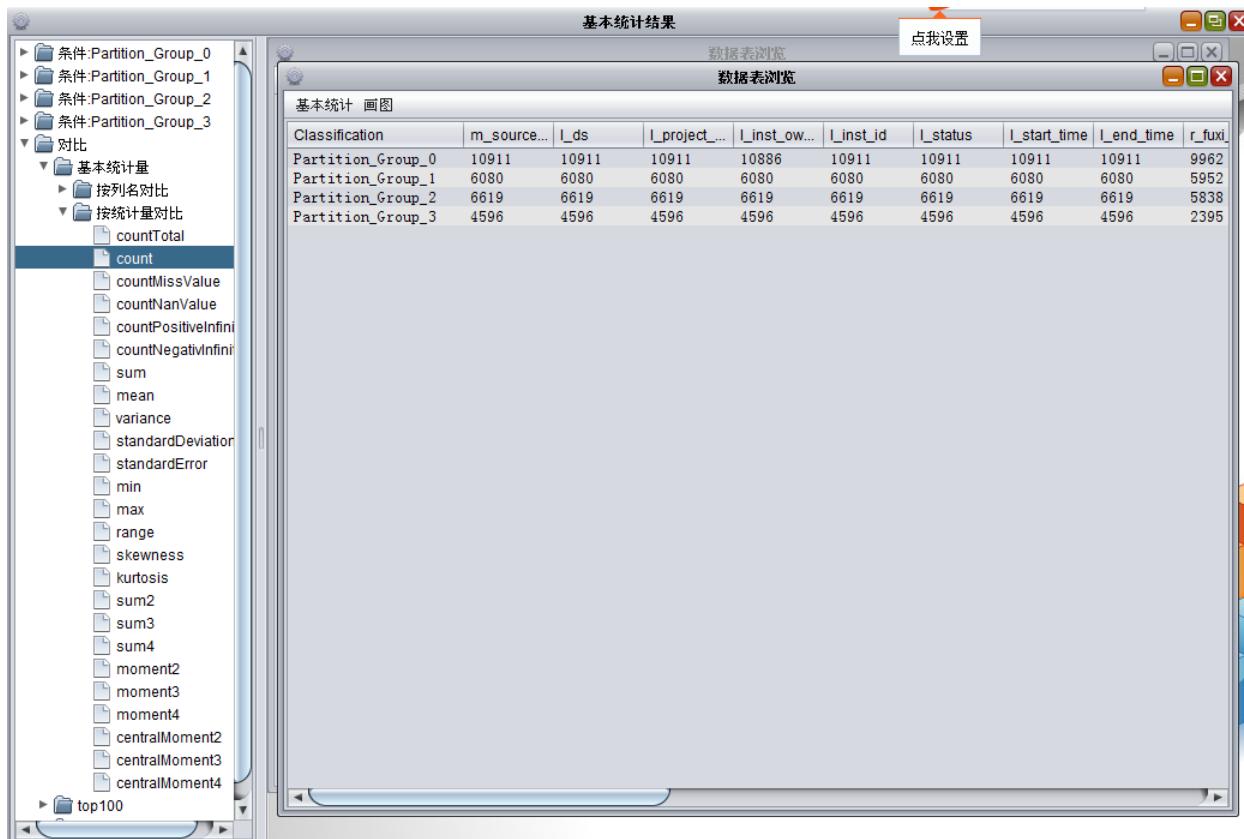
这样就选定了第0号partition组，代表2013年六月的数据。以同样的方式选择第1号partition组代表2013年七月的数据，第2号partition组代表2013年八月的数据，第3号partition组代表2013年九月的数据。

点击确定，开始运行，运行结束后，结果如下图：



Partition_Group_0代表分组号为0的partition组， Partition_Group_1代表分组号为1的partition组， 结果解释参照连续变量分组功能。

如果选择对比->按统计量对比，并选择统计量是count，结果如下图：



可以看出6月次数是10911，7月次数是6080，8月次数是6691，9月次数是4596。

4.16.2 函数

```
def summaryByPartitionGroup(inputTableName, partitionGroups, summaryColNames = None, filter = None):
```

参数:

- inputTableName: 输入表名.
- partitionGroup: 表的分区列表组. (必须是全路径, 不支持表达式)
- summaryColNames: (可选)需要计算的列名列表.
- filter: (可选)过滤条件.

返回:

- srt列表. 其中每一个代表在filter条件下, 每个partitionGroup的srt.

示例:

```
partitionGroup1 = ['ds=20130603']
partitionGroup2 = ['ds=20130606']
[srt1,srt2]= Statistics.summaryByPartitionGroup('tableName', [partitionGroup1, partitionGroup2], ['f1','f2'])
```

```
print srt1.col('f2').count  
print srt2.col('f1').count
```

4.17 分组聚合函数

分组聚合函数，又称作分类变量统计。如果想做groupby，并对每组结果进行汇总统计，可以使用分类变量统计。

4.17.1 界面

菜单→统计→分组聚合函数。如下图：



选择合计函数和列明，并填上输出表名，点击确定，开始运行。

4.17.2 函数

```
def groupby(inputTableName, outTableName, groupedCols, aggregateCols = None, funcs = None, inputPartitions = None):
```

参数:

- inputTableName: 输入表名.
- outTableName: 输出表名.
- groupedCols: 分类变量列(group列).
- aggregateCols: 聚合列.
- funcs: 聚合函数(SUM, MIN, MAX, AVG, COUNT, VAR).
- inputPartitions: 输入表的分区列表.

示例:

```
Statistics.groupby("tableName", "outTableName", ["colName1"])
```

4.18 窗口聚合函数

窗口聚合函数即为 sql 中的窗口函数, 窗口函数的种类及用法详见: http://odps.alibaba-inc.com/doc/prddoc/odps_sql/odps_sql_func.html 窗口函数与聚合函数类似, 是对一定范围内的数据进行聚合操作, 并把聚合的结果返回, 与聚合函数不同的是, 聚合函数的结果会最多返回 group 字段的 distinct value 数量, 而窗口函数对每条数据都会有一个对应的返回值.

4.18.1 界面

点击统计→窗口聚合函数:



添加新增列名与对应的窗口函数表达式, 表达式格式为: `sum(columnA) over (partition by columnB, columnC order by columnD)` 选择留在结果表中的标识列, 填写输出表名称, 即可运行。 运行结果如下图所示:

窗口聚合函数运行结果表				
	age	race	sex	sum_age
0	39	White	Male	760297
1	50	White	Male	760297
2	38	White	Male	760297
3	53	Black	Male	59124
4	28	Black	Female	58863
5	37	White	Female	318126
6	49	Black	Female	58863
7	52	White	Male	760297
8	31	White	Female	318126
9	42	White	Male	760297
10	37	Black	Male	59124
11	30	Asian-Pac-Isl...	Male	27078
12	23	White	Female	318126
13	32	Black	Male	59124
14	40	Asian-Pac-Isl...	Male	27078
15	34	Amer-Indian-E...	Male	7144
16	25	White	Male	760297
17	32	White	Male	760297
18	38	White	Male	760297
19	43	White	Female	318126
20	40	White	Male	760297
21	54	Black	Female	58863
22	35	Black	Male	59124
23	43	White	Male	760297
24	59	White	Female	318126
25	56	White	Male	760297
26	19	White	Male	760297
27	54	Asian-Pac-Isl...	Male	27078
28	39	White	Male	760297
...

可以看到, 在 race 与 sex 相同的情况下, 新生成的 sum_age 列都是相同的。

4.18.2 函数

```
def windowfunc(inputTableName, outTableName, remainColumns, expressions,
               newColumnNames, inputpartitions=None):
```

参数:

- inputTableName: 输入表名.
- outTableName: 输出表名.
- remainColumns: 保留列.
- expressions: 窗口函数的表达式, 格式如: sum(a)over(partition by b, c order by d, e)
- newColumnNames: 根据 expressions 生成的列名
- inputpartitions: (可选)输入表的分区列表

实例:

```
Statistics.windowfunc("adult", "adulttest", ["age", "race", "sex"],
    ["avg(age) over (partition by race, sex order by age)"], ["avg_age"])
```

4.19 扩展直方图

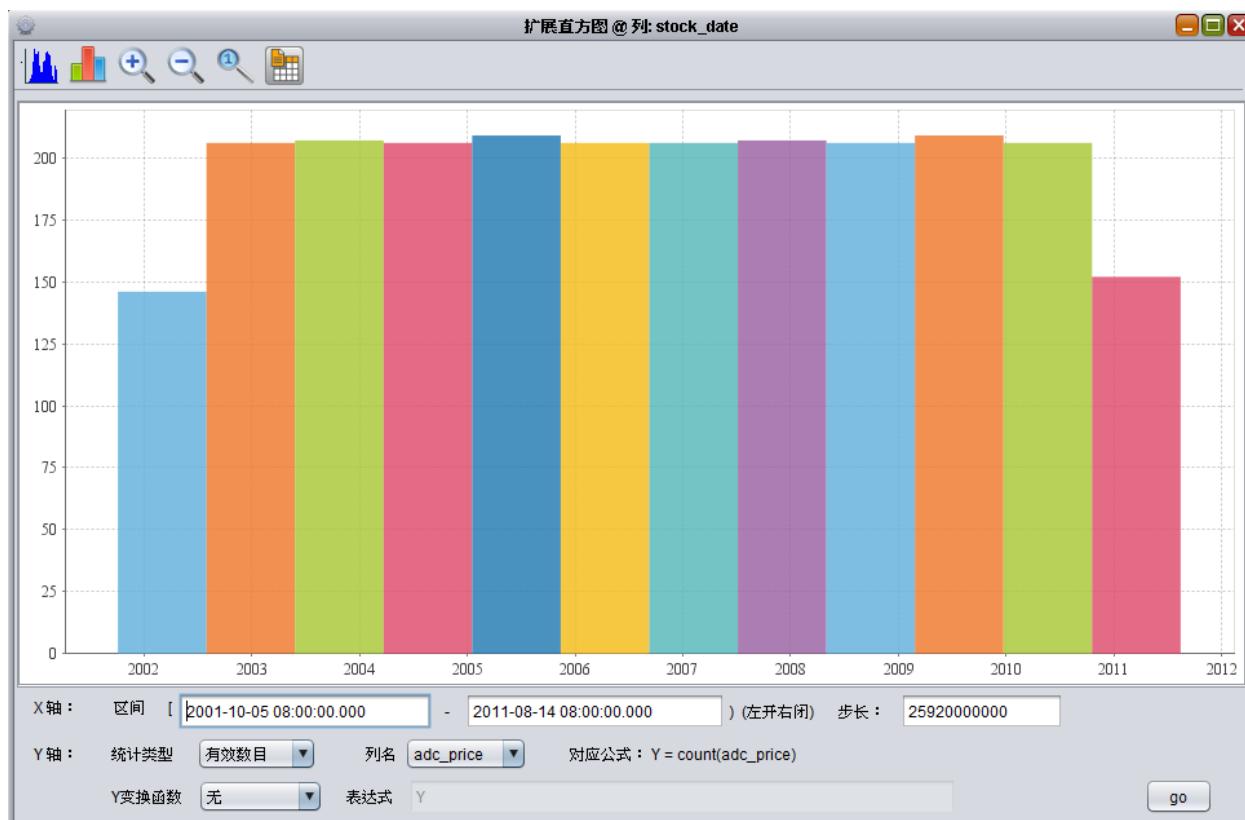
如果想从一个变量看另外一个变量，可以选择扩展直方图。

4.19.1 界面

统计->扩展直方图，选定x轴变量，选定y轴变量，点击确定，如下图：



运行成功后，显示如下：

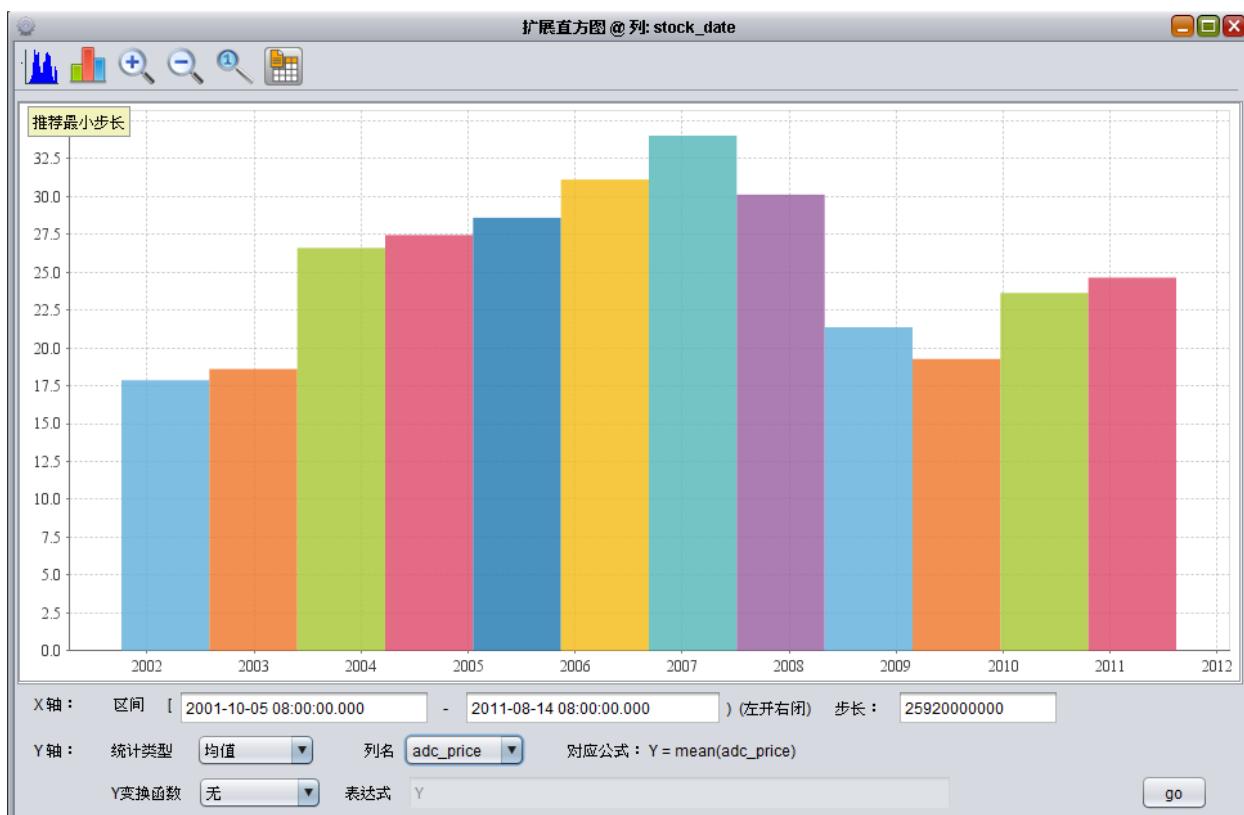


窗口上方工具栏同普通直方图一样，可以缩放图片。

窗口下方包括：
 * X轴：表示显示的x左右的范围，以及步长
 * Y轴：可以选择Y轴，并选择以哪种”统计类型”，进行画图。列名包含所有列，统计类型有：{“有效数目”，“缺失值数目”，“和”，“最小值”，“最大值”，“均值”，“方差”，“标准差”，“变异系数”，“标准误差”，“极差”，“偏度”，“峰度”，“二阶距”，“三阶距”，“四阶距”，“二阶中心距”，“三阶中心距”，“四阶中心距”}

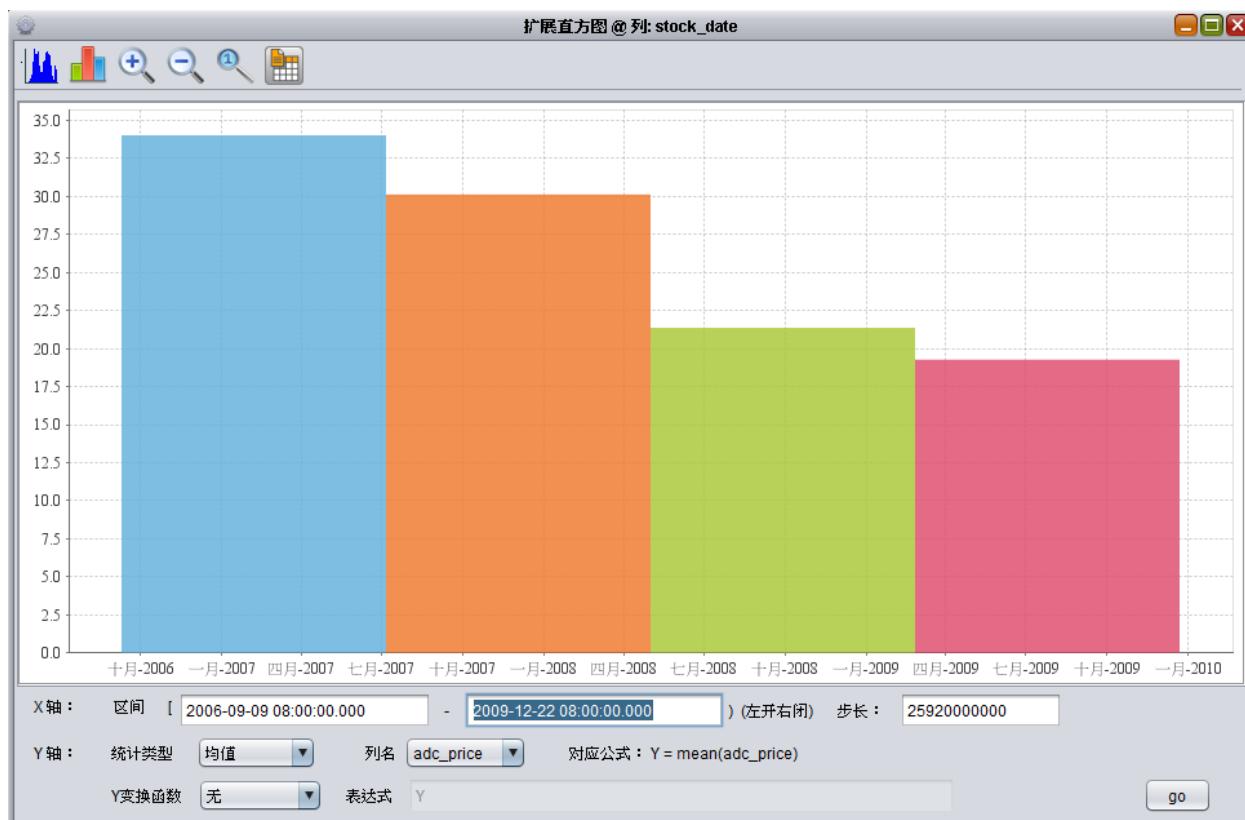
其中Y轴选择：dst_host_error_rateg 统计类型选择：adc_price

显示如下：

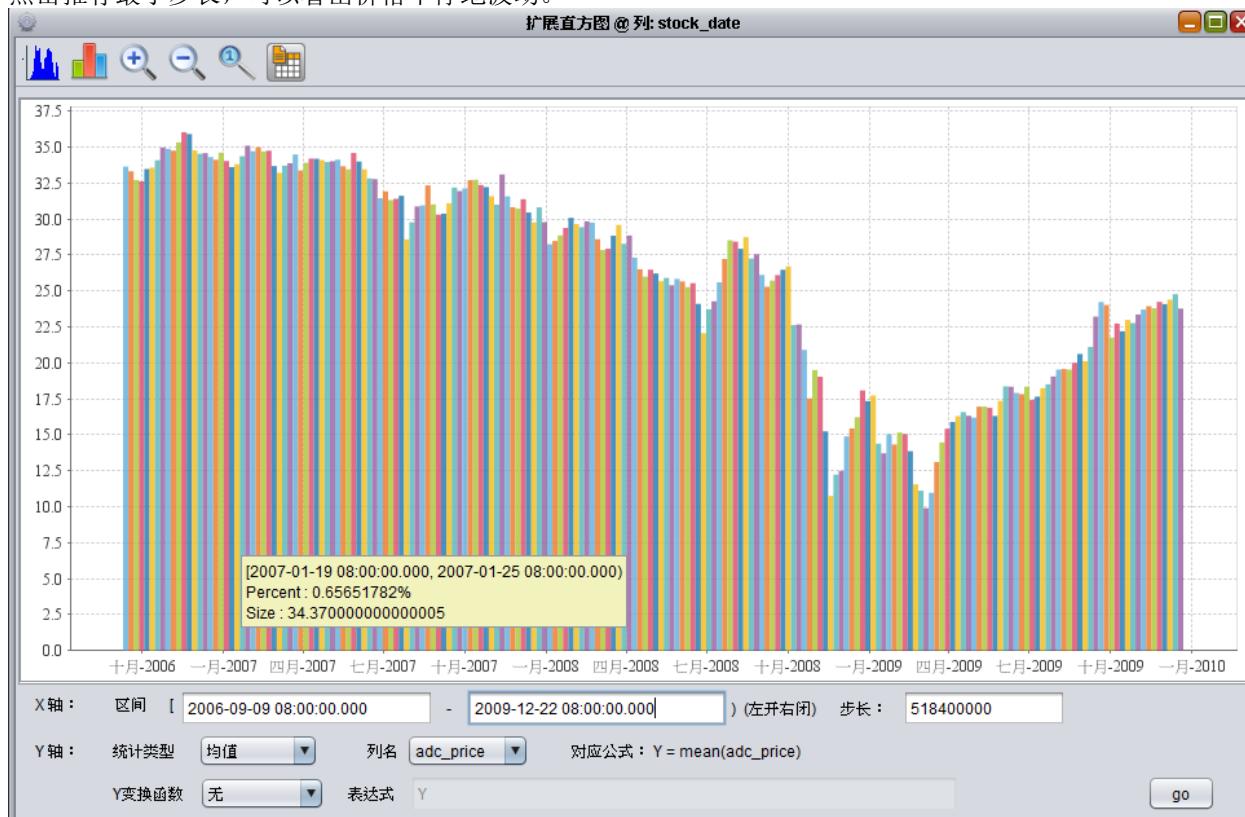


可以看出2007年的平均价格较高，2008年价格下降。

选择区间2006-09-09 08:00:00.000到2009-12-22 08:00:00.000，点击”GO”，可以看出趋势在下降，



点击推荐最小步长，可以看出价格不停地波动。



4.20 交叉表

交叉表是一种常用的分类汇总表格，是将某个表中的字段进行分组，一组在交叉表左侧，一组在交叉表上部，并在交叉表行与列交叉处显示源表满足行分组条件和列分组条件的记录条数，最后一行显示满足列分组条件的记录条数，最后一列显示满足行分组条件的记录条数，最后一行最后一列的位置显示原表的记录条数。

4.20.1 界面

点击菜单->统计->交叉表，输入行表达式列表和列表表达式列表，点击确定，运行结束后会弹出交叉表结果。

以UCI上Bank Marketing数据集为例，我们考虑观察用户年龄和婚姻状况的关系。其中age字段代表用户的年龄，marital字段代表用户的婚姻状况。由于年龄是连续的，将年龄分成6个区间 age<20, age>=20 and age<30, age>=30 and age<40, age>=40 and age<50, age>=50 and age<60, age>=60。婚姻状况只有三个状态，因此分成三个条件，marital=' married'，marital=' single'，marital=' divorced'。如下图：



运行结果如下图：

基本统计 画图								结果
col0	col1	col2	col3	col4	col5	col6	col7	
classf...	job='admin.'	job='blue-c...	job='entrepreneur'	job='housemaid'	job='m...'	job='r...'	job='...	
age>30	4205	8294	1352	1173	8255	2260	1331	
age<=30	966	1438	135	67	1203	4	248	
Total	5171	9732	1487	1240	9458	2264	1579	

4.20.2 函数

```
def crosstabs(tableName, rowExprs, colExprs, partitions=None):
```

参数:

- tableName: 输入表名.
- rowExprs: 行表达式(可以输入列名, 代表这一列所有不同的值).
- colExprs: 列表达式(可以输入列名, 代表这一列所有不同的值).
- partitions: 输入表的分区列表.

返回:

- CrossTabulation

示例1:

```
tableName = "bank_marketing"
rowExprs = ["age<20", "age>=20 and age<30", "age>=30 and age<40"]
colExprs = ["marital='married'", "marital='single'"]
ct = crosstabs(tableName, rowExprs, colExprs)
print ct
```

示例2:

```
tableName = "bank_marketing"
rowExprs = ["age<20", "age>=20 and age<30", "age>=30 and age<40"]
#如果是列名, 那么代表是这个列所有distinct值
colExprs = ["marital"]
ct = crosstabs(tableName, rowExprs, colExprs)
print ct
```

4.21 对比交叉表

如果用户使用交叉表, 不仅想观察个数, 希望观察其他的统计指标, 或者想看多列的情况, 可以使用对比交叉表。

4.21.1 界面

使用方式: 点击菜单->统计->对比交叉表, 运行结束后会弹出交叉表状结果和树状结果。

以UCI上Bank Marketing数据集为例, 我们考虑观察用户年龄, 婚姻状况和的关系。其中age字段代表用户的年龄, marital字段代表用户的婚姻状况。由于年龄是连续的, 将年龄分成6个区间 age<20, age>=20 and age<30, age>=30 and age<40, age>=40 and age<50, age>=50 and age<60, age>=60。婚姻状况只有三个状态, 因此分成三个条件, marital=' married', marital=' single', marital=' divorced'。如下图:



运行结果如下：

对比交叉表

col0	col1	col2	col3	col4
classification	marital='ma...'	marital='s...'	marital='d...'	Total
age<20	0	47	0	47
age>=20 and...	1386	3740	100	5226
age>=30 and...	9860	6728	1501	18089
age>=40 and...	8242	1670	1743	11655
age>=50 and...	6306	544	1560	8410
age>=60	1420	61	303	1784
Total	27214	12790	5207	45211

选列 统计量

可以调整列名和统计量，可以得到不同的交叉表。

4.22 排行榜

排行榜是用来计算在每个榜单分类下的排行。

4.22.1 界面

选择界面→统计→排行榜，如下图：



输入各项参数，点击确定， 运行结果如下图：

	group_job	group_marital	object_age	sum_balance_0	rank_
0	admin.	divorced	57	44765	1
1	admin.	divorced	46	43555	2
2	admin.	divorced	41	39897	3
3	admin.	divorced	60	38660	4
4	admin.	divorced	50	31478	5
5	admin.	divorced	37	31148	6
6	admin.	divorced	45	28707	7
7	admin.	divorced	43	27371	8
8	admin.	divorced	51	27268	9
9	admin.	divorced	38	24307	10
10	admin.	divorced	49	22317	11
11	admin.	divorced	34	21894	12
12	admin.	divorced	42	20940	13
13	admin.	divorced	36	20876	14
14	admin.	divorced	39	19337	15
15	admin.	divorced	58	19124	16

4.22.2 函数

```
Def topstat(tableName, rankingListGroupCols, rankingListObjectCols, aggregateStatCol, aggregateFunc, outputTableName, filter=None)
```

参数

- tableName: 输入表名
- rankingListGroupCols: 榜单分类列, 如果不分类, 可以输入[]
- rankingListObjectCols: 榜单主体列
- aggregateStatCol: 聚合的计算列
- aggregateFunc: 聚合函数. 包括count, countTotal, min, max, sum, mean, variance. 其中count是有效值个数, countTotal是总个数.
- outputTableName: 输出表
- filter: (可选)筛选条件, 默认是没有过滤条件
- partitions: (可选)输入表的partition
- outPartition: (可选)输出表的partition
- k: (可选)每个榜单的主体数目, 默认是100
- addedQuantityCols: (可选)附加的列
- addedQuantityFuncs: (可选)附加列的聚合函数

示例:

```
rankingListGroupCols = ["job", "marital"]
rankingListObjectCols = ["age"]
aggregateStatCol = "balance"
```

```
aggregateFunc = "sum"
Statistics.topstat("bank_marketing", rankingListGroupCols, rankingListObjectCols, aggregateStatCol, aggregateFunc, "ranking_list")
```

计算的是bank_marketing这个表在 job, marital的每一分类中 sum(balance) group by age的最大100个值

结果存在表中, 如下图:

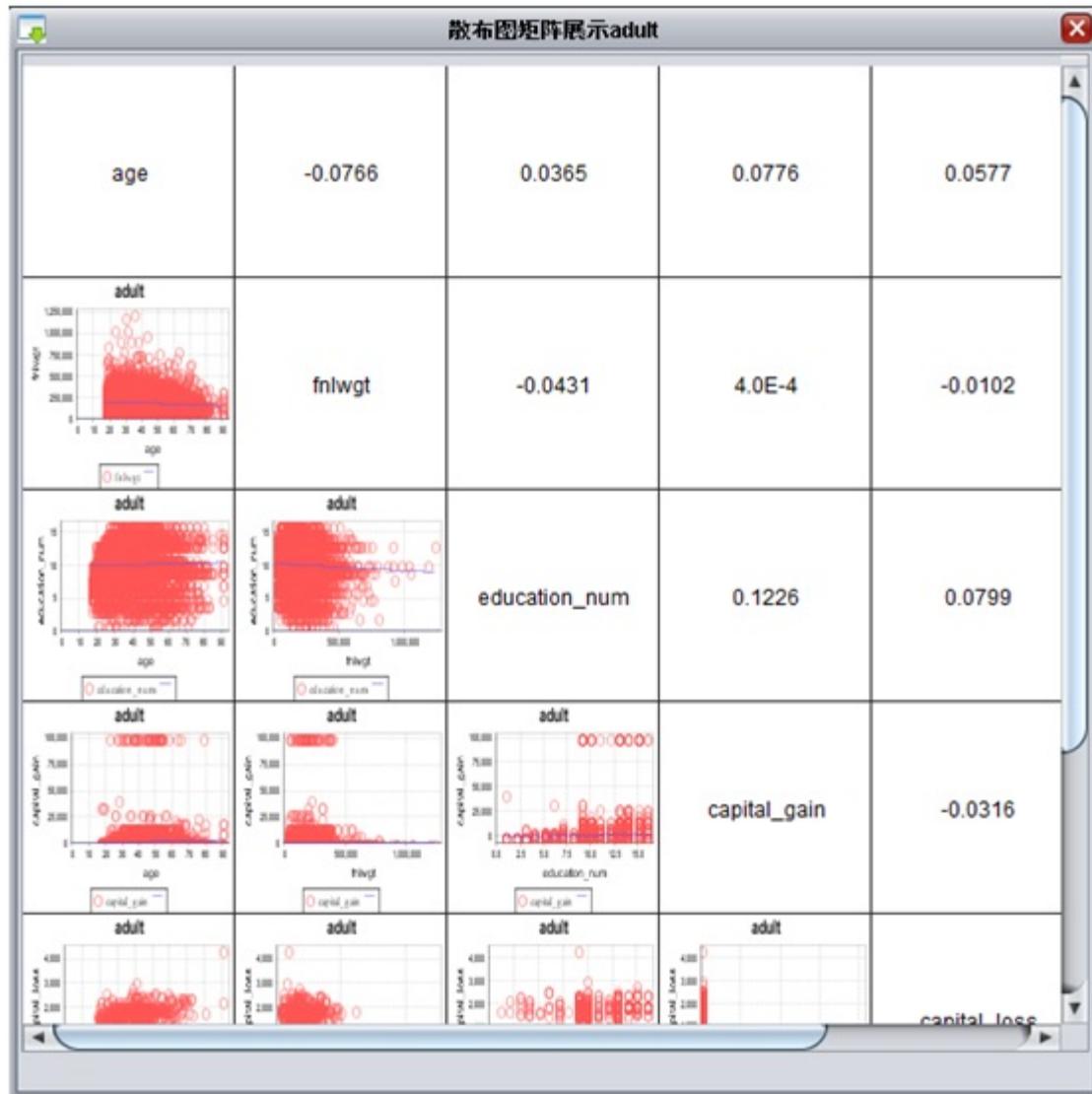
job	marital	age	sum(balance)	rank
admin.	divorced	57	44765.0	1
admin.	divorced	46	43555.0	2
admin.	divorced	41	39897.0	3

4.23 散布图矩阵

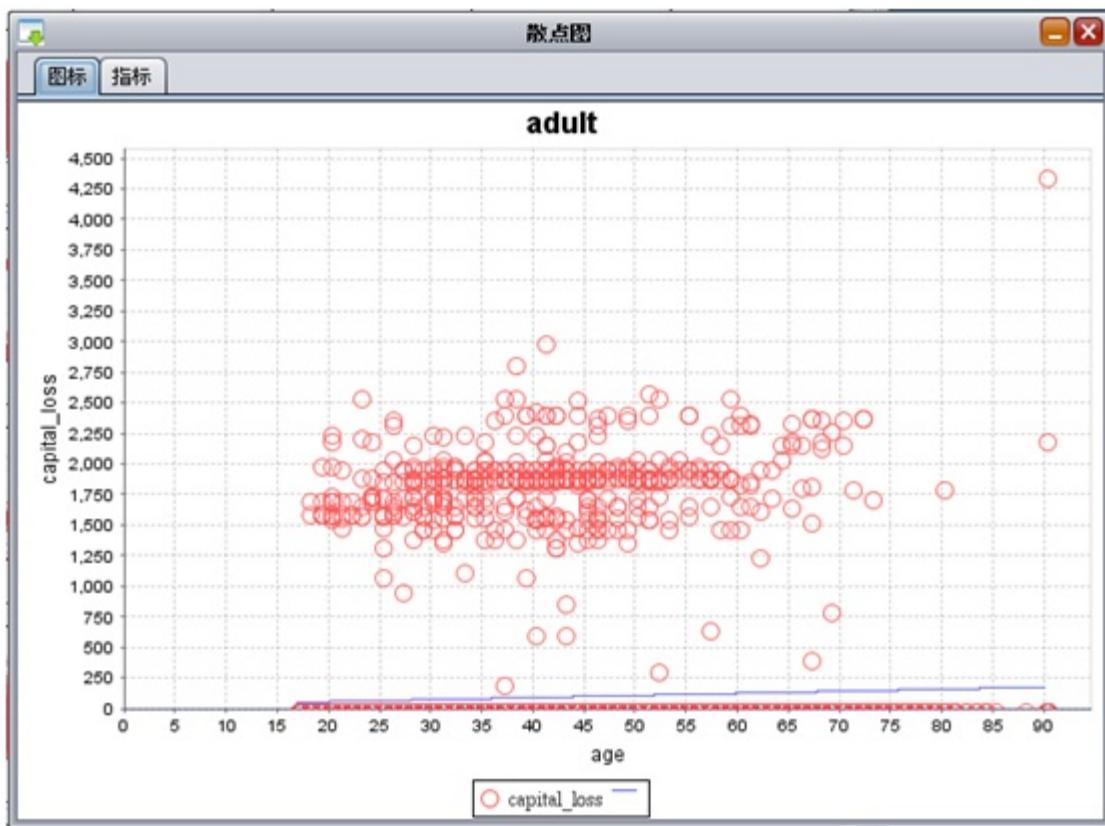
该操作用于展示数据表中每两列间的线性关系以及相关系数. 点击菜单: 全表描述→散布图矩阵, 显示参数配置窗体如下:



散布图矩阵只可选择数值类型的列参与计算， 默认选择所有列，可根据需求进行列的删减与重选。运行结果如下：



图中对角线为所选列名，矩阵右上区域为每两列的相关系数，矩阵左下区域为每两列的线性关系展示，可以点击左下区域单元格查看大图：



图中蓝线代表了所选两列的线性关系，红色的点代表样本空间中的数据分布。

4.24 按行统计

按行统计是指统计每行的一些统计值，目前包括：countTotal count countMissValue colNameOfMax max colNameOfMin min mean sum sum2 sum3 median variance standardDeviation

- countTotal: 总个数
- count: 有效值个数
- countMissValue: 缺失值个数
- colNameOfMax: 最大值的列名
- max: 最大值
- colNameOfMin: 最小值的列名
- min: 最小值
- mean: 均值
- sum: 和
- sum2: 平方和

- sum3: 立方和
- median: 中位数
- variance: 方差
- standardDeviation: 标准差

4.24.1 函数

stat, 具体使用方法, 可用help命令查看, 示例:

```
help(StatisticsByRow.stat)
```

按行统计

```
def stat(inputTableName, selectedColNames, outputTableName, \
functions = None, inputPartitions = None, outputPartition = None, outputColNames = None, appendColNames = None):
```

参数:

- inputTableName 输入表
- selectedColNames 待统计列, 仅支持数值类型
- outputTableName 输出表
- functions 统计函数: countTotal count countMissValue colNameOfMax max colNameOfMin min mean sum sum2 sum3 median variance standardDeviation
- inputPartitions (可选) 输入表分区信息
- outputPartition (可选) 输出表分区信息
- outputColNames (可选) 输出列名
- appendColNames (可选) 附加列

示例1:

```
Table.drop("stat_by_row_test0_outed")
StatisticsByRow.stat("stat_by_row_test0", ["c0", "c1"], "stat_by_row_test0_outed")
```

示例1:

```
inputTableName = "stat_by_row_test2"
selectedColNames = ["c0", "c1", "c2"]
outputTableName = "stat_by_row_test2_outed"
functions = ["colNameOfMax", "max", "colNameOfMin", "min"]
outputColNames = ["colNameOfMax", "max", "colNameOfMin", "min"]
appendColNames = ["doc_id", "c0", "c1", "c2"]
```

```
inputPartitions = ['pt=1', 'pt=2', 'pt=3']
outputPartition = "pt=0001"
Table.drop(outputTableName)
StatisticsByRow.stat(inputTableName, selectedColNames, outputTableName, \
    functions, inputPartitions, outputPartition, outputColNames, appendColNames)
```


第 5 章

统计分析

在XLab中，统计分析主要是指对应分析(Correspondence analysis)、多维对应分析(Multiple correspondence analysis)、多重共线性(Multicollinearity)和主成分分析(PCA)，详细介绍见下文。

注解： 补全以及函数提示的快捷键：Ctrl+p。

5.1 对应分析

对应分析(Correspondence analysis)，是一种变量统计分析技术，可以揭示同一变量的各个类别之间的差异，以及两个变量各个类别之间的对应关系。它是一种视觉化的数据分析方法，它能够将两组看不出任何联系的数据，通过定位图展现出来。

进行对应分析的方式主要有两种：函数和界面。

5.1.1 界面

选择菜单->分析->对应分析，如下图：



用户通过填写”行选项”与”列选项” 进行对应分析。

5.1.2 函数

对应分析支持的函数为: corresp。该函数具体使用方法, 可用help命令查看, 示例:

```
help(StatAnalysis.corresp)
```

```
def corresp(inputTableName, rowLegend, rowTags, rowExprs, colLegend,
            colTags, colExprs, inputPartitions = None, srt = None):
```

参数:

- tableName: 输入表名
- rowLegend: 行说明
- rowTags: 行标签
- rowExprs: 行表达式(可以写列名, 代表这一列所有不同的取值)
- colLegend: 列说明
- colTags: 列标签
- colExprs: 列表达式(可以写列名, 代表这一列所有不同的取值)
- inputPartitions: (可选)输入表的分区
- srt: (可选)table和selectedDataPartitions下的SummaryResultTable.

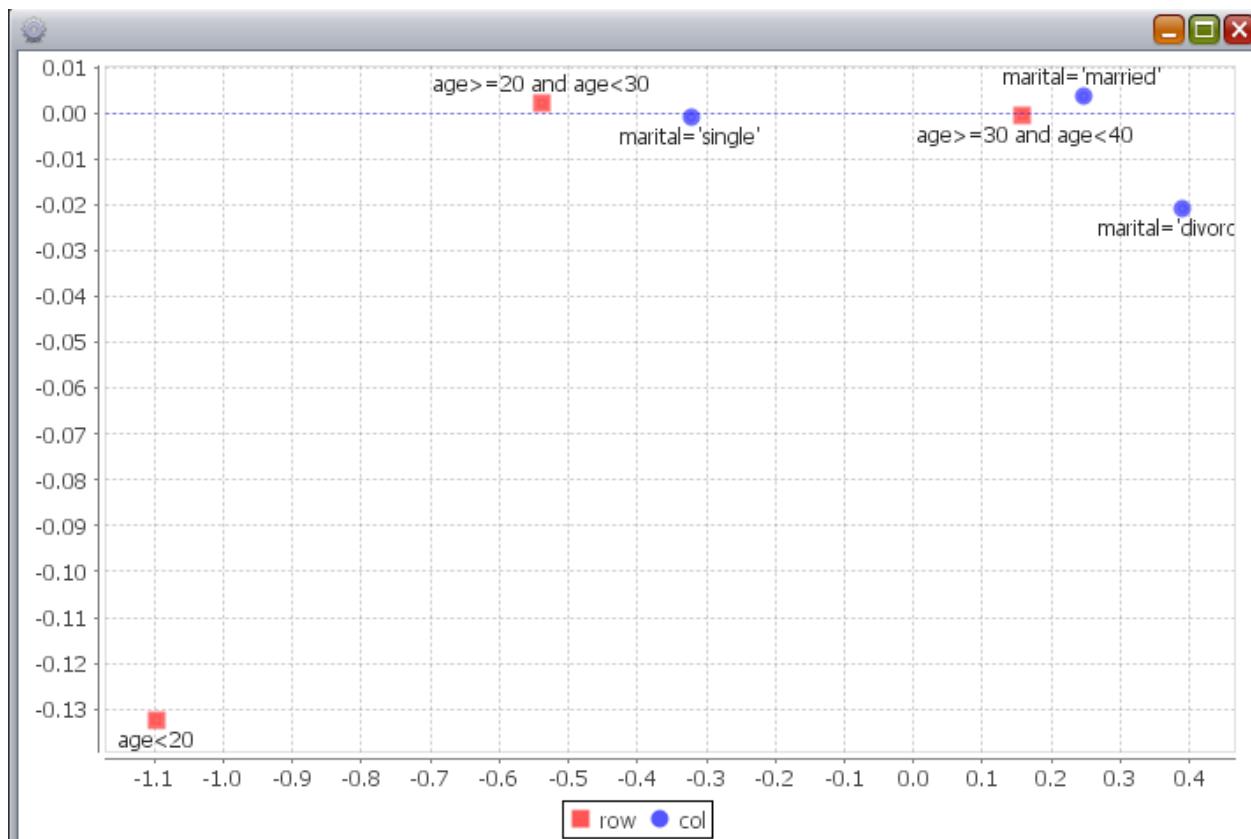
返回:

- CorrespondenceAnalysis

示例:

```
tableName = "bank_marketing";
rowExpr = ["age<20", "age>=20 and age<30", "age>=30 and age<40"]
colExpr = ["marital"]
tableSrt = Statistics.summary(tableName)
ca = StatAnalysis.corresp(tableName, "row", rowExpr, rowExpr,
                           "col", colExpr, colExpr, srt=tableSrt)
show(ca)
```

以UCI上Bank Marketing数据集为例, 我们考虑观察用户年龄和婚姻状况的关系。其中age字段代表用户的年龄, marital字段代表用户的婚姻状况。由于年龄是连续的, 将年龄分成2个区间 age>=20 and age<30, age>=30 and age<40。婚姻状况只有三个状态, 只需要输入列名, 程序会自动分成三组。示例结果, 如下图:



由上图可以看出离婚和结婚和30~40关系比较紧密，单身和20~30关系比较紧密。

5.2 多维对应分析

多维对应分析(Multiple correspondence analysis)，是一种变量统计分析技术，可以揭示多个变量各个类别之间的对应关系。它是一种视觉化的数据分析方法，它能够将多组看不出任何联系的数据，通过定位图展现出来。

进行多维对应分析的方式主要有两种：函数和界面。

5.2.1 界面

选择菜单→分析→多维对应分析，如下图：



用户通过增加维度，填写每个维度的条件，进行多维对应分析。

5.2.2 函数

多维对应分析支持的函数为：multicorresp。该函数具体使用方法，可用help命令查看，示例：

```
help(StatAnalysis.m multicorresp)
```

```
def multicorresp(inputTableName, legends, tags, tagExprs, inputPartitions = None, srt = None):
```

参数：

- tableName: 输入表名
- legends: 每个特征说明

- tags: 标签
- tagExprs: 表达式(可以写列名, 代表这一列所有不同的取值)
- inputPartitions: (可选)输入表的分区
- srt: (可选)table和inputPartitions下的SummaryResultTable.

返回:

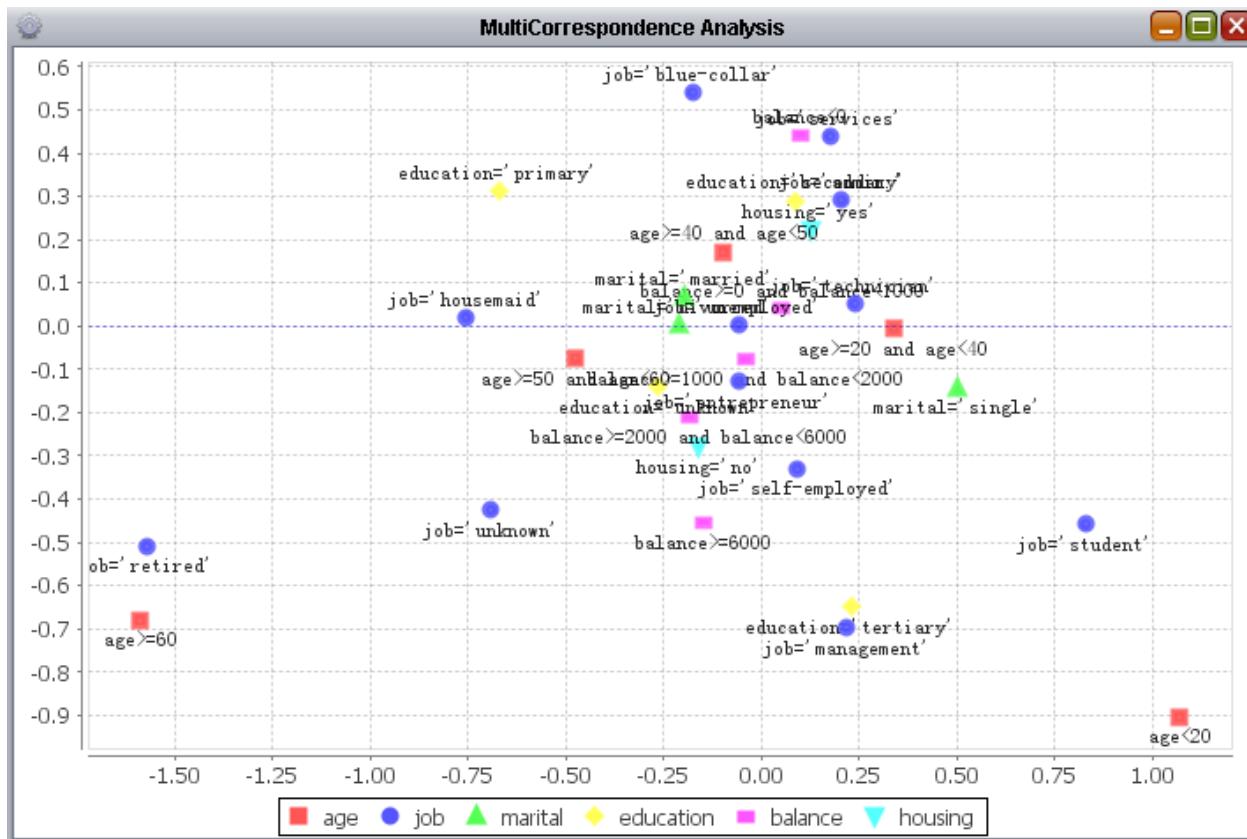
MultiCorrespondenceAnalysis.

可以看出多维对应分析, 如果维数是二, 那么就是对应分析。

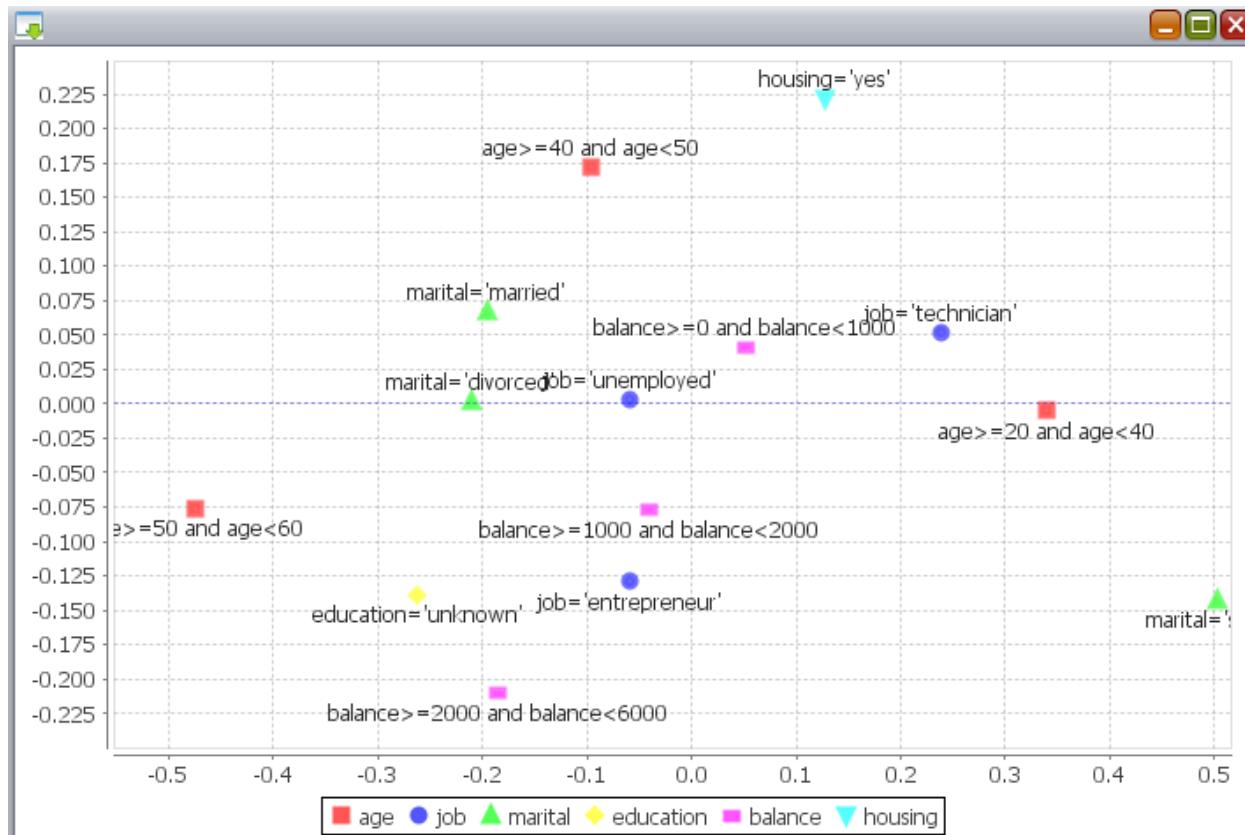
示例:

```
tableName = "bank_data"
ageExpr = ["age<20", "age>=20 and age<40", "age>=40 and age<50", "age>=50 and age<60", "age>=60"]
jobExpr = ["job"]
maritalExpr= ["marital"]
educationExpr = ["education"]
balanceExpr = ["balance<0", "balance>=0 and balance<1000",
               "balance>=1000 and balance<2000", "balance>=2000 and balance<6000", "balance>=6000"]
housingExpr = ["housing"]
tagExprs = [ageExpr, jobExpr, maritalExpr, educationExpr, balanceExpr, housingExpr]
legends = ["age", "job", "marital", "education", "balance", "housing"]
mca = StatAnalysis.multicorresp(tableName, legends, tagExprs, tagExprs)
show(mca)
```

以UCI上Bank Marketing数据集为例, 我们考虑观察用户年龄, 婚姻状况, 教育程度, 工作状况, 净收入, 是否有房贷之间的关系。其中age字段代表用户的年龄, marital字段代表用户的婚姻状况。由于年龄和净收入是连续的, 将年龄分成6个区间, 将净收入分成5个区间。婚姻状况, 工作状况, 教育程度, 是否有房贷等变量只有几个状态, 只需要输入列名, 程序会自动分组。结果, 如下图:



从上图可以看出，小于20岁的人群和学生比较相关，大于60的人群和退休比较相关，受过高等教育的和管理者比较相关。放大中间区域，如下图：



从上图可以看出，技术人员和20到40之间关系比较近，40到50的人群和有房贷人群关系较近。

5.3 多重共线性

所谓多重共线性 (Multicollinearity) 是指线性回归模型中的解释变量之间由于存在精确相关关系或高度相关关系而使模型估计失真或难以估计准确。一般来说，由于经济数据的限制使得模型设计不当，导致设计矩阵中解释变量间存在普遍的相关关系。完全共线性的情况并不多见，一般出现的是在一定程度上的共线性，即近似共线性。

5.3.1 函数

多维对应分析支持的函数为：collinear，共三种用法：

- 用法一：StatAnalysis.collinear(tableName)，tableName为数据表名，对该表的全部列进行共线性判断
- 用法二：StatAnalysis.collinear(tableName, colNames)，tableName为数据表名，colNames为选择需要判断共线性的列名数组。
- 用法三：StatAnalysis.collinear(srt, colNames)，srt为数据表的统计量，colNames为选择需要判断共线性的列名数组。

函数具体使用方法, 可用help命令查看, 示例:

```
help(StatAnalysis.collinear)

def collinear(tableName = None, colNames = None, srt = None, partitions = None):
```

参数:

- tableName: (可选)输入表名
- colNames: (可选)列名列表
- partitions: (可选)输入表的分区
- srt: (可选)table和inputPartitions下的SummaryResultTable.

返回:

- Multicollinearity

注解: 我们需要关注如下共线性的指标: 第一, 矩阵 $X^T X$ 的条件数 κ , 为矩阵 $X^T X$ 的最大特征值与最小特征值的比值。用 κ 判断多重共线性的准则: 1. $0 \leq \kappa < 100$ 时, 认为多重共线性的程度很小 2. $100 \leq \kappa \leq 1000$ 时, 认为存在中等程度或较强程度的多重共线性 3. $\kappa > 1000$ 时, 认为存在严重的多重共线性 第二, 条件指数与方差比例 在实际使用中, 若条件指数值在10与30间为弱相关, 在30与100间为中等相关, 大于100为强相关; 在大的条件指数中由方差比例超过0.5的自变量构成的变量子集就认为是相关变量集。第三, 方差膨胀因子与容忍度 方差膨胀因子(Variance Inflation Factor, 缩写为VIF)。对于全部自变量 X_1, X_2, \dots, X_m , 自变量 X_i 的方差膨胀因子记为 VIF_i , 计算表示如下: $VIF_i = 1/(1-R_i^2)$ 其中, R_i^2 是以 X_i 为因变量时对其它自变量建立多元线性回归模型的多重判定系数。因为多重判定系数 R_i^2 的取值区间为[0, 1], 所以 VIF_i 的取值区间为[1, $+\infty$]。 VIF_i 越接近1, 多重共线性越弱; 值越远大于1, 多重共线性越强。一般地, 当 $VIF_i < 10$ 时, 不存在多重共线性; 当 $10 \leq VIF_i < 100$ 时, 存在较强的多重共线性; 当 $VIF_i > 100$ 时, 存在严重多重共线性 容忍度(Tolerance)为方差膨胀因子(VIF)的倒数。取值范围为[0, 1], 容忍度越接近0, 表示多重共线性越强; 越接近于1, 多重共线性越弱。

示例:

```
corr = StatAnalysis.collinear("iris", colNames=["sepal_length", "sepal_width", \
"petal_length", "petal_width"])
print mcorr
```

运行结果:

Coefficient of Correlation:				
Variable	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.0	-0.10936924995067454	0.8717541573048854	0.8179536333691777
sepal_width	-0.10936924995067454	1.0	-0.4205160964011882	-0.35654408961382245
petal_length	0.8717541573048854	-0.4205160964011882	1.0	0.9627570970509656
petal_width	0.8179536333691777	-0.35654408961382245	0.9627570970509656	1.0
kappa	141.249002155887			

```

max lambda = 2.9108180837520914
min lambda = 0.020607707235620804
Min EigenVector:
0.2619955869000212 * sepal_length + -0.12413481006270811 * sepal_width + -0.8011542690799339 * petal_length + 0.523546271604
Number      Eigenvalue      Condition Index
1      2.9108180837520914    1.0
2      0.9212209307072055    1.777565586411462
3      0.14735327830508105   4.444548308278417
4      0.020607707235620804   11.88482234431323
Variance Proportions
Number      sepal_length      sepal_width      petal_length      petal_width
1      0.013197611312765672   0.01135134601935918   0.003696789023319418   0.006808870825877148
2      0.021184409657575256   0.4430182707196299   1.538487198030385E-5   2.877766119470164E-4
3      0.4966863511841683   0.18939499339389723   0.004290641925703285   0.1688888502881787
4      0.4689316278454907   0.35623538986711384   0.991997184178997   0.8240145022739972
VIF and TOL:
sepal_length  VIF:7.103113442835096  TOL:0.14078333508930496
sepal_width   VIF:2.0990386257425726  TOL:0.476408574733222
petal_length   VIF:31.39729165072643  TOL:0.03184988091088625
petal_width   VIF:16.141563956998493  TOL:0.06195186554809829

```

由上述结果指标, $\kappa=141.249002155887$, 说明存在相关性, 第4个条件指数为11.88482234431323, 也证明为相关, 根据第4行的方差比例知, 第3列和第4列相关; 第3个和第4个VIF值都超过了10, 也证明存在共线性。

5.4 主成分分析

主成分分析(Principal Component Analysis, PCA), 是指将多个变量通过线性变换以选出较少个数重要变量的一种多元统计分析方法。又称主分量分析。

在用统计分析方法研究多变量的课题时, 变量个数太多就会增加课题的复杂性。人们自然希望变量个数较少而得到的信息较多。在很多情形, 变量之间是有一定的相关关系的, 当两个变量之间有一定相关关系时, 可以解释为这两个变量反映此课题的信息有一定的重叠。主成分分析是对于原先提出的所有变量, 建立尽可能少的新变量, 使得这些新变量是两两不相关的, 而且这些新变量在反映课题的信息方面尽可能保持原有的信息。主成分分析, 详细介绍见: [Principal component analysis](#)。

进行主成分分析的方式主要有两种: 函数和界面。

5.4.1 函数

主成分分析支持的函数为: calc。该函数具体使用方法, 可用help命令查看, 示例:

```
help(StatAnalysis.PrinCompAnalysis.calc)
```

```
def calc(inputTableName, columnNames, eigOutputTableName,
         contriRate=0.9, calcuType="CORR", needTransform=False,
         princompOutputTableName=None, transType="Simple",
         remainColumns=None, srt=None):
```

参数:

- inputTableName: 进行主成分分析的输入表
- columnNames: 参与主成分分析运算的数值列
- eigOutputTableName: 特征向量与特征值的输出表
- contriRate: (可选) 解释百分比, 又称累计贡献率, 决定最终选择主成分的数目, 默认值0.9
- calcuType: (可选) 对原表进行特征分解的方式, 可以选择”CORR”, ”COVAR_SAMP”或”COVAR_POP”中的一种, 默认值”CORR”
- needTransform: (可选) 是否进行主成分转换, 默认值False
- princompOutputTableName: (可选) 进行主成分转换后的最终结果输出表
- transType: (可选) 原表转换为主成分表的方式, 包括”Simple”, ”Sub-Mean”, ”Normalization”, 即输入表直接乘以转移矩阵, 输入表每列减去本列均值再乘以转移矩阵, 输入表每列减去本列均值后乘以转移矩阵再除以主成分列的标准差, 默认值”Simple”
- remainColumns: (可选) 只在进行主成分转换时生效, 这些列都会出现在最终主成分结果表中
- srt: (可选) 输入表的SummaryResult

示例:

```
StatAnalysis.PrinCompAnalysis.calc("adult", ["age", "hours_per_week"], "pcaeigouttable1");
srt=Statistics.summary("adult")
StatAnalysis.PrinCompAnalysis.calc("adult", ["age", "hours_per_week"],
    "pcaeigouttable2", contriRate = 0.9, calcuType = "CORR", needTransform = True,
    princompOutputTableName = "pcatransresult", transType = "Simple",
    remainColumns =[ "age"], srt = srt);
```

5.4.2 界面

选择数据表, 点击数据统计→主成分分析, 如下图:



界面说明：

- 计算列：代表参与主成分分析计算的列，必须为数值型。
- 保留列：只在进行转换时生效，当“原表转换为主成分表”被选中时，保留列选择会被激活，所选的保留列都会出现在最终主成分结果表中。
- 分析模式：进行主成分分析时选取原表进行特征分解的方式，可以选择CORR, COVAR_SAMP和CO-VAR_POP，分别代表选择相关系数阵，样本协方差阵和总体协方差阵。
- 解释百分比：决定最终选择主成分的数目，在对特征值进行排序后，当前n个特征值的累计贡献率大于解释百分比时，就选取前n个特征向量作为主成分转移矩阵。
- 特征向量输出表：前n个特征向量即转移矩阵的输出表。
- 原表转换为主成分表：是否进行主成分转换，当此选项被勾选时，会根据转移矩阵进行原表到主成分

表的转换。

- 转换模式：原表转换为主成分表的方式，包括Simple, Sub-Mean, Normalization，即输入表直接乘以转移矩阵，输入表每列减去本列均值再乘以转移矩阵，输入表每列减去本列均值后乘以转移矩阵再除以主成分列的标准差。
- 转换结果输出表：进行主成分转换后的最终结果输出表。

运行结果：主成分分析的运行结果包括两部分：转移矩阵和转换出的主成分表，只有“原表转换为主成分表”被勾选时，才会进行原表到主成分表的转换。结果如下图：

特征向量表										
	prin_name	age	fnlwgt	education...	capital_g...	capital_lo...	hours_pe...	eigenvalue	contributi...	sumcontr...
0	prin0	0.3833...	-0.210...	0.5508...	0.4149...	0.2671...	0.5116...	1.3106...	0.2184...	0.2184...
1	prin1	-0.351...	0.5411...	0.1783...	0.5237...	-0.509...	0.1349...	1.0409...	0.1734...	0.3919...
2	prin2	0.3851...	-0.521...	-0.217...	0.3255...	-0.617...	-0.210...	1.0185...	0.1697...	0.5616...
3	prin3	-0.649...	-0.571...	0.3664...	-0.219...	-0.217...	0.1486...	0.9417...	0.1569...	0.7186...
4	prin4	0.1784...	0.1032...	-0.292...	-0.424...	-0.353...	0.7527...	0.8864...	0.1477...	0.8664...
5	prin5	-0.357...	-0.230...	-0.630...	0.4677...	0.3399...	0.2949...	0.8015...	0.1335...	1

主成分结果表										
	sl_lo...	hours_pe...	native_co...	class	prin0	prin1	prin2	prin3	prin4	prin5
0	40	Unite...	0	5.5047...	0.8842...	-0.966...	-0.007...	1.4215...	-3.284...	
1	13	Unite...	0	4.5614...	0.1810...	-0.320...	-0.822...	0.0501...	-4.368...	
2	40	Unite...	0	4.2227...	1.1867...	-1.435...	-1.212...	2.1233...	-2.717...	
3	40	Unite...	0	4.1781...	0.7590...	-0.937...	-2.314...	2.5656...	-2.662...	
4	40	Cuba	0	4.5535...	2.3513...	-2.662...	-0.832...	1.6579...	-3.704...	
5	40	Unite...	0	5.1278...	1.9125...	-2.227...	-0.826...	1.6093...	-4.067...	
6	16	Jamaica	0	2.7915...	0.0791...	-0.103...	-2.294...	1.2046...	-2.477...	
7	45	Unite...	1	4.8353...	0.8495...	-1.095...	-1.786...	2.6054...	-2.951...	
8	50	Unite...	1	6.6408...	1.9510...	-0.765...	0.4535...	1.0968...	-2.257...	
9	40	Unite...	1	5.5945...	1.4399...	-1.154...	-0.683...	1.3681...	-3.352...	
10	80	Unite...	1	5.9370...	2.0512...	-2.551...	-0.892...	4.4985...	-2.122...	
11	40	India	1	5.0025...	1.2891...	-1.632...	0.1401...	1.4912...	-3.326...	
12	30	Unite...	0	4.4293...	1.2628...	-1.565...	0.4559...	0.7714...	-3.339...	
13	50	Unite...	0	5.1320...	1.6042...	-1.976...	-0.322...	2.3030...	-3.033...	
14	40	?	1	4.8942...	0.7925...	-1.084...	-0.514...	1.8303...	-3.055...	
15	45	Mexico	0	3.1874...	1.1508...	-1.358...	-1.836...	2.9734...	-1.333...	
16	35	Unite...	0	3.7277...	1.2679...	-1.525...	-0.443...	1.6104...	-2.411...	
17	40	Unite...	0	4.1115...	1.1936...	-1.462...	-0.771...	2.0166...	-2.497...	
18	50	Unite...	0	4.5810...	0.1998...	-0.514...	-0.366...	2.7776...	-1.580...	
19	45	Unite...	1	5.4885...	1.8514...	-2.180...	-1.092...	2.0000...	-4.122...	
20	60	Unite...	1	6.6505...	1.7255...	-2.202...	0.0494...	2.5513...	-3.959...	
21	20	Unite...	0	3.6713...	0.9990...	-1.070...	-2.683...	1.1980...	-3.803...	
22	40	Unite...	0	3.5586...	0.2751...	-0.496...	-0.888...	2.4030...	-1.355...	
23	40	Unite...	0	5.4851...	-2.165...	-3.769...	-2.303...	0.5283...	-0.419...	
24	40	Unite...	0	5.0254...	0.0984...	-0.315...	-1.634...	2.2937...	-3.034...	
25	40	Unite...	1	5.5827...	1.0060...	-1.271...	-1.506...	1.9052...	-4.172...	
26	40	Unite...	0	3.7831...	1.4339...	-1.738...	-0.052...	1.8285...	-2.116...	
27	60	South	1	5.7859...	0.8803...	-1.234...	-1.399...	3.4035...	-2.826...	
...

第 6 章

数据处理

在XLab中，数据处理主要是只对数据进行这些操作：数据过滤、连续变量分组、唯一结果集、前N条记录、随机采样、加权采样、数据拆分、追加ID列、多表列合并、多表行合并、排序、信息值、变量转换、缺失值填充、归一化、标准化、分箱、数据生成等。这些处理的运行方式包括：函数和界面，详见下文。

注解： 补全以及函数提示的快捷键：Ctrl+p。

6.1 数据过滤

数据过滤可以对数据按照过滤表达式进行数据的筛选。

6.1.1 界面

点击菜单：数据处理→数据过滤，显示数据过滤参数窗体如下：



窗体分为五部分，包括选择分区，筛选条件，结果保存列，输出表名和运行信息。

- 选择分区：如果不修改是全表，点击修改，出现下面界面，可以通过条件筛选或者勾选，最终以勾选结果确定选择的分区。



其中：

- 筛选条件：目前操作符支持”=” , ”!=”, ”>”, ”<”, “>=”, “<=” 和” like”
- 结果保存列：点击修改，选择数据过滤后希望留下的列名，用户可以选择列名后使用中间的按钮进行对于选中和未选中的列调整。如果没有点击修改，那么所有列都会被保存。
- 输出表名：不存在的表名。
- 运行信息：填写完以上信息之后，点击确定，程序开始运行之后，显示运行的进度和信息。

计算过程中可以点击取消按钮取消计算同时终止云端JOB。

任务完成后结果页面跳出，仍然以表格方式显示，对于结果各项操作同原数据表格，用户可以针对结果数据继续进行计算。

6.1.2 函数

```
def filter(inputTableName, filter, outTableName, inputPartitions = None, cols = None):
```

参数：

- inputTableName: 输入表名.
- outTableName: 输出表名.
- filter: 过滤条件, 这个过滤条件为表达式, 语法可参见表达式.
- inputPartitions: (可选)输入表的分区列表.
- cols: (可选) 结果保存列, 默认全选

返回:

outTableName的SummaryResultTable.

示例:

```
srt = DataProc.filter("bank_marketing", "age>30", "ning_test_k1")  
  
srt = DataProc.filter("bank_marketing", "age>30 and day=5", "test_k3")  
  
srt = DataProc.filter("bank_marketing", "age>30 or day=5", "test_k4")
```

6.2 连续变量分组

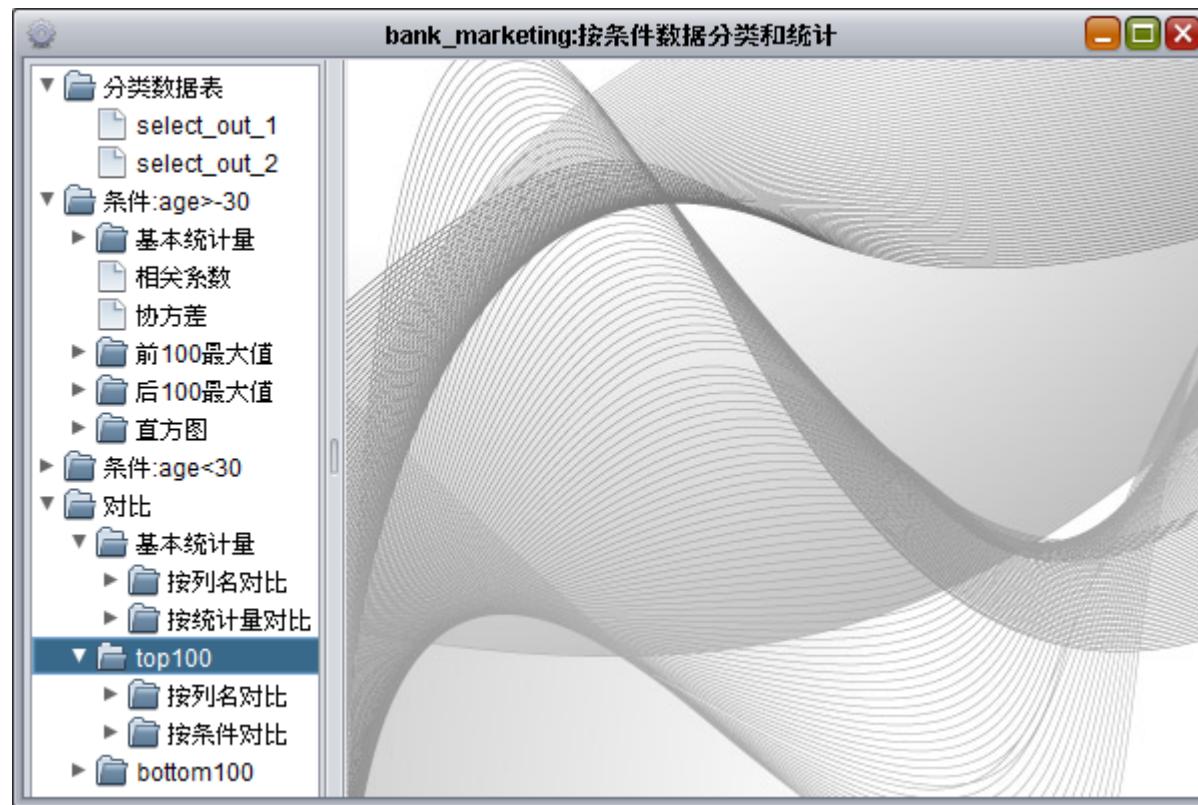
6.2.1 界面

如果想看一个变量多组过滤条件之间统计结果的对比, 可以使用连续变量分组。统计对比->连续变量分组, 输入条件列表, 输入条件列表对应的输出表列表。如果有输出表列表, 那么结果中包含过滤表。

以UCI上Bank Marketing数据集为例, 将age字段分成小于等于30岁和大于30岁两组, 考虑观察每组的balance字段的统计量以及balance字段统计量的对比。如下图:



点击确定，得到运行结果如下：



这是个树状结构，左侧是数据表和条件列表，可以看到每个条件的统计量，以及所有条件直接的对比。对比分为基本统计量的对比，top100的对比，bottom100的对比。如果选定基本统计量按列对比的结果，行显示的是两个分类条件，列显示的是基本统计量。

bank_marketing:按条件数据分类和统计

Statistics	age>30	age<30
countTotal	45211.0	5273.0
count	45211.0	5273.0
countMissValue	0.0	0.0
countNaNValue	0.0	0.0
countPositiveInfinity	0.0	0.0
countNegativInfinity	0.0	0.0
sum	6.1589682E7	5202185.0
sum2	5.03025681562E11	2.9629773521E10
sum3	1.2495578792439942E16	3.74393098375241E14
sum4	6.214329546211325E20	7.1437043238428324E18
min	-8019.0	-8019.0
max	102127.0	36252.0
range	110146.0	44271.0
mean	1362.2720576850766	986.5702636070548
variance	9270598.954472251	4646709.501938038
standardDeviation	3044.765829168518	2155.6227642929634
cv	2.235064436646026	2.1849662855351735
standardError	14.319631447168446	29.685466326884164
skewness	8.360030947252685	5.621432413720926
kurtosis	140.7358483257726	48.17515519767856
moment2	1.1126179061777001E7	5619149.160060686
moment3	2.7638359674503864E11	7.100191511004001E10
moment4	1.3745171631265234E16	1.3547704008804918E15
centralMoment2	9270393.902627468	4645828.275026993
centralMoment3	2.3596911674581204E11	5.629135759001861E10
centralMoment4	1.2352687999286728E16	1.1045502392195314E15

如果选定基本统计量->按统计量对比->count, 那么结果显示的是所有分类条件和所有列的count值的对比。

bank_marketing:按条件数据分类和统计

Classification	balance
age>30	45211
age<30	5273

注意事项： 如果选定基本统计量->按统计量对比->count, 那么结果显示的是所有分类条件和所有列的

count值的对比。

6.2.2 函数

```
def select(inputTableName, by, outTableNames, inputPartitions = None, cols = None):  
  
    * inputTableName: 输入表名.  
    * by: 分组条件列表.  
    * outTableNames: 输出表名列表. 和by参数的个数相同.  
    * inputPartitions: (可选)表的分区列表.  
    * cols: (可选)需要计算的列名列表.
```

返回:

- 如果by=None, 或者by只有一个条件, 返回SummaryResultTable.
- 否则返回SummaryResultTable 列表, 其中每一个代表在filter条件下, by分组条件对应的summary结果.

示例:

```
[srt1, srt2] = DataProc.select("bank_marketing", ["age>30", "age<=30"], ["outTableName1", "outTableName2"])
```

6.3 唯一结果集

该操作类似数据库中的distinct功能, 用以将数据对于若干列进行去重操作。

6.3.1 界面

点击菜单: 数据处理→唯一结果集, 显示参数窗体如下:



参数选择完毕后点击确定进行计算，运行时候查看任务状态，任务中途取消和任务结束后结果数据的显示参照数据过滤功能。

6.3.2 函数

```
def unique(inputTableName, outTableName, keyColNames = None):
```

参数:

- inputTableName: 输入表名.
- outTableName: 输出表名.
- keyColNames: (可选)取唯一结果的列, 默认为全选.

示例:

```
DataProc.unique("bank_marketing", "outTableName", keyColNames=["job", "age"])
```

6.4 前N条记录

该操作条件类似于排序功能，计算该数据按照条件排序后的前n条记录。

6.4.1 界面

点击菜单：数据处理→前N条记录，显示参数窗体如下：



参数选择完毕后点击确定进行计算，运行时候查看任务状态，任务中途取消和任务结束后结果数据的显示参照数据过滤功能。

6.4.2 函数

```
def topn(inputTableName, N, outTableName, keyColNames=None, momotonicity=None):
```

参数：

- `inputTableName`: 输入表名.
- `N`: 结果的记录个数.
- `outTableName`: 输出表名.
- `keyColNames`: (可选)求top的列, 默认为全选.
- `momotonicity`: (可选)排序规则, 默认为升序, 其中”+”是升序, “-”是降序.

示例：

```
DataProc.topn("bank_marketing", 5, "outTableName", keyColNames=["age", "job"], momotonicity=["+", "+"])
```

6.5 随机采样

该操作以随机方式生成采样数据。对于随机采样，单次采样不放回，每次采样是各自独立的。权重值越大的被选中概率越大。

6.5.1 界面

点击菜单：数据处理→采样→随机采样，显示参数窗体如下：



用户可输入采样数，并选择该采样是否是放回采样。参数选择完毕后点击确定进行计算，运行时候查看任务状态，任务中途取消和任务结束后结果数据的显示参照数据过滤功能。

6.5.2 函数

```
def randomSample(inputTableName, sampleSize, outTableName, replace=False, inputPartitions=None):
```

参数：

- `inputTableName`: 输入表名.
- `outTableName`: 输出表名.

- sampleSize: 采样个数.
- replace: (可选)是否放回, 默认是不放回.
- inputPartitions: (可选)输入表的分区.

示例:

```
DataProc.Sample.randomSample("bank_marketing", 11, "outTableName")
```

6.6 加权采样

该操作以加权方式生成采样数据。

6.6.1 界面

点击菜单: 数据处理→采样→随机采样, 显示参数窗体如下:



用户可输入采样数, 加权列, 并选择该采样是否是放回采样。参数选择完毕后点击确定进行计算, 运行时时候查看任务状态, 任务中途取消和任务结束后结果数据的显示参照数据过滤功能。

6.6.2 函数

```
def weightedSample(inputTableName, sampleSize, probCol, outputTableName, replace=False, inputPartitions=None):
```

参数:

- inputTableName: 输入表名.
- outTableName: 输出表名.
- sampleSize: 采样个数.
- probCol: 加权列, 每个值代表所在record出现的权重, 不需要归一化.
- replace: (可选)是否放回, 默认是不放回.
- inputPartitions: (可选)输入表的分区.

示例:

```
DataProc.Sample.weightedSample("bank_marketing", 11, "age", "outputTable")
```

6.7 数据拆分

拆分把一个数据集按比例拆分成多个数据集。 支持两种模式的拆分，比例模式和K折模式。

- 比例模式是，把一个数据集按比例拆分成训练表，验证表和测试表。
- K折模式，是把数据集平均分成k分，输出到同一表的不同partition中。

6.7.1 界面

选择数据表adult，双击打开表>数据处理>拆分，如：



单击拆分，如：



可以选择两种拆分模式，比例模式和K折模式。

- 比例模式：按照比例分别输出到三张表中，其中概率和必须为100，如果某个概率为0，则不需填写输出表
- K折模式：把数据平均分成K分，输出到同一张表的不同partition中，其中partition的命名为K=N, $N \in [1, k]$

填写参数，单击“数据拆分”，即可运行程序：



运行成功，会自动弹出结果表：

adult_dp_10 - 数据表浏览

	age	workclass	fnlwgt	education	education...	marital_s...	occupation	relationship	race	sex
0	38	Private	215646	HS-grad	9	Divorced	Handle...	Not-in...	White	Male
1	40	Private	193524	Doctorate	16	Marrie...	Prof-s...	Husband	White	Male
2	56	Local-gov	216851	Bachelors	13	Marrie...	Tech-s...	Husband	White	Male
3	47	Private	51835	Prof-s...	15	Marrie...	Prof-s...	Wife	White	Femal
4	41	Private	147372	HS-grad	9	Marrie...	Adm-cl...	Husband	White	Male
5	30	Private	59496	Bachelors	13	Marrie...	Sales	Husband	White	Male
6	36	Private	155537	HS-grad	9	Marrie...	Craft...	Husband	White	Male
7	29	Self-e...	162298	Bachelors	13	Marrie...	Sales	Husband	White	Male
8	42	Private	124692	HS-grad	9	Marrie...	Handle...	Husband	White	Male
9	20	Private	34310	Some-c...	10	Never...	Sales	Own-child	White	Male
10	64	Private	181232	11th	7	Marrie...	Craft...	Husband	White	Male
11	43	?	174662	Some-c...	10	Divorced	?	Not-in...	White	Femal
12	47	Local-gov	186009	Some-c...	10	Divorced	Adm-cl...	Unmarried	White	Femal
13	48	Private	187715	HS-grad	9	Marrie...	Craft...	Husband	White	Male
14	68	?	38317	1st-4th	2	Divorced	?	Not-in...	White	Femal
15	44	Self-e...	78374	Masters	14	Divorced	Exec-m...	Unmarried	Asian...	Femal
16	28	Private	88419	HS-grad	9	Never...	Exec-m...	Not-in...	Asian...	Femal
17	28	State-gov	149624	Bachelors	13	Marrie...	Prof-s...	Husband	White	Male
18	35	Private	138992	Masters	14	Marrie...	Prof-s...	Other...	White	Male
19	18	Private	140164	11th	7	Never...	Sales	Own-child	White	Femal
20	37	Private	635913	Bachelors	13	Never...	Exec-m...	Not-in...	Black	Male
21	vate	313321	Assoc-...	12	Divorced	Sales	Not-in...	White	Femal	
22	k=0	vate	130408	HS-grad	9	Divorced	Sales	Unmarried	Black	Femal
23	k=1	te-gov	268700	HS-grad	9	Marrie...	Other...	Husband	White	Male
24	k=2	vate	115458	HS-grad	9	Never...	Transp...	Own-child	White	Male
25	k=3	vate	172579	HS-grad	9	Separated	Other...	Not-in...	White	Femal
26	k=4	vate	204518	7th-8th	4	Divorced	Craft...	Not-in...	White	Male
27	k=5	vate	245918	11th	7	Never...	Other...	Own-child	White	Male
28	k=6	-----	148012	-----	10	Marrie...	-----	-----	-----	-----

6.7.2 函数

```
def declustering(inputTableName, factors, outputTableNames, outputPartitions=None):
```

参数:

- inputTableName: 输入普通表名,
- factors: 拆分比例, double类型数组, 每个值范围(0, 1), sum(factors) = 1
- outputTableNames: 输出表名列表, outputTableNames和factors一一对应
- outputPartitions: 输出表分区列表, 如果不输出到分区, 则分区为empty string' '

示例:

```
DataProc.declustering('adult', [0.6, 0.3, 0.1], ['adult_train', 'adult_test', 'adult_validate'])
DataProc.declustering('adult', [0.3, 0.3, 0.4], ['adult_dp1', 'adult_dp2', 'adult_dp3'],
    ['pdate=20140318', 'pdate=20140319', 'pdate=20140320'])
```

6.8 追加ID列

在数据表第一列追加ID列

6.8.1 函数

```
def appendID(inputTableName, outputTableName, selectedColNames=None, IDColName=None)
```

参数:

- inputTableName: 输入普通表名
- outputTableName: 输出表名列表
- selectedColNames: (可选)输入表选择列名类型, 默认选择全部列
- IDColName: (可选)ID列列名, 默认为'append_id', ID从0开始

示例:

```
DataProc.appendID('adult', 'adult_append_id', IDColName='ID')
DataProc.appendID('adult', 'adult_append_id', selectedColNames=['age', 'workclass'], IDColName='ID')
```

6.9 多表列合并

将多个表按照列合并为一个表

6.9.1 函数

```
def appendColumns(inputTableNames, outputTableName, selectedColNamesList=None,
                  inputPartitionsInfoList=None, autoRenameCol=False, outputTableColNames=None, outputPartition=None)
```

参数:

- inputTableNames: 输入表名
- outputTableName: 输出表名
- selectedColNamesList: (可选)与输入表对应的选中列名列表(二维数组), 默认为全选
- inputPartitionsInfoList: (可选)与输入表对应的选中的partition列表(二维数组), 默认为全选
- autoRenameCol: (可选)输出表自动重定义列名, 默认为不自动重定义
- outputTableColNames : (可选)输出表自定义列名, 默认不指定
- outputPartition: (可选)输出partition, 默认为空

示例:

```
DataProc.appendColumns(['input_table1','input_table2'], 'output_table', autoRenameCol=True, outputPartition='p=1')
DataProc.appendColumns(['input_table1','input_table2'], 'output_table', selectedColNamesList=[['col0','col1'],['col3']])
DataProc.appendColumns(['input_table1','input_table2'], 'output_table', selectedColNamesList=[['col0','col1'],['col3']],
    outputTableColNames=['new_col0','new_col1','new_col3'])
```

6.10 多表行合并

将多个表按照行合并为一个表

6.10.1 函数

```
def appendRows(inputTableNames, outputTableName, selectedColNamesList=None, inputPartitionsInfosList=None,
    outputTableColNames=None, outputPartition=None)
```

参数:

- inputTableName: 输入表名
- outputTableName: 输出表名
- selectedColNamesList: (可选)与输入表对应的选中列名列表(二维数组), 默认为全选
- inputPartitionsInfosList: (可选)与输入表对应的选中的partition列表(二维数组), 默认为全选
- outputTableColNames: (可选)输出表自定义列名, 默认不指定
- outputPartition: (可选)输出partition, 默认为空

示例:

```
DataProc.appendRows(['input_table1','input_table2'], 'output_table', outputPartition='p=1')
DataProc.appendRows(['input_table1','input_table2'], 'output_table',
    selectedColNamesList=[['col0','col1'],['col2','col3']], outputTableColNames=['new_col0','new_col1'])
```

6.11 排序

数据按照自定义规则全排序

6.11.1 函数

```
def sort(inputTableName, outputTableName, selectedColNames=None, sortColRule=None)
```

参数:

- inputTableName: 输入表名
- outputTableName: 输出表名
- selectedColNames: (可选)输入表排序列名, 默认选择全部列
- sortColRule: (可选)排序规则, 长度必须和selectedColNames相等, +表示升序, -表示降序, 默认升序

示例:

```
DataProc.sort('input', 'output')
DataProc.sort('input', 'output', selectedColNames=['col0', 'col1'], sortColRule=['+', '-'])
```

6.12 信息值

目前InfoValue算法支持 属性列是连续类型, 目标列是二分类的计算

算法按N分划分数数据, 计算信息增益, 基尼增益, 信息增益率, 信息值, 其中计算信息值的中间结果可以输出到WOE表中

公式:

$$\text{WOE} = \text{Log}(\text{Distribution Good}/\text{Distribution Bad})$$

$$\text{IV} = \{\sum (\text{Dist Good} - \text{Dist Bad}) \times \text{WOE}\}$$

6.12.1 界面

双击adult, 菜单选择 分析>属性选择, 如:

adult - 数据表浏览					
描述	统计	分析	模型	工具	画图
workclass	fn	属性选择	tal_s...	occupation	
State-gov	71	随机森林	► r-...	Adm-cl...	
Self-e...	83	决策树C5	► ie...	Exec-m...	
Private	21	逻辑回归	► rced	Handle...	
Private	23	线性支持向量机(LinearSVM)	► ie...	Handle...	
Private	33	非线性支持向量机(NonlinearSVM)	► ie...	Prof-s...	
Private	28	朴素贝叶斯(Naive Bayes)	► ie...	Exec-m...	
Private	16	贝叶斯判别(Bayes)	► ie...	Other-...	
Self-e...	20	费希尔判别(Fisher)	► ie...	Exec-m...	
Private	45	马氏距离判别(MDistance)	► r...	Prof-s...	
Private	15		► ie...	Exec-m...	

普通表的属性选择界面如下：



其中：

- 选择属性列，默认选择数值列除去最后一列
- 选择目标列，默认选择数值列的最后一列
- 算法类型，按N分位数且 和 探索所有二分点
- 缺失值处理方式 忽略和替换，可以自定替换值
- 信息值IV选项，调整步长，当概率为0时，调整概率的大小

- WOE值输出表, 当算法类型是按N分位切, 可以输出IV的计算过程

稠密矩阵的属性选择界面如下:



其中:

- 选择属性列, 稠密矩阵不支持选择
- 选择目标列, 默认选择数值列的最后一列
- 算法类型, 按N分位且 和 探索所有二分点
- 缺失值处理方式 忽略和替换, 可以自定替换值

- 信息值IV选项， 调整步长， 档概率为0是， 调整概率的大小
- WOE值输出表， 当算法类型是按N分位切， 可以输出IV的计算过程

填写完参数，单击确定：



运行成功，自动弹出结果结果表，列说明：

- featureindex: 特征列序号
- infogain: 信息增益
- ginigain: 基尼增益

- infogainratio: 信息增益率
- iv: 信息值

featureindex	infogain	ginigain	infogainrate	iv
0	0	0	0	0
1	0.0993483330448709	0.06444444444444442	0.03905566900265868	0.5948438866349601

Woe表, 列说明:

- Var_name: 特征列序号
- Seg_var: 分位点
- Var_range: 分位点对应的值
- Population: n_event_attr + n_nonevent_attr
- n_event_attr: good的分布
- n_nonevent_attr: bad的分布
- event_rate: n_event_attr / n_nonevent_attr
- p_nonevent : n_nonevent_attr / Σ (n_nonevent_attr)
- p_event: n_event_attr / Σ (n_event_attr)
- woe: $\log(p_{\text{nonevent}} / p_{\text{event}})$;

var_index	seg_var	var_range	population	n_event...	n_noneve...	event_rate	p_noneve...	p_event	woe
0	0	<1	10	5	5	1	0.3333...	0.3333...	0
1	0	<=2	20	10	10	1	0.6666...	0.6666...	0
2	1	<3	5	3	2	1.5	0.1333...	0.2	-0.405...
3	1	<5	3	2	1	2	0.0666...	0.1333...	-0.693...
4	1	<7	6	1	5	0.2	0.3333...	0.0666...	1.6094...
5	1	<8	4	2	2	1	0.1333...	0.1333...	0
6	1	<9	6	3	3	1	0.2	0.2	0
7	1	<=10	6	4	2	2	0.1333...	0.2666...	-0.693...

6.12.2 函数

```
class InfoValue:
    @staticmethod
```

```
def calc(inputTableName, labelColName, outputTableName, outputWOETableName="",
         nDivide=10, missingType=0, missingReplace=0, filterType=0):
```

参数:

- inputTableName: 输入表名, 可以为普通表或稠密矩阵
- labelColName: 目标列的索引或列名, 如果输入表为稠密表, 则为列序号
- outputTableName: 输出表名.
- outputWOETableName: (可选) WOE输出表
- nDivide: (可选) 每次计算数据被平均划分成N份, 取值(2, 100]
- missingType: (可选) 缺失值处理, 0表示忽略; 1 表示替换, 用missingReplace替换; 默认忽略缺失值
- missingReplace: (可选) 如何missingType = 1, 起作用, 缺失值被替换成missingReplace
- filterType: (可选) 过滤类型, 0 不忽略任何值; 1 忽略0值; 2: 忽略<=0的值

示例:

```
DataProc.InfoValue.calc("dshen1_tmall_new_test", "v2", "iv_output", \
    "woe_output", 10, 1, 0.0, 2)

DataProc.InfoValue.calc("denseinputtable_2_group", 2, "denseinputtable_2_group_iv_output", \
    "denseinputtable_2_group_woe_output", 10, 1, 0.0, 2)
```

6.13 变量转换

变量转换可以根据数据表中已有变量参与条件表达式, 根据需求生成新的变量。提供多种操作符与函数, 可以较灵活的生成变量。

6.13.1 界面

选择数据表, 点击数据处理→变量转换:



在右侧的可选列名与可选函数列表中选择变量与函数生成新的变量表达式，也可根据需要输入变量表达式内容，对新增变量命名后即可将其添加到下方的表达式列表部分。也可对表达式列表中的表达式进行编辑或删除。

- 新增变量名称：新增变量在结果表中的列名。
- 新增变量类型：新增变量的类型，如不指定则会自动判断。
- 新增变量表达式：根据列名和可选函数构成的表达式，只有合法的表达式才会被添加到下方的表达式列表部分。
- 输出表名：最终结果输出表。
- 保留原表所有列：勾选时在结果表中保留原表所有列，不勾选则不保留原表，只保留新增变量列。

运行结果： 变量转换会把输出结果表展示出来，如下图所示：

数据转换结果表										
	native_country	race	sex	capit...	capita...	hour...	class	newvari...	incom...	▲
0	United-States	White	Male	0	0	13	0	83361	1	
1	United-States	Black	Male	0	0	15	0	311534	1.5	
2	United-States	Black	Fe...	0	0	20	0	302200	2.22...	
3	United-States	White	Fe...	0	0	30	0	122295	2.30...	
4	United-States	White	Fe...	0	0	25	0	544110	2.77...	
5	United-States	White	Fe...	0	0	40	0	284619	2.85...	
6	United-States	White	Fe...	0	0	40	0	128398	2.85...	
7	United-States	White	Male	2174	0	40	0	77555	3.07...	
8	Cuba	Black	Fe...	0	0	40	0	338437	3.07...	
9	United-States	White	Male	5178	0	40	1	159491	3.07...	
10	India	Asi...	Male	0	0	40	1	141327	3.07...	
11	United-States	White	Male	0	0	40	1	216907	3.07...	
12	United-States	White	Male	0	0	40	0	88559	3.07...	
13	United-States	Black	Male	0	0	40	1	337952	3.07...	
14	United-States	White	Male	0	1408	40	0	386985	3.07...	
15	Jamaica	Black	Fe...	0	0	16	0	160236	3.2	
16	United-States	White	Fe...	0	0	45	1	292218	3.21...	
17	United-States	White	Male	0	0	40	0	265525	3.33...	
18	United-States	White	Fe...	14084	0	50	1	45812	3.57...	
19	?	Asi...	Male	0	0	40	1	121812	3.63...	
20	United-States	White	Male	0	0	40	0	101644	3.63...	
21	United-States	White	Male	0	0	60	1	193564	3.75	
22	?	White	Male	0	0	38	1	84185	3.8	
23	United-States	White	Male	0	0	50	0	173011	3.84...	
24	United-States	White	Male	0	0	35	0	176781	3.88...	
25	United-States	White	Male	0	0	35	0	290005	3.88...	
26	United-States	White	Male	0	0	43	0	271495	3.90...	
27	United-States	White	Male	0	0	40	0	59981	4	
28	United-States	White	Male	0	0	40	0	197221	4	
...

6.13.2 函数

```
def transform(inputTableName, outputTableName, outputColNames, outputColExpressions,
             inputPartitions = None, outputPartition = None, outputColTypes = None):
```

参数

- `inputTableName`: 输入普通表名
- `outputTableName`: 输出表的表名
- `outputColNames`: 输出表各列的列名
- `outputColExpressions`: 产生输出表各列所用的表达式
- `inputPartitions`: (可选)输入数据所用的分区列表; 默认的None表示整个`inputTableName`表
- `outputPartition`: (可选)输出数据到分区; 默认的None表示直接输出到表
- `outputColTypes`: (可选)输出表各列的列类型; 默认为None, 会自动推测; 列表中可以填写的类型为: `boolean|bigint|double|string|datetime`;

示例:

```
DataProc.transform('bank_full', 'str_label_bank_full', ['str_label'],
['case when label==1 then "yes" else "no" end'])
```

6.14 行列转换(行转列)

行列转换(行转列)可以根据数据表中某一列的distinct value值生成新的列, 构造用户所需要的特征

6.14.1 函数

```
def pivot(inputTableName, outputTableName, pivotColumn, valueColumn,
signalColumn , pivotValues = None, newColumnName = None, partitions = None ):
```

参数

- inputTableName: 输入普通表名
- outputTableName: 输出表的表名
- pivotColumn: 进行行列转换的转换字段, 会根据此字段的distinct 值生成新的列
- valueColumn: 进行行列转换的值字段, 此列值会输出至结果表对应的新生列下
- signalColumn: 进行行列转换的标识字段
- pivotValues: (可选)可指定 pivotColumn distinct 值中的一部分, 只针对这部分值生成新的列
- newColumnName: (可选)当指定 pivotValues 时, 可制定生成列名
- partitions: (可选)输入数据所用的分区列表; 默认的None 表示整个 inputTableName 表

示例:

```
sql("create table testpivot as select race, sex, workclass, avg(age) as avg_age from
adult group by race, sex, workclass")
DataProc.pivot('testpivot', 'adult_test_pivot', 'sex',
               'avg_age', ['race', 'workclass'])
DataProc.pivot('testpivot', 'adult_test_pivot1', 'workclass',
               'avg_age', ['race', 'sex'], ['State-gov', 'Self-emp-not-inc', 'Private'],
               ['newcol1', 'newcol2', 'newcol3'])
```

6.14.2 界面

使用数据表:pivot_defect

表格 数据处理 全表统计 统计对比 分析 模型 工具 画图				
	grade	name	course	score
0	1	张三	english	90
1	1	张三	math	80
2	1	李四	english	100
3	1	李四	math	90
4	1	张三	physics	
5	1	李四	physics	90

1, 选择数据表, 点击数据处理→行列转换(行转列)



选择转换列, 转换列即为 pivotColumn, 进行行列转换的转换字段, 会根据此字段的 distinct 值生成新的列. 选择值列, 值列即为 valueColumn, 进行行列转换的值字段, 此列值会输出至结果表对应的新生列下. 选择标识列, 标识列即为 signalColumn, 进行行列转换的标识字段. 在高级设置中还可设置旋转值集合, 重命名新增列, 即 pivotValues 与 newColumnNames



2, 运行结果: 行列转换会把输出结果表展示出来, 如下图所示:

表格 数据处理 全表统计 统计对比 分析 模型 工具 画图					
	name	grade	pivot_english	pivot_math	pivot_physics
0	张三	1	90	80	
1	李四	1	100	90	90

6.15 行列转换(列转行)

行列转换(列转行)可以根据数据表中某几列的值生成新的行, 与行列转换(行转列)互为逆操作.

6.15.1 函数

```
def unpivot(inputTableName, outputTableName, unpivotColumnName, signalColumn,
            mergeColumnName, valueColumnName, unpivotValueMapping = None,
            partitions = None):
```

参数

- inputTableName: 输入普通表名
- outputTableName: 输出表的表名

- unpivotColumn: 进行行列转换的合并字段, 会把这些列合成两列, 每一行数据都会生成多行数据, 除 signalColumn 外, 每行数据都会有额外两列: 其中一列是 unpivotColumn 中某列列名, 另一列是该列在这行数据中对应的值
- signalColumn: 进行行列转换的标识字段
- mergeColumn: unpivotColumn 合并后, 保存原始列名的那一列
- valueColumn: unpivotColumn 合并后, 根据 unpivotColumn 中的值生成的新列
- unpivotValueMapping: (可选) 可指定新的值替换 mergeColumn 中的原始unpivotColumn列名, 大小需与unpivotColumn 相同
- partitions: (可选) 输入数据所用的分区列表; 默认的None 表示整个 inputTableName 表

示例:

```
DataProc.unpivot('adult', 'adult_test_unpivot', ['sex', 'race', 'workclass'],
['occupation', 'relationship'], 'unpivotmergeColumn', 'unpivotvalueColumn',
['new_sex', 'new_race', 'new_workclass'])
```

6.15.2 界面

界面运行示例: 使用数据表: testpivotout(pivot 操作生成的表)

	表格 数据处理 全表统计 统计对比 分析 模型 工具 画图				
	name	grade	pivot_english	pivot_math	pivot_physics
0	张三	1	90	80	
1	李四	1	100	90	90

1, 选择数据表, 点击数据处理→行列转换(列转行)



选择标识列, 标识列即为 signalColumn, 进行行列转换的标识字段. 选择合并列, 转换列即为 unpivotColumn, 进行行列转换的合并字段, 会把这些列合成两列, 每一行数据都会生成多行数据, 除 signalColumn 外, 每行数据都会有额外两列: 其中一列是 unpivotColumn 中某列列名, 另一列是该列在这行数据中对应的值选择值列, 值列即为 valueColumn, 进行行列转换的值字段, 此列值会输出至结果表对应的新生列下.

2, 运行结果: 行列转换会把输出结果表展示出来, 如下图所示:

表格 数据处理 全表统计 统计对比 分析 模型 工具 画图				
	name	grade	course	score
0	张三	1	pivot_english	90
1	张三	1	pivot_math	80
2	李四	1	pivot_english	100
3	李四	1	pivot_math	90
4	李四	1	pivot_physics	90

6.16 缺失值填充

通过给定一个缺失值的配置列表, 来实现将输入表的缺失值, 用指定的值来填充。

6.16.1 界面

双击数据表adult, 菜单栏选择 数据处理->缺失值填充, 如:

adult - 数据表浏览

表格	数据处理	全表描述	统计	分析	模型	工具	画图
0	过滤						
1	采样						
2	拆分						
3	动态分区						
4	排序						
5	前N条记录						
6	唯一结果集						
7	缺失值填充						
8	归一化/标准化						
9	分箱						
10	变量转换						
11	行列转换(行转列)						
12	行列转换(列转行)						
13							
14							
15	34	Private	fnlwgt	education	education...	education...	education...
16	25	Self-...	245487	7th-8th	4	HS-grad	9
			176756				

单击缺失值填充, 如:



单击添加按钮:



其中：

- 选列快捷键，可以快速选择数值列(double|long)或字符串列
- 选择数值列，支持多选
- 选择缺失值，缺失值支持四种类型， 分别为空值，空字符串，空值和空字符串，自定义。
- 选择替换值，替换值对于字符串支持自定义，数值型(double和long) 支持最大值和最小值替换，double类型还支持均值替换

填写完参数后，单击“确定”按钮，新增加的记录如下：



选中刚才添加的记录，单击”编辑”按钮显示如：



替换值更换为”Other-service”，单击”确定按钮”，如：



选中记录，单击删除按钮，即可删除不需要的记录。填写输出表名，单击确定：



运行成功，自动弹出结果表：

adult_missingfilling_out - 数据表浏览

	age	workclass	fnlwgt	education	education...	marital_s...	occupation	relationship	race	sex
0	39	State...	77516	Bache...	13	Never...	Adm-c...	Not-i...	White	Male
1	50	Self-...	83311	Bache...	13	Marri...	Exec...	Husband	White	Male
2	38	Private	215646	HS-grad	9	Divorced	Handl...	Not-i...	White	Male
3	53	Private	234721	11th	7	Marri...	Handl...	Husband	Black	Male
4	28	Private	338409	Bache...	13	Marri...	Prof...	Wife	Black	Fema
5	37	Private	284582	Masters	14	Marri...	Exec...	Wife	White	Fema
6	49	Private	160187	9th	5	Marri...	Other...	Not-i...	Black	Fema
7	52	Self-...	209642	HS-grad	9	Marri...	Exec...	Husband	White	Male
8	31	Private	45781	Masters	14	Never...	Prof...	Not-i...	White	Fema
9	42	Private	159449	Bache...	13	Marri...	Exec...	Husband	White	Male
10	37	Private	280464	Some-...	10	Marri...	Exec...	Husband	Black	Male
11	30	State...	141297	Bache...	13	Marri...	Prof...	Husband	Asian...	Male
12	23	Private	122272	Bache...	13	Never...	Adm-c...	Own-c...	White	Fema
13	32	Private	205019	Assoc...	12	Never...	Sales	Not-i...	Black	Male
14	40	Private	121772	Assoc...	11	Marri...	Craft...	Husband	Asian...	Male
15	34	Private	245487	7th-8th	4	Marri...	Trans...	Husband	Amer-...	Male
16	25	Self-...	176756	HS-grad	9	Never...	Farmi...	Own-c...	White	Male
17	32	Private	186824	HS-grad	9	Never...	Machi...	Unmar...	White	Male
18	38	Private	28887	11th	7	Marri...	Sales	Husband	White	Male
19	43	Self-...	292175	Masters	14	Divorced	Exec...	Unmar...	White	Fema
20	40	Private	193524	Doctor...	16	Marri...	Prof...	Husband	White	Male
21	54	Private	302146	HS-grad	9	Separ...	Other...	Unmar...	Black	Fema
22	35	Feder...	76845	9th	5	Marri...	Farmi...	Husband	Black	Male
23	43	Private	117037	11th	7	Marri...	Trans...	Husband	White	Male
24	59	Private	109015	HS-grad	9	Divorced	Tech...	Unmar...	White	Fema
25	56	Local...	216851	Bache...	13	Marri...	Tech...	Husband	White	Male
26	19	Private	168294	HS-grad	9	Never...	Craft...	Own-c...	White	Male
27	54	?	180211	Some-...	10	Marri...	?	Husband	Asian...	Male
28	30	Business	267080	HS-grad	0	Divorced	Exec...	Marri...	White	Male

Whole Table ▾ 位置 数据大小: -- 行, 当前显示前 50 行 (最多10000行) GO!

6.16.2 函数

```
def fillMissingValues(inputTableName, outputTableName, configs, \
    inputPartitions = None, outputPartition = None):
```

参数:

- `inputTableName`: 输入数据的表名
- `outputTableName`: 输出表的表名
- `configs`: 缺失值填充的配置, 格式如: `[('col1', 'null', '3.14'), ('col3', 'empty', 'hello'), ('col4', 'null-empty', 'xlab'), ('col8', 'user-defined', '?', 'world')]`
- `inputPartitions`: (可选) 输入数据所用的分区列表; 默认为None, 表示整个`inputTableName`表; 如: `['ds=20140320', 'ds=20140321']`
- `outputPartition`: (可选) 输出数据到分区; 默认为None, 表示直接输出到表; 如: `'ds=20140322'`

示例:

```
## 将表 table_a 的 feature1 列的 NULL, 用 3.14 填充
configs = [('feature1', 'null', '3.14')]
DataProc.fillMissingValues('table_a', 'table_b', configs)
```

例2:

```
## 将表 table_a 的 feature3 列的 空字符串, 用 'hello' 填充
configs = [('feature3', 'empty', 'hello')]
DataProc.fillMissingValues('table_a', 'table_b', configs)
```

例3:

```
## 将表 table_a 的 feature4 列的 NULL或者空字符串, 用字符串 'xlab' 填充
configs = [('feature4', 'null-empty', '3.14')]
DataProc.fillMissingValues('table_a', 'table_b', configs)
```

例4:

```
## 将表 table_a 的 feature8 列的 '?', 用 'world' 填充
configs = [('feature8', 'user-defined', '?', 'world')]
DataProc.fillMissingValues('table_a', 'table_b', configs)
```

例5:

```
## 将表 table_a 的 feature9 列(datetime类型)的 NULL, 用 "2014-03-23 08:13:54" 填充,
## 注意这里的日期类型的字符串格式
configs = [('feature9', 'null', '2014-03-23 08:13:54')]
DataProc.fillMissingValues('table_a', 'table_b', configs)
```

6.17 归一化

6.17.1 界面

双击数据表adult，菜单栏选择 数据处理->归一化/标准化，如：

adult - 数据表浏览

	fnlwgt	education	education...	...
0	77516	Bache...	13	
1	83311	Bache...	13	
2	215646	HS-grad	9	
3	234721	11th	7	
4	338409	Bache...	13	
5	284582	Masters	14	
6	160187	9th	5	
7	209642	HS-grad	9	
8	45781	Masters	14	
9	159449	Bache...	13	
10	280464	Some-...	10	
11	141297	Bache...	13	
12	122272	Bache...	13	
13	205019	Assoc...	12	
14	121772	Assoc...	11	
15	34	PRIVATE	7th-8th	4
16	25	Self-emp...	HS-grad	0

单击归一化/标准化，打开界面：



其中：

- 选择partition， 默认整个表
- 选择列， 默认选择所有的数值列(double|long类型)
- 归一化的类型， 一个是归一化， 一个是标准化
- 填写输出表
- 选择是否标有归一化的原有列， 否输出：非归一化+归一化结果列，是：原表列+归一化结果列

填写完参数，单击”确定”：



运行完成后，自动弹出结果表：

adult_normalize_out - 数据表浏览

	age	workclass	fnlwgt	education	education...	marital_s...	occupation	relationship	race	sex
0	0.3013...	State...	0.0443...	Bache...	0.8	Never...	Adm-c...	Not-i...	White	Male
1	0.4520...	Self-...	0.0482...	Bache...	0.8	Marri...	Exec...	Husband	White	Male
2	0.2876...	Private	0.1381...	HS-grad	0.5333...	Divorced	Handl...	Not-i...	White	Male
3	0.4931...	Private	0.1510...	11th	0.4	Marri...	Handl...	Husband	Black	Male
4	0.1506...	Private	0.2214...	Bache...	0.8	Marri...	Prof...	Wife	Black	Fema
5	0.2739...	Private	0.1849...	Masters	0.8666...	Marri...	Exec...	Wife	White	Fema
6	0.4383...	Private	0.1004...	9th	0.2666...	Marri...	Other...	Not-i...	Black	Fema
7	0.4794...	Self-...	0.1340...	HS-grad	0.5333...	Marri...	Exec...	Husband	White	Male
8	0.1917...	Private	0.0227...	Masters	0.8666...	Never...	Prof...	Not-i...	White	Fema
9	0.3424...	Private	0.0999...	Bache...	0.8	Marri...	Exec...	Husband	White	Male
10	0.2739...	Private	0.1821...	Some-...	0.6	Marri...	Exec...	Husband	Black	Male
11	0.1780...	State...	0.0876...	Bache...	0.8	Marri...	Prof...	Husband	Asian...	Male
12	0.0821...	Private	0.0746...	Bache...	0.8	Never...	Adm-c...	Own-c...	White	Fema
13	0.2054...	Private	0.1308...	Assoc...	0.7333...	Never...	Sales	Not-i...	Black	Male
14	0.3150...	Private	0.0743...	Assoc...	0.6666...	Marri...	Craft...	Husband	Asian...	Male
15	0.2328...	Private	0.1583...	7th-8th	0.2	Marri...	Trans...	Husband	Amer...	Male
16	0.1095...	Self-...	0.1117...	HS-grad	0.5333...	Never...	Farmi...	Own-c...	White	Male
17	0.2054...	Private	0.1185...	HS-grad	0.5333...	Never...	Machi...	Unmar...	White	Male
18	0.2876...	Private	0.0112...	11th	0.4	Marri...	Sales	Husband	White	Male
19	0.3561...	Self-...	0.1900...	Masters	0.8666...	Divorced	Exec...	Unmar...	White	Fema
20	0.3150...	Private	0.1230...	Docto...	1	Marri...	Prof...	Husband	White	Male
21	0.5068...	Private	0.1968...	HS-grad	0.5333...	Separ...	Other...	Unmar...	Black	Fema
22	0.2465...	Feder...	0.0438...	9th	0.2666...	Marri...	Farmi...	Husband	Black	Male
23	0.3561...	Private	0.0711...	11th	0.4	Marri...	Trans...	Husband	White	Male
24	0.5753...	Private	0.0656...	HS-grad	0.5333...	Divorced	Tech...	Unmar...	White	Fema
25	0.5342...	Local...	0.1389...	Bache...	0.8	Marri...	Tech...	Husband	White	Male
26	0.0273...	Private	0.1059...	HS-grad	0.5333...	Never...	Craft...	Own-c...	White	Male
27	0.5068...	?	0.1140...	Some-...	0.6	Marri...	?	Husband	Asian...	Male
28	0.2012	Private	0.2410	HS-grad	0.5333	Divorced	Ex-...	Not-i...	Unmar...	Male

Whole Table ▾ 位置 数据大小: -- 行, 当前显示前 50 行 (最多10000行) GO!

6.17.2 函数

```
def normalize(inputTableName, outputTableName, \
    inputPartitions = None, selectedColNames = None, outputPartition = None):
```

参数:

- inputTableName: 输入数据的表名
- outputTableName: 输出表的表名
- inputPartitions: (可选) 输入数据所用的分区列表; 默认为None, 表示整个inputTableName表; 如: ['ds=20140320', 'ds=20140321']
- selectedColNames: (可选) 要归一化的列名列表; 默认为None, 将表中所有bigint列和double列归一化; 如: ['col1', 'col2']
- outputPartition: (可选) 输出数据到分区; 默认为None, 表示直接输出到表; 如: 'ds=20140322'

示例:

```
## 将表 bank_full 的所有bigint类型和double类型列, 都归一化
DataProc.normalize('bank_full', 'normalized_bank_full')
## 也可以选定列, 进行归一化
DataProc.normalize('bank_full', 'standardize_bank_full', selectedColNames = ['f0', 'f1', 'f2', 'f3'])
```

6.18 标准化

6.18.1 界面

参考 [归一化](#)

6.18.2 函数

```
standardize(inputTableName, outputTableName, \
            inputPartitions = None, selectedColNames = None, outputPartition = None):
```

参数:

- inputTableName: 输入数据的表名
- outputTableName: 输出表的表名
- inputPartitions: (可选)输入数据所用的分区列表; 默认为None, 表示整个inputTableName表; 如: ['ds=20140320', 'ds=20140321']
- selectedColNames: (可选)要标准化的列名列表; 默认为None, 将表中所有bigint列和double列标准化; 如: ['col1', 'col2']
- outputPartition: (可选)输出数据到分区; 默认为None, 表示直接输出到表; 如: 'ds=20140322'

示例:

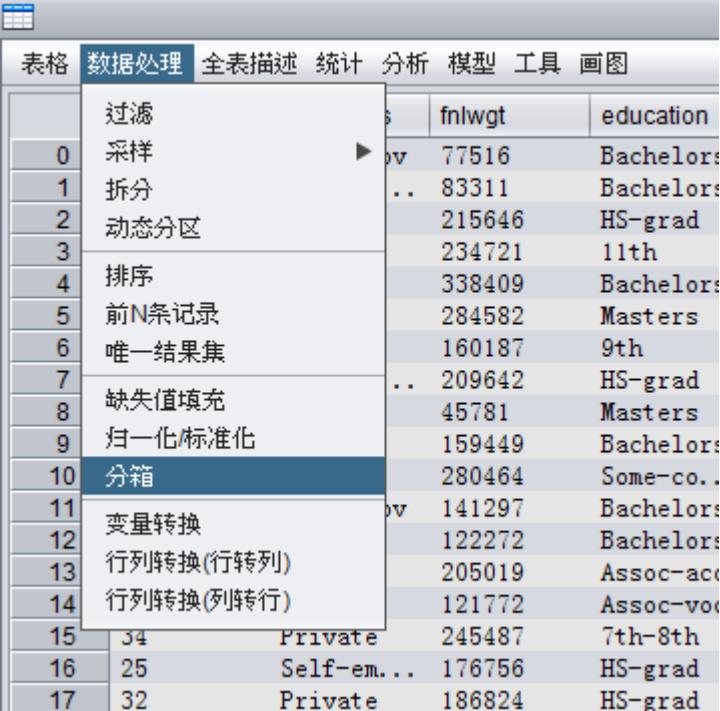
```
## 将表 bank_full 的所有bigint类型和double类型列, 都标准化
DataProc.standardize('bank_full', 'standardize_bank_full')
## 也可以选定列, 进行标准化
DataProc.standardize('bank_full', 'standardize_bank_full', selectedColNames = ['f0', 'f1', 'f2', 'f3'])
```

6.19 分箱

分箱有三种方式: 等距分箱, 等量分箱, 均值/标准差分箱

6.19.1 界面

双击数据表adult，菜单栏选择 数据处理->分箱，如：



The screenshot shows a software interface with a menu bar at the top. The 'Data Processing' option is highlighted in blue. A dropdown menu is open under 'Data Processing', showing various data manipulation options: 过滤 (Filter), 采样 (Sampling), 拆分 (Split), 动态分区 (Dynamic Partitioning), 排序 (Sort), 前N条记录 (Top N Records), 唯一结果集 (Unique Result Set), 缺失值填充 (Missing Value Filling), 归一化/标准化 (Normalization/Standardization), 分箱 (Binning), 变量转换 (Variable Transformation), 行列转换(行转列) (Transpose Row to Column), and 行列转换(列转行) (Transpose Column to Row). The 'Binning' option is selected and highlighted with a blue background.

		fnlwgt	education
0	ov	77516	Bachelors
1	..	83311	Bachelors
2	215646	HS-grad	
3	234721	11th	
4	338409	Bachelors	
5	284582	Masters	
6	160187	9th	
7	..	209642	HS-grad
8	45781	Masters	
9	159449	Bachelors	
10	280464	Some-co..	
11	ov	141297	Bachelors
12	122272	Bachelors	
13	205019	Assoc-acd	
14	121772	Assoc-voc	
15	34	Private	7th-8th
16	25	Self-em...	HS-grad
17	32	Private	HS-grad

单击分箱，如：



单击添加按钮：



其中：

- 选择数值列，支持多选，默认显示所有数值列
- 分箱方法，等距，等量，均值/标准差
- 等距分箱下，可以选择箱子步长或分箱个数
- 等量分箱下，可以选择记录数或分箱个数
- 均值/标准差，可以选择 ± 1 标准差， ± 2 标准差， ± 3 标准差，

填写完参数后，单击“确定”按钮，新增加的记录如下：



选中刚才添加的记录，单击”编辑”按钮显示如：



将箱子步长设置为21，单击“确定按钮”，如：



选中记录，单击删除按钮，即可删除不需要的记录。填写输出表明，单击确定：



运行成功，自动弹出结果表：

age_bining_out - 数据表浏览

6.19.2 函数

```
def binning(inputTableName, outputTableName, configs, \
           inputPartitions = None, outputPartition = None, isKeepOriginalCol = False):
```

参数:

- `inputTableName`: 输入数据的表名
- `outputTableName`: 输出表的表名
- `configs`: 分箱配置,
- 等距分箱格式: (列名, `fixed-width`, `bin-num`(分箱个数)/`bin-width`(箱子宽度), 值), 如: `[('col1', 'fixed-width', 'bin-num', '10'), ('col2', 'fixed-width', 'bin-width', '10.50')]`
- 等量分箱格式: (列名, `equal-count`, `bin-num`(分箱个数)/`record-count`(记录数), 值), 如: `[('col1', 'equal-count', 'bin-num', '10'), ('col2', 'equal-count', 'record-count', '10000')]`

- 均值/标准差分箱格式: (列名, mean/std, sd1(+/- 1标准差)/ sd2(+/- 2标准差)/ sd3(+/- 3标准差)), 如: [(‘col1’, ‘mean/std’ , ‘std1’), (‘col2’, ‘mean/std’ , ‘std2’), (‘col3’, ‘mean/std’ , ‘std2’)]
- inputPartitions: (可选)输入数据所用的分区列表; 默认为None, 表示整个inputTableName表; 如: [‘ds=20140320’ , ‘ds=20140321’]
- outputPartition: (可选)输出数据到分区; 默认为None, 表示直接输出到表; 如: ‘ds=20140322’
- isKeepOriginalCol: (可选)是否保持分箱列的原有列, 默认False

示例:

```
## 将表 adult 的 age列等距分箱且箱子数3个, capital_gain列等量分箱且记录数为1000, hours_per_week列均值/标准差分箱且使用(+/- config)
configs=[('age','fixed-width','bin-num','3'), ('capital_gain','equal-count','record-count','1000'),
         ('hours_per_week','mean/std', 'std1')]
DataProc.binning('adult', 'adult_binning_out', configs)
```

6.20 数据生成

可以生成均匀分布, 高斯分布, 泊松分布, 集合和表达式类型的数据。其中集合指的是给出确定的一个数据集和对应权重的权重, 生成这个集合内的数据, 数据的个数服从给出的权重。表达式类型指的是使用xlib表达式, 其中加入了#row和#col这两个常量。#row表示数据的当前行号, #col表示数据的当前列号。列名不指定的情况下服从col0, col1…的规则。可以生成表, 矩阵。

6.20.1 函数

```
def randomTable(outputTableName, rows, cols, outputPartition = None, colNames = None, colConfs = None) :
```

参数:

- outputTableName: 输出表名。
- rows: 生成表的行数。
- cols: 生成表的列数。
- outputPartition: (可选)输出分区名。默认为无分区。
- colNames: (可选)输出表的列名。默认为[‘col0’ , ‘col1’ …]。
- colConfs: (可选)输出表的列分布。默认为[0.0, 1.0)均匀分布。

本参数使用key-value来表示列分布, 支持的参数有:

```
{'method': 'uniform', 'param': '0.0, 2.0'} 表示[0.0, 2.0)的均匀分布。
{'method': 'uniform_open', 'param': '0.0, 2.0'} 表示[0.0, 2.0]的均匀分布。
```

```
{'method': 'gauss', 'param': '0.0, 2.0'} 表示均值为0.0, 方差为2.0的高斯分布。
{'method': 'poisson', 'param': '2.0'} 表示 $\lambda$ 为2.0的泊松分布。
{'method': 'weight_set', 'param': '0.0, 2.0, 1.0, 2.0'} 表示0.0和1.0的集合, 权重分别为2.0和2.0。
{'method': 'function', 'param': '#col0 + 1'} 表示本列的值为col0列加1。
```

如果要指定空值的比例, 则用如下形式:

```
{'method': 'uniform', 'param': '0.0, 2.0', 'null_per': '0.1'}
```

如果要指定列的类型, 则用如下形式:

```
{'method': 'uniform', 'param': '0.0, 2.0', 'null_per': '0.1', 'type': 'string'}
```

示例:

```
DataGenerator.randomTable('output_table_name', 1000, 6, outputPartition = 'p=1',
    colNames = ['col0', 'col1', 'col2', 'col3', 'col4', 'col5'],
    colConfs = [ {'method': 'uniform', 'param': '0.0, 2.0'},
        {'method': 'uniform_open', 'param': '0.0, 2.0'},
        {'method': 'gauss', 'param': '0.0, 2.0'},
        {'method': 'poisson', 'param': '2.0'},
        {'method': 'weight_set', 'param': '0.0, 2.0, 1.0, 2.0'},
        {'method': 'function', 'param': '#col0 + 1'}])
```

```
def randomDense(outputMatrixName, rows, cols, colConf = None) :
```

参数:

- outputMatrixName: 输出稠密矩阵名。
- rows: 生成稠密矩阵的行数。
- cols: 生成稠密矩阵的列数。
- colConf: (可选)输出矩阵的分布。默认为[0.0, 1.0)均匀分布。

本参数使用key-value来表示列分布, 支持的参数有:

```
{'method': 'uniform', 'param': '0.0, 2.0'} 表示[0.0, 2.0)的均匀分布。
{'method': 'uniform_open', 'param': '0.0, 2.0'} 表示[0.0, 2.0]的均匀分布。
{'method': 'gauss', 'param': '0.0, 2.0'} 表示均值为0.0, 方差为2.0的高斯分布。
{'method': 'poisson', 'param': '2.0'} 表示 $\lambda$ 为2.0的泊松分布。
{'method': 'weight_set', 'param': '0.0, 2.0, 1.0, 2.0'} 表示0.0和1.0的集合, 权重分别为2.0和2.0。
```

示例:

```
DataGenerator.randomDense('output_matrix_name', 1000, 6,
    colConf = {'method': 'uniform', 'param': '0.0, 2.0'})
```

```
def randomSparse(outputMatrixName, rows, cols, nonZero, colConf = None) :
```

参数:

- outputMatrixName: 输出 Xlib稀疏矩阵表 名。
- rows: 生成稀疏矩阵的行数。
- cols: 生成稀疏矩阵的列数。
- nonZero: 生成稀疏矩阵的非零元个数。
- colConfs: (可选)输出矩阵的分布。默认为[0.0, 1.0)均匀分布。本参数使用key-value来表示列分布, 支持的参数有:

```
{'method': 'uniform', 'param': '0.0, 2.0'} 表示[0.0, 2.0)的均匀分布。  

{'method': 'uniform_open', 'param': '0.0, 2.0'} 表示[0.0, 2.0]的均匀分布。  

{'method': 'gauss', 'param': '0.0, 2.0'} 表示均值为0.0, 方差为2.0的高斯分布。  

{'method': 'poisson', 'param': '2.0'} 表示 $\lambda$ 为2.0的泊松分布。  

{'method': 'weight_set', 'param': '0.0, 2.0, 1.0, 2.0'} 表示0.0和1.0的集合, 权重分别为2.0和2.0。
```

示例:

```
DataGenerator.randomSparse('output_matrix_name', 1000, 6, 1000,  

    colConf = {'method': 'uniform', 'param': '0.0, 2.0'})
```

6.20.2 界面

在数据浏览界面, 点击工具->数据生成, 选择生成数据的类型。



以生成表为例:

打开生成表界面



可以选择配置列信息或者不配置，如果不配置的情况下，默认是(0, 1]区间的均匀分布。填入行数，列数和输出表名，点击开始即可生成对应的表。如果有配置列的需求的话，点击增加按钮：



可以配置对应的列信息。点击确定即可完成编辑。

第 7 章

格式转换

在XLab中，格式转换特指对这些对象：表(Table)、XLib稀疏矩阵表、XLib稠密矩阵表、三元组(Triple)表、索引三元组表、KV表、索引KV表的格式进行相互转换。

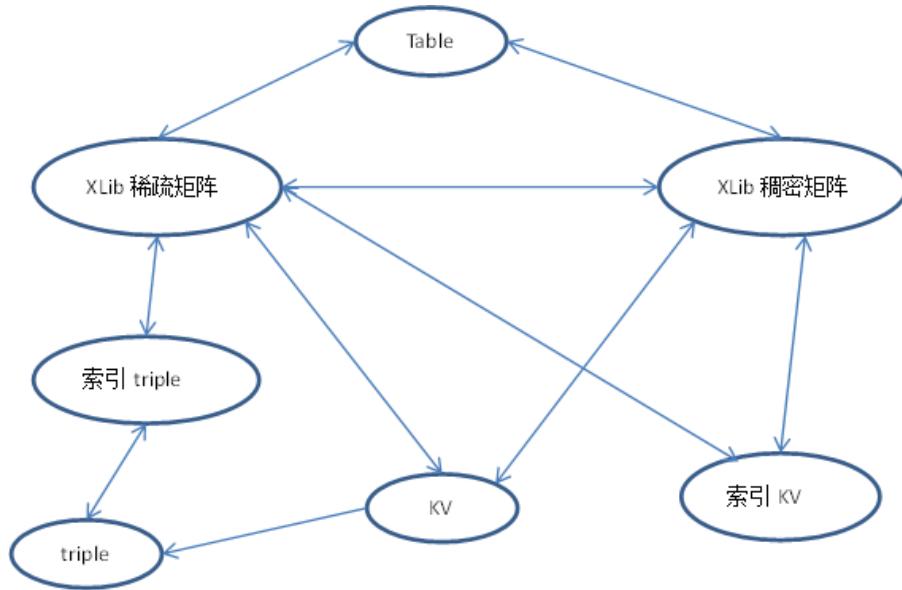
格式转换涉及：普通表与XLib稀疏矩阵表、XLib稠密矩阵表间的相互转换；三元组(Triple)、索引三元组与XLib稀疏矩阵表、XLib稠密矩阵表间的相互转换；KV表、索引KV表与XLib稀疏矩阵表、XLib稠密矩阵表间的相互转换；KV表转三元组；XLib稀疏矩阵表与XLib稠密矩阵表间的相互转换。这些对象间的关系见转换图。

XLab之所以提供格式转换函数的原因主要有两个：

- 对于用户的输入数据已在ODPS中的情况，XLib提供的算法和功能一般支持的操作对象是ODPS表（列1024），但是有些算法比如GBRT，逻辑回归、SVM等算法也支持大数据（列大于1024）的训练预测，这种算法的输入就不能是表，而需用XLib自定义的数据格式XLib稀疏矩阵表和XLib稠密矩阵表。
- 对于用户的输入数据不在ODPS中，需要从外部导入的情况，如果输入数据数据的列小于等于1024，可直接利用ODPS的导数据工具Tunnel将外部数据导成ODPS表；如果输入数据列大于1024，那么就需要将输入数据生成KV格式，然后将KV格式的数据通过ODPS Tunnel导成KV表，然后将KV表转为XLib稀疏矩阵表或XLib稠密矩阵表即可完成大数据的统计和挖掘。

注解： 补全以及函数提示的快捷键：Ctrl+p。

7.1 转换图



7.2 普通表转XLib稠密矩阵表

```
def tableToDenseMatrix(input, output, selectedColIndex = None):
```

参数:

- input: 输入普通表
- output: 输出表, 表需不存在
- selectedColIndex: (可选)输入表选列, 默认选所有列

示例:

```
DataConvert.tableToDenseMatrix("appendinputleft",
    "appendinputleft_dense", selectedColIndex=[0, 1, 2])
```

7.3 普通表转XLib稀疏矩阵表

```
def tableToSparseMatrix(input, output, selectedColIndex = None):
```

参数:

- input: 输入普通表
- output: 输出表, 表需不存在
- selectedColIndex: (可选)输入表选列, 默认选所有列

示例:

```
DataConvert.tableToSparseMatrix("appendinputleft",
    "appendinputleft_out", selectedColIndex=[0, 1, 2])
```

7.4 XLib稠密矩阵表转普通表

```
denseMatrixToTable(input, output, selectedColIndex = None):
```

参数:

- input: XLib稠密矩阵表
- output: 输出表, 表需不存在
- selectedColIndex: (可选)输入表选列, 默认选所有列

示例:

```
DataConvert.denseMatrixToTable("appendinputleft_dense",
    "appendinputleft_dense_out", selectedColIndex=[0, 1, 2])
```

7.5 XLib稀疏矩阵表转普通表

```
def sparseMatrixToTable(input, output, selectedColIndex = None):
```

参数:

- input: 输入XLib稀疏矩阵
- output: 输出表, 表需不存在
- selectedColIndex: (可选)输入表选列, 默认选所有列

示例:

```
DataConvert.sparseMatrixToTable("appendinputleft_sparse",
    "appendinputleft_sparse_out", selectedColIndex=[0, 1, 2])
```

7.6 XLib稀疏矩阵表转XLib稠密矩阵表

```
def sparseMatrixToDenseMatrix(input, output, selectedColIndex = None):
```

参数:

- input: 输入XLib稀疏矩阵
- output: 输出表, 表需不存在
- selectedColIndex: (可选)输入表选列, 默认选所有列

示例:

```
DataConvert.sparseMatrixToDenseMatrix("appendinputleft_sparse",
"appendinputleft_sparse_out", selectedColIndex=[0, 1, 2])
```

7.7 XLib稠密矩阵表转XLib稀疏矩阵表

```
def denseMatrixToSparseMatrix(input, output, selectedColIndex = None):
```

参数:

- input: 输入XLib稠密矩阵表
- output: 输出表, 表需不存在
- selectedColIndex: (可选)输入表选列, 默认选所有列

示例:

```
DataConvert.denseMatrixToSparseMatrix("appendinputleft_dense",
"appendinputleft_dense_out", selectedColIndex=[0, 1, 2])
```

7.8 索引三元组表转稀疏矩阵表

```
def indexedTripleToSparseMatrix(indexedTriple, sparseMatrixName,
rName = None, cName=None, vName=None, partitions = None, nrow=None, ncol=None):
```

参数:

- indexedTriple: 输入索引三元组
- sparseMatrixName: 输出矩阵表名
- rName: (可选)表示输入indexedTriple 行的列名
- cName: (可选)表示输入indexedTriple 列的列名

- vName: (可选) 表示输入indexedTriple 值的列名, 如果为二元组则vName为None
- partitions: (可选) 表示索引三元组的输入partition, 默认表示全选
- nrow: (可选) 输出稀疏矩阵行数, 默认表示由indexTriple数据自动生成行数, 否则nrow必须>=由indexTriple数据自动生成行数
- ncol: (可选) 输出稀疏矩阵列数, 默认表示由indexTriple数据自动生成列数, 否则ncol必须>=由indexTriple数据自动生成列数

示例:

```
DataConvert.indexedTripleToSparseMatrix("indexedTriple_1", "sparse_1");
DataConvert.indexedTripleToSparseMatrix("indexedTriple_1", "sparse_1", nrow=10000, ncol=100);
```

7.9 XLib稀疏矩阵表转索引三元组表

```
def sparseMatrixToIndexedTriple(sparseMatrixName, indexedTriple):
```

参数:

- sparseMatrixName: 输入稀疏矩阵
- indexedTriple: 输出三元组, 类型【long, long, double】

示例:

```
DataConvert.sparseMatrixToIndexedTriple("sparse_table", "sparse_table_out");
```

7.10 索引三元组表+行map或列map表转三元组表

```
def indexedTripleToTriple(indexedTriple, rName, cName, vName, triple,
    rowMap=None, rKey=None, rIndex=None, colMap=None, cKey=None, cIndex=None):
```

参数:

- indexedTriple 输入索引三元组
- rName: 表示输入indexedTriple 行的列名
- cName: 表示输入indexedTriple 列的列名
- vName: 表示输入indexedTriple 值的列名, 如果为二元组则vName为None
- triple: 输出三元组
- rowMap: (可选) 输入行map表 , 可选, 默认为None, 表示三元组行索引值, 无须转换
- rKey: (可选) rowMap表中与triple对应的行名

- rIndex: (可选)rowMap表中行索引列
- colMap: (可选)输入列map表, 可选, 默认为None, 表示三元组列列为索引值, 无须转换
- cKey: (可选)colMap表中与triple对应的列名
- cIndex: (可选)colMap表中列索引

示例:

```
DataConvert.indexedTripleToTriple("index_triple_1", "col0", "col1", "col2",
    "triple_out1", "rowMap", "rName", "rIndex", "colMap", "cName", "cIndex");
DataConvert.indexedTripleToTriple("index_triple_2", "col0", "col1", "col2",
    "triple_out2", "rowMap", "rName", "rIndex");
DataConvert.indexedTripleToTriple("index_triple_3", "col0", "col1", "col2",
    "triple_out3", colMap="colMap", cKey="cName", cIndex="cIndex");
```

7.11 获取三元组表的行map表或列map表

```
def getTripleMaps(triple, rName=None, rowMap=None, cName=None, colMap=None):
```

参数:

- triple: 三元组
- rName: (可选)行名
- rowMap: (可选)输出行map表
- cName: (可选)列名
- colMap: (可选)输出列map表

示例:

```
DataConvert.getTripleMaps("triple_1", rName="col0", rowMap="rowmap_out",
    cName="col1", colMap="colmap_out")
DataConvert.getTripleMaps("triple_1", rName="col0", rowMap="rowmap_out")
DataConvert.getTripleMaps("triple_1", cName="col1", colMap="colmap_out")
```

7.12 三元组表转索引三元组表+行map或列map表

```
def tripleToIndexedTriple(triple, indexedTriple, rowMap=None,
    colMap=None, rName = None, cName = None, vName = None):
```

参数:

- triple: 三元组(类型为XXD 或 XXL 或 XX, X表示任意类型, D表示Double, L表示bigint)

- indexedTriple: 输出索引三元组
- rowMap: (可选)输出行map表, 默认为None, 表示三元组行列为索引值, 无须转换, 即triple类型为LXD | LXL | LX
- colMap: (可选)输出列map表, 默认为None, 表示三元组列列为索引值, 无须转换, 即triple类型为XLD | XLL | XL
- rName: (可选)表示三元组行的列名
- cName: (可选)表示三元组列的列名
- vName: (可选)表示三元组值的列名
- rName, cName, vName默认为None, 表示第0列为rName, 第一列cName, 如果存在第二列则为vName
- 如果triple为二元组, 则vName为None

示例:

```
DataConvert.tripleToIndexedTriple("triple_1", "index_triple_out",
    "rowmap_out", "colmap_out")
DataConvert.tripleToIndexedTriple("triple_2", "index_triple_out",
    colMap="colmap_out")
DataConvert.tripleToIndexedTriple("triple_3", "index_triple_out", "rowmap_out")
DataConvert.tripleToIndexedTriple("triple_4", "index_triple_out",
    "rowmap_out", "colmap_out", "col0", "col1")
```

7.13 三元组表+行map或列map表转索引三元组表

```
def tripleWithMapToIndexedTriple(triple, rName, cName, vName, indexedTriple,
    rowMap=None, rKey=None, rIndex=None, colMap=None, cKey=None, cIndex=None):
```

参数:

- triple: 三元组(类型为XXD+rowMap+colMap 或 XLD + rowMap 或者 LXD+colMap, X表示任意类型, D表示Double, L表示bigint)
- rName: 表示三元组triple行的列名
- cName: 表示三元组triple列的列名
- vName: 表示三元组triple值的列名
- indexedTriple: 输出索引三元组
- rowMap: (可选)输入行map表, 默认为None, 表示三元组行列为索引值, 无须转换, 即triple类型为LXD | LXL | LX
- rKey: (可选)rowMap表中与triple对应的行名
- rIndex: (可选)rowMap表中行索引列

- colMap: (可选)输入列map表, 默认为None, 表示三元组列列为索引值, 无须转换, 即triple类型为XLD | XLL | XL
- cKey: (可选)colMap表中与triple对应的列名, cKey不能含有减号字符 “-”
- cIndex: (可选)colMap表中列索引
- rName, cName, vName默认为None, 表示第0列为rName, 第一列cName, 如果存在第二列则为vName
- 如果triple为二元组, 则vName为None

示例:

```
DataConvert.tripleWithMapToIndexedTriple("triple_1", "col0", "col1", "col2",
    "indexedTriple_out", "rowMap", "rName", "rIndex", "colMap", "cName", "cIndex");

DataConvert.tripleWithMapToIndexedTriple("triple_1", "col0", "col1", "col2",
    "indexedTriple_out", "rowMap", "rName", "rIndex");

DataConvert.tripleWithMapToIndexedTriple("triple_1", "col0", "col1", "col2",
    "indexedTriple_out", colMap="colMap", cKey="cName", cIndex="cIndex");
```

7.14 KV表转稠密矩阵表

```
def KVToDenseMatrix(kvMatTableName, mapTableName, matTableName,
    kvMatColsName=None, kvMatPartitions = None, colDelimiter=None, valDelimiter=None, isAscii=False):
```

参数:

- kvMatTableName: 输入的kv表。
- mapTableName: map表。
- matTableName: 输出的稠密矩阵。
- kvMatColsName: (可选)kv表中被选中的列名。默认选中所有列。
- kvMatPartitions: (可选) kv表被选中的partitions。默认选中所有partitions。
- colDelimiter: (可选)列分割符。默认为”,”。
- valDelimiter: (可选)key-value分隔符。默认为”：“。
- isAscii: (可选)分隔符是否为ascii码。默认为False。如果为True, 则colDelimiter和valDelimiter为ascii码的字符串, 如colDelimiter="44"。

示例:

```
DataConvert.KVToDenseMatrix("kv_table_name", "map_table_name", "out_matrix_name",
    kvMatColsName = ['col1'], kvMatPartitions = None, colDelimiter = ",",
    valDelimiter = ":" , isAscii = False)
```

7.15 KV表转稀疏矩阵表

```
def KVToSparseMatrix(kvMatTableName, mapTableName, matTableName,
    kvMatColsName=None, kvMatPartitions = None, colDelimiter=None, valDelimiter=None, isAscii=False):
```

参数:

- kvMatTableName: 输入的kv表。
- mapTableName: map表。
- matTableName: 输出的稀疏矩阵。
- kvMatColsName: (可选)kv表中被选中的列名。默认选中所有列。
- kvMatPartitions: (可选) kv表被选中的partitions。默认选中所有partitions。
- colDelimiter: (可选)列分割符。默认为”,”。
- valDelimiter: (可选)key-value分隔符。默认为”：“。
- isAscii: (可选)分隔符是否为ascii码。默认为False。如果为True, 则colDelimiter和valDelimiter为ascii码的字符串, 如colDelimiter=”44”。

示例:

```
DataConvert.KVToSparseMatrix("kv_table_name", "map_table_name", "out_matrix_name",
    kvMatColsName = ['col1'], kvMatPartitions = None, colDelimiter = ",",
    valDelimiter = ":" , isAscii = False)
```

7.16 索引KV表转稠密矩阵表

```
def indexedKVToDenseMatrix(kvMatTableName, matTableName, kvMatColsName=None, kvMatPartitions = None,
    colDelimiter=None, valDelimiter=None, isAscii=False):
```

参数:

- kvMatTableName: indexed key-value表。
- matTableName: 输出的稠密矩阵。
- kvMatColsName: (可选) indexed key-value表被选中的列名。默认选中所有列。
- kvMatPartitions: (可选) indexed key-value表被选中的partitions。默认选中所有partitions。
- colDelimiter: (可选)列分隔符。默认为”,”。
- valDelimiter: (可选)key-value分隔符。默认为”：“。

- `isAscii`: (可选) 分隔符是否为ascii码。默认为False。如果为True, 则`colDelimiter`和`valDelimiter`为ascii码的字符串, 如`colDelimiter="44"`。

示例:

```
DataConvert.indexedKVToDenseMatrix("kv_table_name", "out_matrix_name",
    kvMatColsName = ['col1'], kvMatPartitions = None,
    colDelimiter = ",", valDelimiter = ":", isAscii = False)
```

7.17 索引KV表转稀疏矩阵表

```
def indexedKVToSparseMatrix(kvMatTableName, matTableName, kvMatColsName=None, kvMatPartitions = None,
    colDelimiter=None, valDelimiter=None, isAscii=False):
```

参数:

- `kvMatTableName`: indexed key-value表。
- `matTableName`: 输出的稀疏矩阵。
- `kvMatColsName`: (可选) indexed key-value表被选中的列名。默认选中所有列。
- `kvMatPartitions`: (可选) indexed key-value表被选中的partitions。默认选中所有partitions。
- `colDelimiter`: (可选) 列分隔符。默认为”,”。
- `valDelimiter`: (可选) key-value分隔符。默认为”：“。
- `isAscii`: (可选) 分隔符是否为ascii码。默认为False。如果为True, 则`colDelimiter`和`valDelimiter`为ascii码的字符串, 如`colDelimiter="44"`。

示例:

```
DataConvert.indexedKVToSparseMatrix("kv_table_name", "out_matrix_name",
    kvMatColsName = ['col1'], kvMatPartitions = None, colDelimiter = ",",
    valDelimiter = ":", isAscii = False)
```

7.18 矩阵表转索引KV表

```
def matrixToIndexedKV(matTableName, kvMatTableName, colDelimiter=None,
    valDelimiter=None, isAscii=False, kvTableColNum=None):
```

参数:

- `matTableName`: 矩阵表。
- `kvMatTableName`: indexed key-value表。

- colDelimiter: (可选)列分割符。列分割符。默认为”,”。
- valDelimiter: (可选)key-value分隔符。默认为”：“。
- isAscii: (可选)分隔符是否为ascii码。默认为False。如果为True, 则colDelimiter和valDelimiter为ascii码的字符串, 如colDelimiter=”44”。
- kvTableColNum: (可选)输出表的列数。如果不指定这个参数, 那么本方法计算能完整存储kv所需要的列数。如果指定, 则按照指定的列来存储kv。

示例:

```
DataConvert.matrixToIndexedKV("input_matrix_name", "out_kv_table_name",
    colDelimiter = ",", valDelimiter = ":", isAscii = False, kvTableColNum = 1)
```

7.19 矩阵表转KV表

```
def matrixToKV(matTableName, kvMatTableName, mapTableName,
    mapSelectedCols = None, colDelimiter=None, valDelimiter=None,
    isAscii=False, kvTableColNum=None):
```

参数:

- matTableName: 矩阵表。
- kvMatTableName: key-value表。
- mapTableName: 列号列名对应map表。
- mapSelectedCols: (可选)mapTableName中这些列来表示map关系。默认选中所有列。
- colDelimiter: (可选)列分割符。默认为”,”。
- valDelimiter: (可选)key-value分隔符。默认为”：“。
- isAscii: (可选)分隔符是否为ascii码。默认为False。如果为True, 则colDelimiter和valDelimiter为ascii码的字符串, 如colDelimiter=”44”。
- kvTableColNum: (可选)输出表的列数。如果不指定这个参数, 那么本方法计算能完整存储kv所需要的列数。如果指定, 则按照指定的列来存储kv。

示例:

```
DataConvert.matrixToKV("input_matrix_name", "out_kv_table_name", "map_table_name",
    mapSelectedCols = ['col1', 'col2'], colDelimiter = ",",
    valDelimiter = ":", isAscii = False, kvTableColNum = 1)
```

7.20 KV表转三元组表

```
def KVToTriple(kvTableName, colSeparator, vSeparator, outputTriple, selectedColsIndex=None, trim=False):
```

参数:

- kvTableName: 输入kv表
- selectedColsIndex: 输入变选列, 必须列为String类型
- colSeparator: kv对之间分隔符, 必须输入
- vSeparator: k, v之间分隔符, 如果为空则表示输出无value列
- trim: 是否忽略分隔符前后空格
- outputTriple: 输出三元组或者二元组表名, 必须输入

示例:

```
DataConvert.KVToTriple("input_kv_table_name", ",", ":" , "output_triple_name")
```

第 8 章

分类预测

分类预测是一种利用已知类别的样本训练分类模型，为未知类别的样本预测类别的有监督机器学习方法。XLab支持的分类预测模型包括：随机森林、逻辑回归（二分类回归、多分类回归和逐步回归）、线性支持向量机（Linear SVM）、朴素贝叶斯（Naive Bayes）、贝叶斯判别（Bayes）、费希尔判别（Fisher）、马氏距离判别（MDistance）。同时，XLab也提供分类评估功能，包括计算准确率、召回率、F值，对于二分类还支持画ROC。分类预测的执行方式有：函数和界面，详细使用说明见下文。

注解： 补全以及函数提示的快捷键：Ctrl+p。

8.1 随机森林

随机森林是一个包含多个决策树的分类器，并且其输出的类别是由个别树输出的类别的众数而定。Leo Breiman和Adele Cutler发展出推论出随机森林的算法。而“Random Forests”是他们的商标。这个术语是1995年由贝尔实验室的Tin Kam Ho所提出的随机决策森林(random decision forests)而来的。这个方法则是结合Breimans的“Bootstrap aggregating”想法和 Ho 的“random subspace method”以建造决策树的集合。

- 随机森林算法介绍： Random forest 。
- 随机森林算法参考： Leo Breiman (2001). Random Forests. Machine Learning. 45(1):5–32.

8.1.1 函数

随机森林模型包括以下几个函数: train, predict, isRandomForestModel, loadModel和exportModel。

这几个函数具体使用方法, 可用help命令查看, 示例:

```
help(Classification.RandomForest.train)
help(Classification.RandomForest.predict)
help(Classification.RandomForest.isRandomForestModel)
help(Classification.RandomForest.loadModel)
help(Classification.RandomForest.exportModel)
```

训练

```
def train(inputTableName, featureColNames, isFeatureContinuous,
          labelColName, modelTableName, treeNum,
          inputPartitions = None, weightColName = None, templateTableName = None,
          validateTableName = None, validatePartitions = None, algorithmTypes = None,
          randomColNum = -1, minNumObj = -1, minNumPer = -1.,
          maxTreeDeep = -1, maxRecordSize = -1) :
```

参数:

- inputTableName: 训练输入表的表名。
- featureColNames: 输入表中选择的用于训练的特征列名。
- isFeatureContinuous: 特征对应的类型, 表示对应特征是否为连续值。True为连续, False为离散。[True, False, False]表示三列特征中第一列为连续, 第二和第三列为离散。
- labelColName: 输入表中标签列的列名。
- modelTableName: 输出的模型名。
- treeNum: 森林中树的个数, 范围(0, 1000]
- inputPartitions: (可选)。输入表对应的输入分区, 选中全表则为None。
- weightColName: (可选)。输入表中权重列的列名, 无权重列则为None, weightColName>0。
- templateTableName: (可选)。模版表的表名, 无模版表则为None。
- validateTableName: (可选)。验证表的表名, 无验证表则为None。
- validatePartitions: (可选)。验证表对应的输入分区, 无分区则为None。
- algorithmTypes: (可选)。单棵树的算法在森林中的位置。如果有则长度为2, 例如: 在一个拥有5棵树的森林中, [2, 4]表示0, 1为id3算法, 2, 3为cart算法, 4为c4.5算法。如果输入为None, 则算法在森林中均分。

- randomColNum: (可选)。单颗树在生成时, 每次选择最优特征, 随机的特征个数。-1表示 \log_2 (特征的总个数)。
- minNumObj: (可选)。叶节点数据的最小个数。-1表示最小个数为2。
- minNumPer: (可选)。叶节点数据个数占父节点的最小比例。-1. 表示无这一限制。范围[0.0, 1.0]
- maxTreeDeep: (可选)。单颗树的最大深度。-1表示完全生长。范围[1, ∞)
- maxRecordSize: (可选)。森林中单颗树输入的随机数据的个数。范围为(0, 1000000)。-1表示100000。

返回:

- RandomForestModel类型, 表示随机森林的模型。

示例:

```
Classification.RandomForest.train('table_name', ['col1', 'col2'], [True, False],
    'label', "model_table_name", 10, inputPartitions = ['p=1'],
    weightColName = 'weight', templateTableName = 'template_table_name', validateTableName = 'vali_table_name',
    validatePartitions = ['p=1'], algorithmTypes = [3, 6], randomColNum = -1, minNumObj = -1, minNumPer = -1.,
    maxTreeDeep = 5, maxRecordSize = -1)
```

注意事项:

- 随机森林通过对bagging方法的改进, 在大数据集上构建一个单颗树不相关的森林。在很多问题上, 随机森林跟boosting方法类似, 有类似的训练过程。
- 本方法对于单颗树的增长, 可选的有id3, cart或c4.5。通过参数treeNum来指定森林中树的个数, 树个数的范围为[1, 1000]。通过编辑好的模版, 可以控制单棵树的结构。可以通过其他参数控制单颗树叶结点上数据的最小个数, 叶结点上数据占父节点的最小比例数, 树的最大深度等。
- 当出现input table is empty!错误时, 可能是以下情况: 抽样比率设置太小, 也就是maxRecordSize太小; 输入数据为空表。

预测

```
def predict(inputTableName, model, outputTableName,
            inputPartitions = None, appendColNames = None, outputPartition = None,
            isBin = False, labelValueToPredict = None):
```

参数:

- inputTableName: 预测输入表名。
- model: RandomForestModel类型。随机森林模型
- outputTableName: 预测输出表。
- inputPartitions: (可选)。输入表对应的输入分区, 选中全表则为None。

- appendColNames: (可选)。输出表中需要增加的预测输入表的列。
- outputPartition: (可选)。指定输出到输出表的分区。
- isBin: (可选)。模型对应的分类是否为二分。True为二分, False为多分。
- labelValueToPredict: (可选)。输出的概率对应的分类值。

示例:

```
rfModel = Classification.RandomForest.train('table_name', ['col1', 'col2'], [True, False],
    'label', "model_table_name", 10, inputPartitions = ['p=1'],
    weightColName = 'weight', templateTableName = 'template_table_name', validateTableName = 'vali_table_name',
    validatePartitions = ['p=1'], algorithmTypes = [3, 6], randomColNum = -1, minNumObj = -1, minNumPer = -1.,
    maxTreeDeep = 5, maxRecordSize = -1)
Classification.RandomForest.predict("table_name", rfModel, "out_table_name",
    inputPartitions = ['p=1'], appendColNames = ['label'], outputPartition = "p=1",
    isBin = True, labelValueToPredict = 'good')
```

注意事项:

- 随机森林预测根据输入表和模型, 输出预测结果。
- 本方法在二分类情况下, 森林中投票的过程完全按照概率进行计算, 可以有概率输出。在多分类情况下, 则是按照森林中每棵树给出的结论的个数来投票, 无概率输出。在二分有概率输出的情况下, 输出两列, 第一列为预测的结论, 第二列为labelValueToPredict 所指定的分类对应的概率。

是否为随机森林模型

```
def isRandomForestModel(modelTableName):
```

参数:

- modelTableName: 输入模型表名,

返回:

- True|False: 输出是否随机森林模型

示例:

```
isRFModel=Classification.RandomForest.isRandomForestModel("adult_model")
```

加载模型

```
def loadModel(modelTableName):
```

参数:

- modelTableName: 输入模型表名

返回:

- RandomForestModel: 输出随机森林模型

示例:

```
rfModel=Classification.RandomForest.loadModel("adult_model")
```

导出模型

导出随机森林模型, 目前支持导出单棵树记录小于10000条

```
def exportModel(model, outputPath, config):
```

参数:

- model: 输入随机森林模型, 模型中单棵树的规模小于10000
- outputPath: 输出模型的文件路径
- config: 导出配置

配置格式: [语法类型, 结论类型, (目标主分类值), (按结论分组排序), (每组保留规则数)], 前两个参数是必

- 语法类型: java/sql, 目前支持java和odps sql语法
- 结论类型: label/prop, label表示输出结论, prop表示输出goodValue的概率, 如果选择prop, 目标列必须是二分类, 且必须输入下面的目标主分类值
- 目标主分类值: (可选) 比如目标主分类值是1, 则表示给出1的概率
- 按结论分组排序: true
- 每组保留规则数: 在结论分组为true生效, double类型的数值

如: [‘java’, ‘label’], [‘java’, ‘label’, ‘true’],
[‘java’, ‘label’, ‘true’, ‘10’], [‘sql’, ‘prop’, ‘good’, ‘true’]

示例:

- java语法, 输出结论

```
adultModel=Classification.RandomForest.loadModel("adult_100_cart_model");
Classification.RandomForest.exportModel(adultModel,"D: \\ adult_model.rf",
                                         config=['java','label'])
```

- java语法, 输出结论, 规则集按结论分组

```
adultModel=Classification.RandomForest.loadModel("adult_100_cart_model");
Classification.RandomForest.exportModel(adultModel,"D: \\ adult_model.rf",
                                         config=['java','label','true'])
```

- java语法, 输出结论, 规则集按结论分组, 每组保留top10条记录

```
adultModel=Classification.RandomForest.loadModel("adult_100_cart_model");
Classification.RandomForest.exportModel(adultModel,"D:\\adult_model.rf",
                                         config=['java','label','true','10'])
```

- java语法, 输出good概率, 规则集按结论分组, 每组保留top10条记录

```
adultModel=Classification.RandomForest.loadModel("adult_100_cart_model");
Classification.RandomForest.exportModel(adultModel,"D:\\adult_model.rf",
                                         config=['java','prop','good','true','10'])
```

8.1.2 界面

使用训练集: tree_census_demo, 验证集: tree_census_demo_test

双击打开数据表, 模型菜单中选择随机森林, 如:



- “编辑查看”， 如果当前表是随机森林模型，那以树的方式，打开模型。如果当前表示普通表，则以编辑模板的方式，打开当前表的一个空模型。
- “训练”， 打开随机森林的训练界面
- “预测”， 打开随机森林的预测界面
- “导出模型”， 打开导出模型界面

训练

由菜单->随机森林>训练, 进入训练界面如下:



训练界面一共有三个tab页，分别为：属性选择，参数配置，模型评估配置。

- 属性选择tab页如上图，一共分为四个部分，
- 第一部分为属性选择(默认为除去最后一列的所有其他列)，显示属性的表一共三类，第一列表示当前属性是否选择，第二列是属性名称，第三列表示属性是否连续的，默认double和long会当成连续的。
- 第二部分为目标列选择(默认为最后一列)以及权重设置。可以选择一个数值列做为权重列，权重列每一个cell表示这一行记录的权重。
- 第三部分为模型输出表，用于存储模型数据，
- 第四部分为进度信息。
- 参数配置tab页如下图：
- 算法类型：四种，混合，ID3，C45，Cart
- 树的数目：训练树的数目
- 随机属性类型：算法每步分裂的属性数目M， $M \leq N$ ，N为属性总数目(不包含目标列)，这里提供选项分别N(全集)， $\log(N)$ ， \sqrt{N} ， $N/3$ ，1.
- 树最大深度：默认为0，深度无限制
- 叶子节点最少记录数：M，当节点记录少于M，停止分裂
- 叶子节点最小百分比：N，范围[0, 1)，当节点记录数少于父节点*N，停止分裂
- 每棵树最大记录数：每棵树处理的最大记录数，范围[1000, 1000000]
- 使用模板：是否使用模板，输入为普通表，如何定义下面会具体介绍

- 使用验证: 是否使用验证集, 输入为普通表。可以提前使用数据拆分模块, 拆分出训练集, 验证集。
(由于验证表是单节点计算的, 数据量建议不要超过512M)



- 模型评估配置tab页如下:



模型评估配置分为四部分, 第一部分评估输入表, 第二部分预测, 第三部分为计算混淆矩阵, 第四部分为

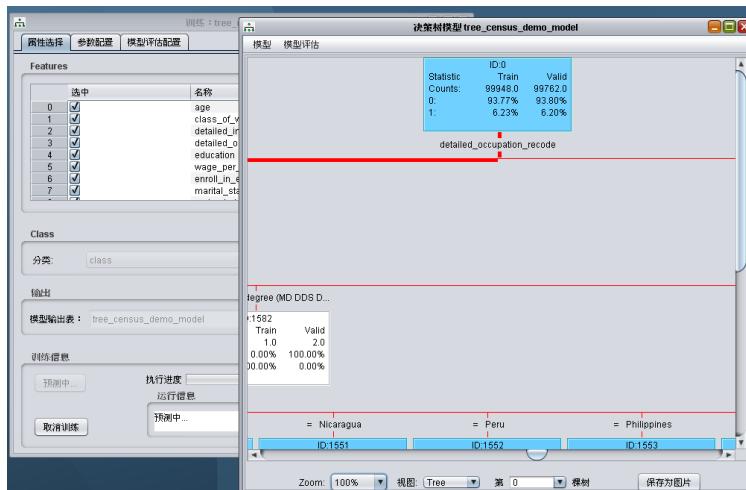
计算画图数据。

- 评估输入表：两个选项，可以用训练表使用模型进行评估；或者准备验证表(tab页面，必须输入验证表)，验证表使用模型评估
- 参数配置：如果选中”评价模型”或”计算画图数据”，预测输出表不可为空
- 指标：如果选中”评价模型”，填写混淆矩阵输出表。
- ROC, lift and Recall-FP：只有数据目标列是二分的，才可以画ROC, lift图，如果选中二分类，需要指定目标列主分类的值，以及画图数据存储的输出表

参数配置完成以后，回到属性选择tab页，单击”开始训练”，界面如下：



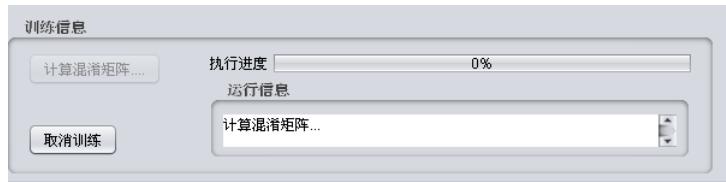
训练结束后会自动显示训练的模型，后面会详细介绍模型：



如果选中了”指标评价”，训练结束会，先运行预测，界面如下：



然后运行计算混淆矩阵，提示信息如下：



计算完成后，通过“模型界面的菜单”模型评估”>”混淆矩阵”，查看混淆矩阵如下：

```
Correctly Classified Instances      189581          95.0171 %
Incorrectly Classified Instances   9942           4.9829 %
Kappa statistic                   0.3522
Total Number of Instances        199523

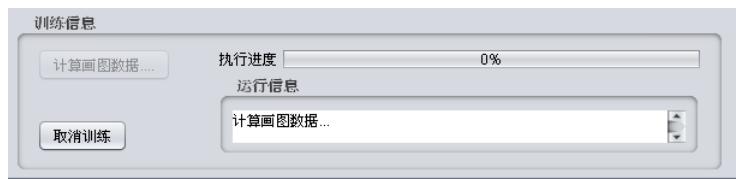
== Detailed Accuracy By Class ==

      TP Rate   FP Rate   Precision   Recall   F-Measure   Class
          0.997     0.765     0.952     0.997     0.974       0
          0.235     0.003     0.86      0.235     0.369       1
Weighted Avg.      0.95      0.717     0.946     0.95      0.937

== Confusion Matrix ==

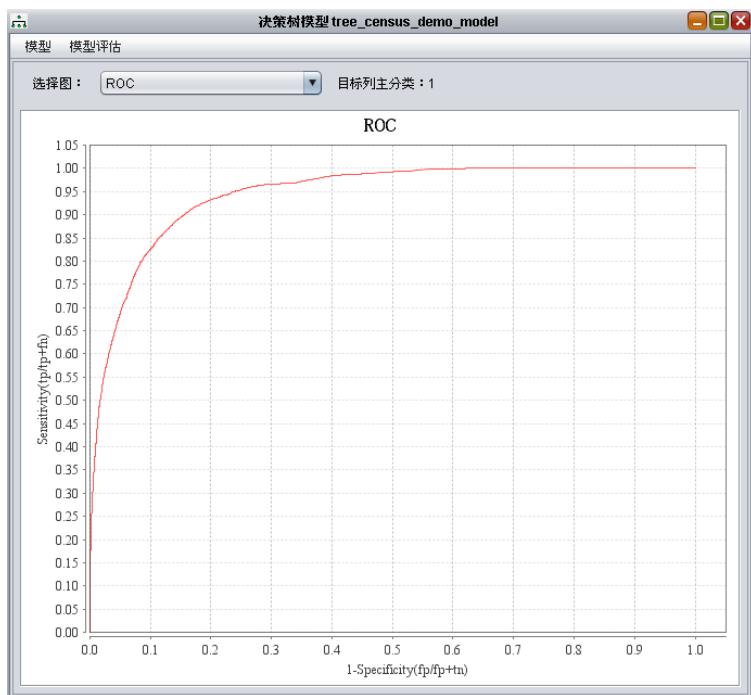
      a      b      TOTAL    <-- classified as
186668     473    187141 |      a = 0
      9469    2913    12382 |      b = 1
196137    3386    199523
```

如果选中了”画图”，训练结束，先运行预测，然后运行画图：

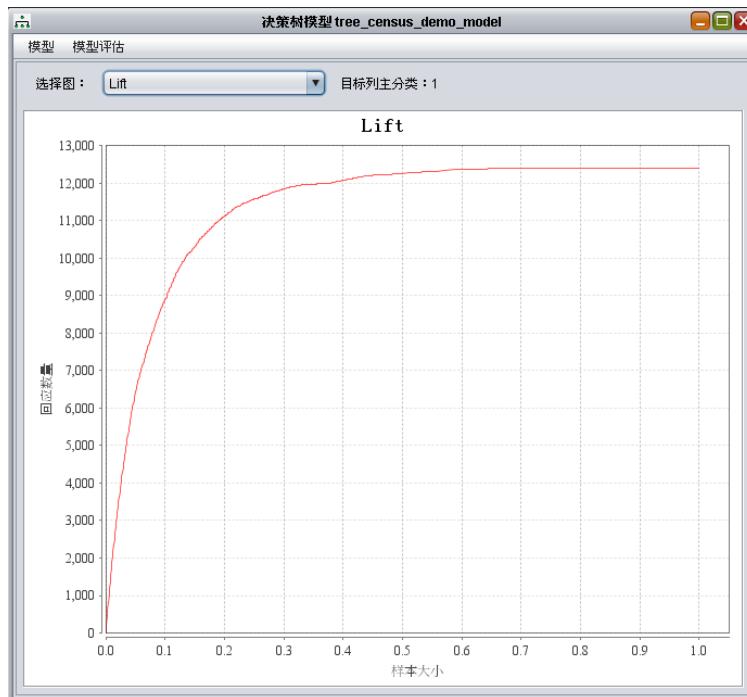


计算完成后，通过“模型界面的菜单”模型评估”>”ROC/Lift/Precious-Recall”，查看如下：

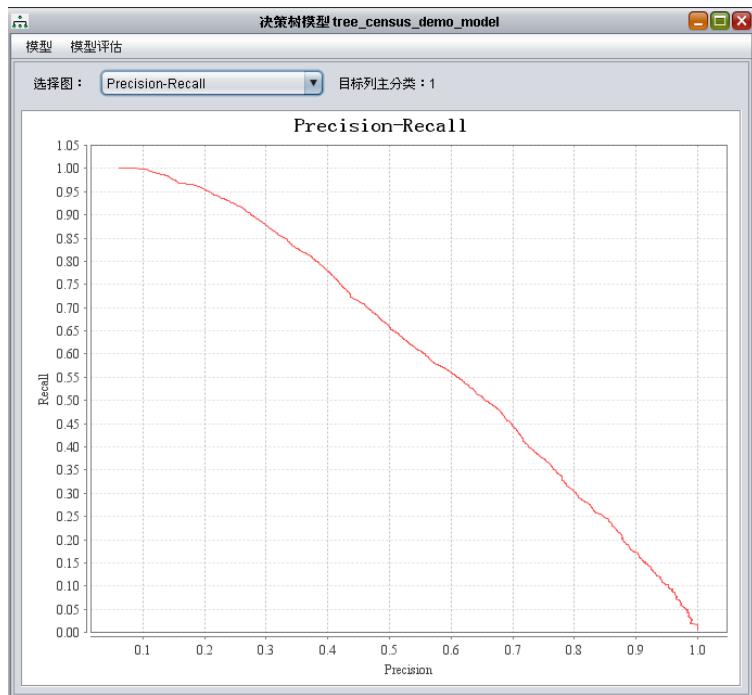
ROC图：



Lift图：



Precious-Recall:



预测

由菜单->随机森林>预测，进入预测界面如下：



其中：

- 预测输入表：要预测的表
- 结果附加列：可以选择预测输入表中需要输出到预测结果表的列，默认为无。
- 模型名：随机森林模型名
- 概率：如果目标列是二分的，且输入主分类，预测结果输出的是结论和输入分类的概率
- 输出表：预测输出表

单击“预测”，预测完成后，即可打开预测输出表查看预测结果：



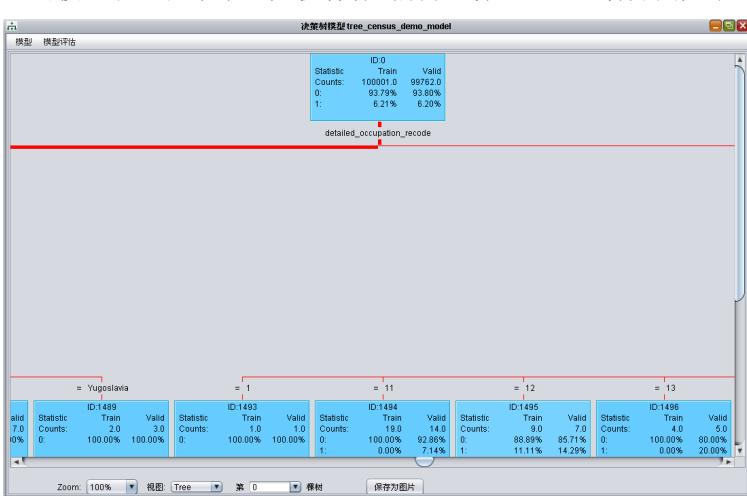
预测成功，可以单击表tree_census_demo_test_predict查看预测结果：

tree_census_demo_test_predict - 数据表浏览		
表格		数据处理
全表统计		条件统计
统计对比		
conclusion		probability
0	0	0.01427968799417521
1	0	0.0906946009392913
2	0	0.009105104821658072
3	0	0.006081586690155855
4	0	0.006081586690155855
5	0	0.09416668071378748
6	0	0.20896040420174236
7	0	0.04195450626547229
8	0	0.06601651913331914
9	0	0.08353567834522102
10	0	0.006081586690155855
11	0	0.014630877323388967
12	0	0.09422759789562876
13	0	0.08788231782859447
14	0	0.13161667110331124
15	0	0.006034387079591852
16	0	0.0798002506859265
17	0	0.018646165922965235
18	0	0.00783242011848176
19	0	0.11032546077659915
20	0	0.006667747494122751
21	0	0.09068310888389154

其中conclusion为预测的结果， probability为分类1的概率。

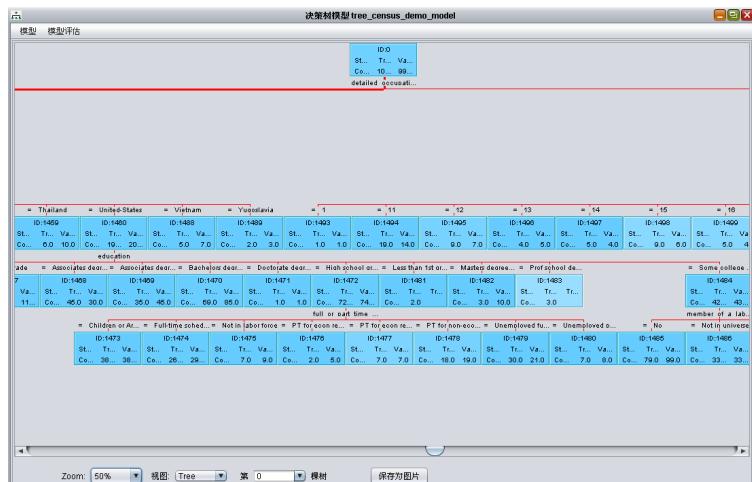
模型显示

双击模型表，在菜单→随机森林>编辑查看，进入查看界面如下：



模型界面页脚工具栏提供以下功能：

缩放：缩放50%如下：

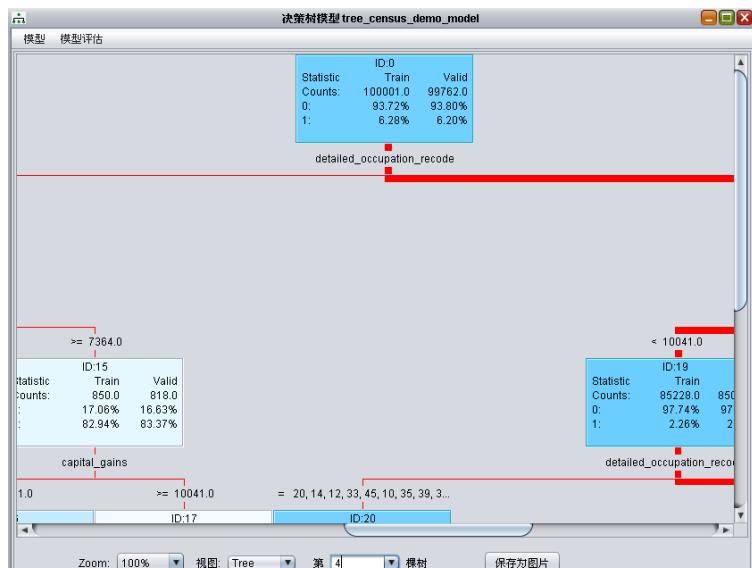


选择查看第几棵树:

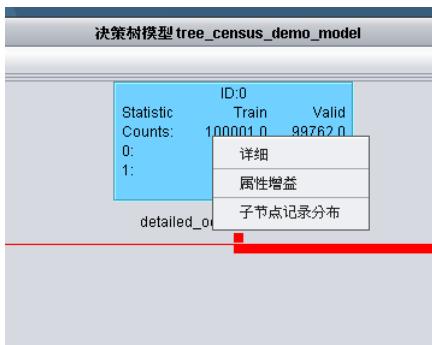


视图, 分为三种:

第一种为tree view, 如图:

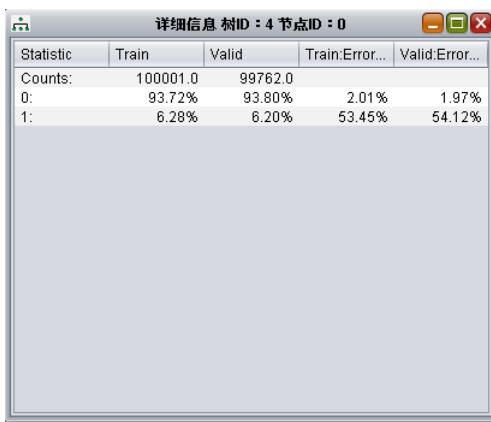


对于tree view显示视图, 默认树节点显示当前节点目标列group分布, 右击树节点, 如下图:



树节点支持三种事件，查看详细信息，查看属性增益，查看子节点记录分布：

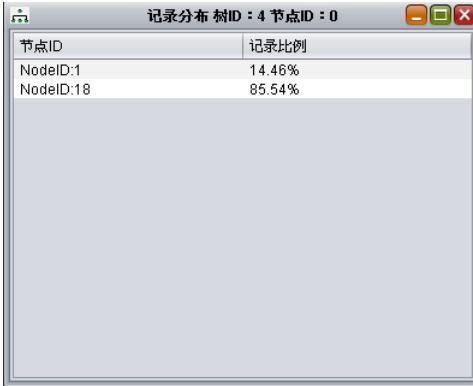
- 详细信息：显示节点训练数据目标列group分布，验证数据集目标列group分布，训练数据目标列group错误率，验证数据集目标列group错误率 如下：



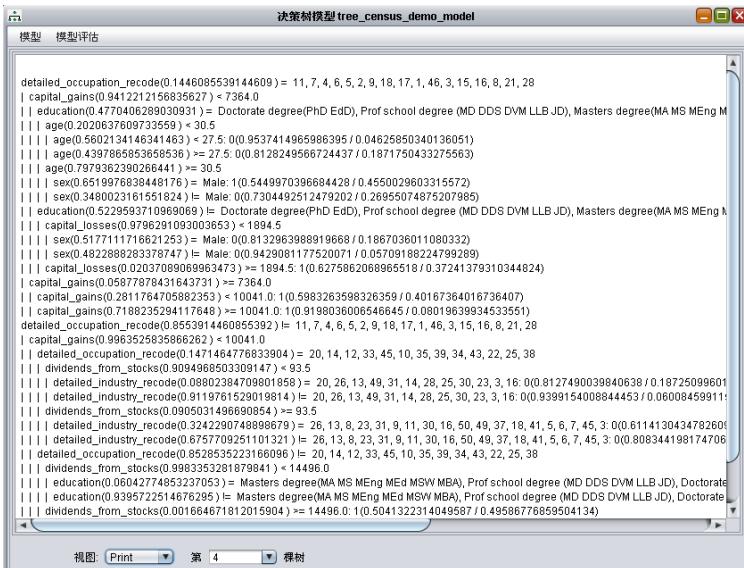
- 属性增益：显示属性名称，以及增益值(如果是ID3算法，则为信息增益，如果为C45，则为信息增益率，如果为Cart，则为gini增益)：

属性增益 树ID : 4 节点ID : 0	
属性值	增益
age	0.0048925
class_of_worker	0.005829
detailed_industry_recode	0.00807046
detailed_occupation_recode	0.0167876
education	0.0137693
wage_per_hour	6.87527E-4
enroll_in_edu_inst_last_wk	4.81553E-4
marital_stat	0.00448097
major_industry_code	0.00762145
major_occupation_code	0.0131261
race	3.76654E-4
hispanic_origin	5.11634E-4
sex	0.00290356
member_of_a_labor_union	6.55207E-4
reason_for_unemployment	8.87735E-5
full_or_part_time_employment...	0.00269953
capital_gains	0.0121529
capital_losses	0.00356315

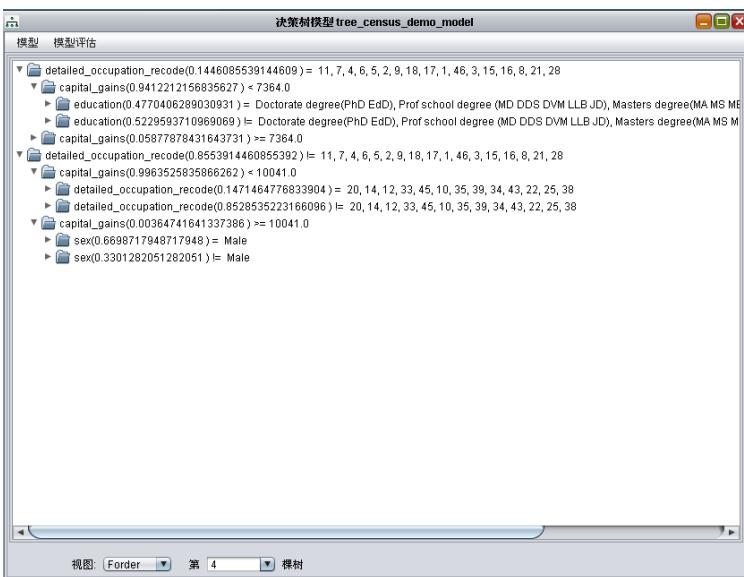
- 子节点记录分布：显示节点ID，以及节点记录数比例：



第二种为print view, 如图:



第三种为: folder view, 如图:



导出模型

“导出模型”模块用于导出随机森林模型到特定格式的文本，模型需要满足一下几个条件：

模型中单棵树的条数小于10000(由于ODPS odps_common_project 的限制)；模型导出支持两种格式：分别为JAVA 和 ODPS SQL两种格式。

选择模型数据表，右击表名选择→随机森林→导出模型，如：



- 语法类型：支持JAVA和ODPS SQL两种语法
- 结论类型：目标值和概率。在目标列是二分类时，支持概率输出，且必须指定目标主分类值，即输出谁的概率
- 输出规则：1) 默认是按深度优先遍历输出规则集 2) 可以按目标结论分组，且按记录数排序，在该条件下，支持每个分组保留topN条规则
- 保存路径：保存的文本路径，单击文本框后面的文件夹可以选择文件的保存路径，如下



单击“开始”，则开始导出模型，并显示导出的进度，如：

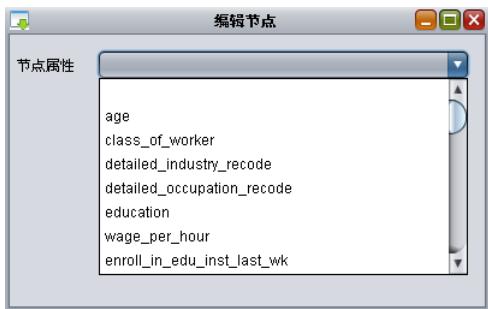


定义模板

双击打开非随机森林模型表，由菜单→随机森林>编辑查看，进入编辑界面如下：



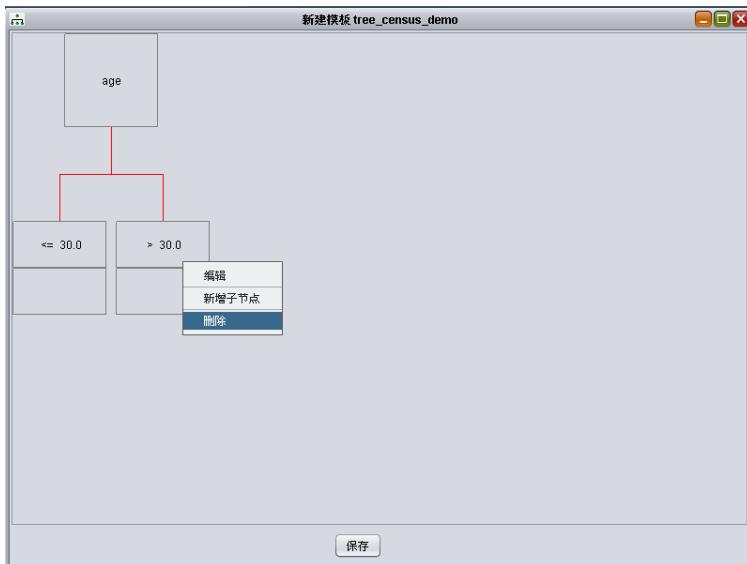
只有一个根节点，右击根节点，两个事件，“编辑”，“新增子节点”。其中编辑界面如下：



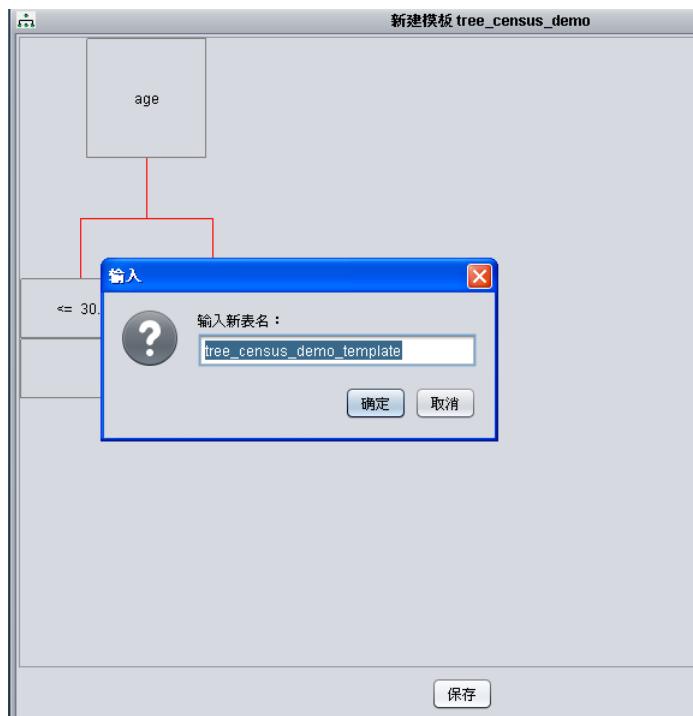
新增子节点界面如下：



除根节点外，其他节点多一个事件，删除节点，如图：



编辑完成后，单击保存，即可完成定义模板。



8.2 逻辑回归

经典逻辑回归是一个二分类算法，Xlab中的逻辑回归可以支持多分类。XLab中的逻辑回归模型分为三种：二分类逻辑回归，二分类逐步逻辑回归和多分类逻辑回归。逻辑回归算法介绍：[Logistic regression](#)。

- 二分类逻辑回归和多分类逻辑回归支持普通表和矩阵两种数据格式；而逐步逻辑回归只支持普通表。
- 对于普通表，要求特征列和label列（或者叫做目标列）都在同一张表里；而对于矩阵，要求整个矩阵都是特征，而label则在另外一张普通表中。
- 对于目标列，xlab支持富类型，包括boolean类型、bigint类型、string类型，double类型虽然也支持，但不鼓励使用。
- 逻辑回归使用数据表：adult进行使用说明。

特别说明：GoodValue和主分类是针对二分类的概念。

- GoodValue：对于逻辑回归，GoodValue就是逻辑回归中的正例，比如label列有1, 0两个值，一般来说就把1作为正例。
- 主分类：要输出概率的分类。

8.2.1 函数

显示帮助信息

- 二分类逻辑回归的训练和预测，以及模型load：

```
help(Classification.LogistReg.train)
help(Classification.LogistReg.predict)
help(Classification.LogistReg.trainSparse)
help(Classification.LogistReg.predictSparse)
help(Classification.LogistReg.loadModel)
```

- 二分类逐步逻辑回归的训练和预测, 以及模型load:

```
help(Classification.LogistRegStepWise.train)
help(Classification.LogistRegStepWise.predict)
help(Classification.LogistRegStepWise.loadModel)
```

- 多分类逻辑回归的训练和预测, 以及模型load:

```
help(Classification.MultiLogistReg.train)
help(Classification.MultiLogistReg.predict)
help(Classification.MultiLogistReg.trainSparse)
help(Classification.MultiLogistReg.predictSparse)
help(Classification.MultiLogistReg.loadModel)
```

二分类训练

普通表的二分类逻辑回归训练

```
def train(inputTableName, featureColNames, labelColName, modelTableName,
          inputPartitions = None, goodValue = None, maxIter = 100, epsilon = 1.0e-06,
          regularizedType = '11', regularizedLevel = 1.0):
```

参数:

- inputTableName: 训练数据的表名
- featureColNames: 特征列的列名数组, 列名必须是 inputTableName 里的列
- labelColName: label列(又叫目标列)的列名, 列名必须是 inputTableName 里的列
- modelTableName: 输出的模型的表名
- inputPartitions: (可选)训练数据所在的partition列表, 必须是 inputTableName 中的partition, 默认是None, 表示整表
- goodValue: (可选)指定训练系数针对的label值; 默认是随机选择一个, 模型里面会显示最终选择的是哪个
- maxIter: (可选)指定L-BFGS的最大迭代次数, 默认是100
- epsilon: (可选)指定终止条件, 就是两次迭代之间log-likelihood的差, 默认为1.0e-06
- regularizedType: (可选)正则化类型, 可以选择'11' 和'12', 或者None, 默认为'11'

- regularizedLevel: (可选) 正则项的系数; 当regularizedType为None时, 该项也可以写None; 注意需要写成1.0这种形式; 默认为1.0

返回:

- 二分类逻辑回归模型

示例:

```
Classification.LogistReg.train('bank_full', ['f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10', 'f11', 'f12', 'f13', 'f14']
```

```
Classification.LogistReg.train('bank_full', ['f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10', 'f11', 'f12', 'f13', 'f14']
```

矩阵的二分类逻辑回归训练

```
def trainSparse(featureMatrixName, labelTableName, labelColName, modelTableName,
                labelPartitions = None, goodValue = None, maxIter = 100, epsilon = 1.0e-06,
                regularizedType = '11', regularizedLevel = 1.0)
```

参数:

- featureMatrixName: 训练数据的特征所在矩阵的表名
- labelTableName: label(或者叫做目标)所在的表名
- labelColName: label列(又叫目标列)的列名, 列名必须是 labelTableName 里的列
- modelTableName: 输出的模型的表名
- labelPartitions: label(或者叫做目标)所在的分区
- goodValue: (可选) 指定训练系数针对的label值; 默认是随机选择一个, 模型里面会显示最终选择的是哪个
- maxIter: (可选) 指定L-BFGS的最大迭代次数, 默认是100
- epsilon: (可选) 指定终止条件, 就是两次迭代之间log-likelihood的差, 默认为1.0e-06
- regularizedType: (可选) 正则化类型, 可以选择'11' 和 '12', 或者None, 默认为'11'
- regularizedLevel: (可选) 正则项的系数; 当regularizedType为None时, 该项也可以写None; 注意需要写成1.0这种形式; 默认为1.0

返回:

- 二分类逻辑回归模型

示例:

```
Classification.LogistReg.trainSparse('bank_full_sparse', 'bank_full', 'label', 'bank_full_model')
```

```
Classification.LogistReg.trainSparse('bank_full_sparse', 'bank_full', 'label', 'bank_full_model', goodValue='1', maxIter=150)
```

[二分类预测](#)

[普通表的二分类逻辑回归预测](#)

```
def predict(inputTableName, model, outputTableName,
           inputPartitions=None, outputPartition=None, labelValueToPredict=None, appendColNames=None)
```

参数:

- inputTableName: 预测数据的表名
- model: 预测所用的二分类逻辑回归模型
- outputTableName: 预测输出表的表名
- inputPartitions: (可选) 预测输入数据的分区列表; None表示用全表
- outputPartition: (可选) 预测输出数据的分区; None表示不输出到分区
- labelValueToPredict: (可选) 预测所需要输出概率的label的值; None则不输出概率
- appendColNames: (可选) inputTableName中所需要附加到输出表的列名列表; None表示不附加任何列

示例:

```
bankFullModel = Classification.LogistReg.train('bank_full', ['f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10', 'f11', 'label'])

Classification.LogistReg.predict('bank_full', bankFullModel, 'predict_bank_full_result', labelValueToPredict='1', appendColNames=None)
```

矩阵的二分类逻辑回归预测

```
def predictSparse(featureMatrixName, model, outputTableName,
                  outputPartition=None, labelValueToPredict=None)
```

参数:

- featureMatrixName: 预测特征所在矩阵的表名
- model: 预测所用的二分类逻辑回归模型
- outputTableName: 预测输出表的表名
- outputPartition: (可选) 预测输出数据的分区; None表示不输出到分区
- labelValueToPredict: (可选) 预测所需要输出概率的label的值; None则不输出概率

示例:

```
bankFullModel = Classification.LogistReg.trainSparse('bank_full_sparse', 'bank_full', 'label', 'bank_full_model', goodValue=1)

Classification.LogistReg.predictSparse('bank_full_sparse', bankFullModel, 'predict_bank_full_result', labelValueToPredict='1')
```

二分类加载模型

从表加载一个二分类逻辑回归的模型

```
def loadModel(modelTableName)
```

参数:

- modelTableName: 二分类逻辑回归模型数据所在的表名

返回:

- 二分类逻辑回归模型

示例:

```
bankFullModel = Classification.LogistReg.loadModel('bank_full_model')
```

```
Classification.LogistReg.predictSparse('bank_full_sparse', bankFullModel, 'predict_bank_full_result', labelValueToPredict=1)
```

二分类逐步回归训练

普通表的二分类逐步逻辑回归

```
def train(inputTableName, featureColNames, labelColName, modelTableName, selectionInfoTableName,
          selectionMethod = 'stepwise',
          maxIter = 100, epsilon = 1.0e-06,
          inputPartitions = None, goodValue = '',
          slentry = 0.05, slstay = 0.05, include = 0)
```

参数:

- inputTableName: 训练数据的表名
- featureColNames: 特征列的列名数组, 列名必须是 inputTableName 里的列
- labelColName: label列(又叫目标列)的列名, 列名必须是 inputTableName 里的列
- modelTableName: 输出的模型的表名
- selectionInfoTableName: 存取选择过程信息的输出表名
- selectionMethod: (可选)可以是' stepwise' 和' forward' ; 默认为' stepwise' ;
- maxIter: (可选)指定Newton迭代的最大迭代次数, 默认是100
- epsilon: (可选)指定终止条件, 就是两次迭代之间log-likelihood的差, 默认为1.0e-06
- inputPartitions: (可选)训练数据所在的partition列表, 必须是 inputTableName 中的partition, 默认是None, 表示整表
- goodValue: (可选)指定训练系数针对的label值; 默认是随机选择一个, 模型里面会显示最终选择的是哪个
- slentry: (可选)控制特征进入模型的score test值的显著性水平; 默认为0.05
- slstay: (可选)控制特征继续留在模型中的wald test值的显著性水平; 默认为0.05
- include: (可选)强制featureColNames列表中的前多少个特征必须进入模型; 默认为0

返回:

- 二分类逐步逻辑回归模型

示例:

```
bankFullModel = Classification.LogistRegStepWise.train('bank_full', ['f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10']
```

二分类逐步回归预测

普通表的二分类逐步逻辑回归预测

```
def predict(inputTableName, model, outputTableName,
           inputPartitions=None, outputPartition=None, labelValueToPredict=None, appendColNames=None)
```

参数:

- inputTableName: 预测数据的表名
- model: 预测所用的二分类逐步逻辑回归模型
- outputTableName: 预测输出表的表名
- inputPartitions: (可选) 预测输入数据的分区列表; None表示用全表
- outputPartition: (可选) 预测输出数据的分区; None表示不输出到分区
- labelValueToPredict: (可选) 预测所需要输出概率的label的值; None则不输出概率
- appendColNames: (可选) inputTableName中所需要附加到输出表的列名列表; None表示不附加任何列

示例:

```
bankFullModel = Classification.LogistRegStepWise.train('bank_full', ['f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10']
Classification.LogistRegStepWise.predict('bank_full', bankFullModel, 'predict_bank_full_result', labelValueToPredict='1', ap
```

二分类逐步回归加载模型

从表加载一个二分类逻辑回归的模型

```
def loadModel(modelTableName)
```

参数:

- modelTableName: 二分类逻辑回归模型数据所在的表名

返回:

- 二分类逻辑回归模型

示例:

```
bankFullModel = Classification.LogistRegStepWise.loadModel('bank_full_model')
Classification.LogistRegStepWise.predict('bank_full', bankFullModel, 'predict_bank_full_result', labelValueToPredict='1', ap
```

多分类训练

普通表的多分类逻辑回归，通过one vs else的方式实现

```
def train(inputTableName, featureColNames, labelColName, modelTableName,
          inputPartitions = None, maxIter = 100, epsilon = 1.0e-06)
```

参数：

- inputTableName: 训练数据的表名
- featureColNames: 特征列的列名数组，列名必须是 inputTableName 里的列
- labelColName: label列(又叫目标列)的列名，列名必须是 inputTableName 里的列
- modelTableName: 输出的模型的表名
- inputPartitions: (可选)训练数据所在的partition列表，必须是 inputTableName 中的partition，默认是None，表示整表
- maxIter: (可选)指定L-BFGS的最大迭代次数，默认是100
- epsilon: (可选)指定终止条件，就是两次迭代之间log-likelihood的差，默认为 $1.0e-06$

返回：

- 多分类逻辑回归模型

示例：

```
bankFullModel = Classification.MultiLogistReg.train('bank_full', [f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10', 'f
```

矩阵的多分类逻辑回归，通过one vs else的方式实现

```
def trainSparse(featureMatrixName, labelTableName, labelColName, modelTableName,
                labelPartitions = None, maxIter = 100, epsilon = 1.0e-06)
```

参数：

- featureMatrixName: 训练数据的特征所在矩阵的表名
- labelTableName: label(或者叫做目标)所在的表名
- labelColName: label列(又叫目标列)的列名，列名必须是 labelTableName 里的列
- modelTableName: 输出的模型的表名
- labelPartitions: (可选)label(或者叫做目标)所在的分区
- maxIter: (可选)指定L-BFGS的最大迭代次数，默认是100

- epsilon: (可选) 指定终止条件, 就是两次迭代之间log-likelihood的差, 默认为 $1.0e-6$

返回:

- 多分类逻辑回归模型

示例:

```
bankFullModel = Classification.MultiLogitReg.trainSparse('bank_full_sparse', 'bank_full', 'label', 'bank_full_model', maxIt=100, epsilon=1.0e-6)
```

多分类预测

普通表的多分类逻辑回归预测

```
def predict(inputTableName, model, outputTableName,
           inputPartitions=None, outputPartition=None, outputProbability=False, appendColNames=None)
```

参数:

- inputTableName: 预测数据的表名
- model: 预测所用的二分类逻辑回归模型
- outputTableName: 预测输出表的表名
- inputPartitions: (可选) 预测输入数据的分区列表; None表示用全表
- outputPartition: (可选) 预测输出数据的分区; None表示不输出到分区
- outputProbability: (可选) 是否输出各label值的预测概率; 默认为False, 不输出
- appendColNames: (可选) inputTableName中所需要附加到输出表的列名列表; None表示不附加任何列

示例:

```
bankFullModel = Classification.MultiLogitReg.train('bank_full', [f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10', 'f11'])
Classification.MultiLogitReg.predict('bank_full', bankFullModel, 'predict_bank_full_result', outputProbability=True, appendColNames=None)
```

矩阵的多分类逻辑回归预测

```
def predictSparse(featureMatrixName, model, outputTableName,
                  outputPartition=None, outputProbability=False)
```

参数:

- featureMatrixName: 预测特征所在矩阵的表名
- model: 预测所用的多分类逻辑回归模型
- outputTableName: 预测输出表的表名
- outputPartition: (可选) 预测输出数据的分区; None表示不输出到分区

- outputProbability: (可选)是否输出各label值的预测概率的开关; 默认是False, 表示不输出

示例:

```
bankFullModel=Classification.MultiLogitReg.trainSparse('bank_full_sparse', 'bank_full', 'label', 'bank_full_model', maxIter=10)
Classification.MultiLogitReg.predictSparse('bank_full_sparse', bankFullModel, 'predict_bank_full_result', outputProbability=True)
```

多分类加载模型

从表名生成一个多分类逻辑回归的模型

```
def loadModel(modelTableName)
```

参数:

- modelTableName: 多分类逻辑回归模型数据所在的表名

示例:

```
bankFullModel = Classification.MultiLogitReg.loadModel('bank_full_model')
```

```
Classification.MultiLogitReg.predict('bank_full', bankFullModel, 'predict_bank_full_result', outputProbability=True, append=True)
```

正则项

通过调整正则项的参数regularizeType和regularizeLevel, 可以将拟合的模型泛化, 使得比较多的不太突出的特征, 拟合系数为0;

通过正则项参数regularizedType 和 regularizeLevel 可以指定 正则项 的系数(以下公式中的 C)。

$$\text{L1 正则: } \min_{\theta} \sum_{i=1}^m -\log p(y^{(i)}|x^{(i)}; \theta) + C \|\theta\|_1$$

$$\text{L2 正则: } \min_{\theta} \sum_{i=1}^m -\log p(y^{(i)}|x^{(i)}; \theta) + C \|\theta\|_2$$

注意事项:

- LR回归正则项输出, L1正则项系数比较大, weight为0的个数会增多。
- LR回归, 不能同时指定l1系数和l2系数。

8.2.2 界面

训练

选择数据表, 点击数据分析→逻辑回归→训练



其中：

- Partition筛选：选择参与逻辑回归模型训练的partition。
- 目标变量：数据表中目标分类列，支持Long, Integer, String, Boolean类型。
- 二分类：标记目标变量是否为二分类，当二分类选项被勾选时，逐步回归会被激活，需填写Good-Value。

- GoodValue: 当目标变量为二分类时, 需要指定GoodValue的值, 即逻辑回归运算中映射为1的值。
- 输入表达式列: 参与逻辑回归计算的列, 只支持数值型。
- 最大迭代次数: 使用优化方法求最优解时的最大迭代次数, 当迭代次数等于此值时, 不论是否收敛, 都将停止运算。
- 收敛误差: 当两次迭代的结果小于此值时, 认为算法已经收敛, 停止运算。
- 模型输出表: 存储逻辑回归模型的数据表。
- 逐步回归: 逻辑回归算法可选择逐步回归以筛选变量, 支持stepwise和forward两种模式, 当逐步回归被勾选时, 算法会进行多次逻辑回归运算: 保留那些对目标变量影响较大的变量, 剔除那些对目标变量影响较小的变量。
- 逐步回归信息输出表: 逐步回归的过程信息会输出至此表, 可从此表中获得每一步添加或删除一个变量的信息。
- 逐步回归方式: forward模式会逐步增加变量至模型中, stepwise模式会在每次增加变量后再尝试剔除变量。
- 引入变量阈值: 进行逐步回归欲引入变量score值的检验值, 只有当变量score值大于此值时才会把此变量引入。
- 剔除变量阈值: 进行逐步回归欲剔除变量wald值的检验值, 只有当变量wald值小于此值时才会把此变量剔除。
- 必选变量数: 进行逐步回归时模型中至少保留的变量数。

训练结果

会将所得模型表展示出来, 如下图所示:

模型信息表	
表格	数据处理
全表统计	条件统计
统计对比	数据分析
其它	
model_info	label_1
0	{"featuresList": ["age", ...]
1	Intercept -8.450481723123444
2	age 0.04337778999349031
3	fnlwgt 5.713663940680682E-7
4	education_num 0.3232041813363388
5	capital_gain 3.1861451264086156E-4
6	capital_loss 7.005608591429022E-4
7	hours_per_week 0.04090688306739953

预测



其中：

- Partition筛选：选择参与逻辑回归预测的partition。
- 使用模型：选择进行逻辑回归预测的模型。
- 预测输出表：选择预测结果的输出表。
- 保留列选择：选择会保留在预测结果表中的列。

预测结果

logisticpredresult - 数据表浏览													
表格 数据处理 全表统计 条件统计 统计对比 数据分析 其它													
predict_	class	age	workclass	fnwgt	education	education	marital_s	occupation	relationship	race	sex		
0	1	0	State...	77516	Bache...	13	Never...	Adm-c...	Not-i...	White	Male		
1	1	31	Private	45781	Masters	14	Never...	Prof...	Not-i...	White	Fem		
2	1	42	Private	159449	Bache...	13	Marri...	Exec...	Husband	White	Male		
3	1	0	Private	117037	11th	7	Marri...	Trans...	Husband	White	Male		
4	0	50	Self...	83311	Bache...	13	Marri...	Exec...	Husband	White	Male		
5	0	38	Private	215646	HS-grad	9	Divorced	Handl...	Not-i...	White	Male		
6	0	53	Private	234721	11th	7	Marri...	Husband	Black	Male			
7	0	28	Private	338409	Bache...	13	Marri...	Prof...	Wife	Black	Fem		
8	0	37	Private	284582	Masters	14	Marri...	Exec...	Wife	White	Fem		
9	0	49	Private	160187	9th	5	Marri...	Other...	Not-i...	Black	Fem		
10	0	52	Self...	209842	HS-grad	9	Marri...	Exec...	Husband	White	Male		
11	0	37	Private	280464	Somer...	10	Marri...	Exec...	Husband	Black	Male		
12	0	30	State...	141297	Bache...	13	Marri...	Prof...	Husband	Asian...	Male		
13	0	23	Private	122272	Bache...	13	Never...	Adm-c...	Own-c...	White	Fem		
14	0	32	Private	205019	Assoc...	12	Never...	Sales	Not-i...	Black	Male		
15	0	40	Private	121772	Assoc...	11	Marri...	Craft...	Husband	Asian...	Male		
16	0	34	Private	245487	7th-8th	4	Marri...	Trans...	Husband	Amer...	Male		
17	0	25	Self...	176756	HS-grad	9	Never...	Farmi...	Own-c...	White	Male		
18	0	32	Private	186824	HS-grad	9	Never...	Machi...	Unmar...	White	Male		
19	0	38	Private	28887	11th	7	Marri...	Sales	Husband	White	Male		
20	0	43	Self...	292175	Masters	14	Divorced	Exec...	Unmar...	White	Fem		
21	0	40	Private	193524	Doctor...	16	Marri...	Prof...	Husband	White	Male		
22	0	54	Private	302146	HS-grad	9	Separ...	Other...	Unmar...	Black	Fem		
23	0	35	Feder...	76845	9th	5	Marri...	Farmi...	Husband	Black	Male		
24	0	59	Private	109015	HS-grad	9	Divorced	Tech...	Unmar...	White	Fem		
25	0	56	Local...	216851	Bache...	13	Marri...	Tech...	Husband	White	Male		
26	0	19	Private	168294	HS-grad	9	Never...	Craft...	Own-c...	White	Male		
27	0	54	?	180211	Somer...	10	Marri...	?	Husband	Asian...	Male		

8.3 线性支持向量机(Linear SVM)

支持向量机(SVM)是建立在VC维理论和结构风险最小原理基础上的一种模式识别方法。目前的版本仅支持线性二分类问题，更多功能将在后续版本中逐步推出。SVM算法介绍：[Support vector machine](#)。

下文使用数据表：adult，进行介绍函数和界面的操作方法。

8.3.1 函数

线性SVM包括以下几个函数：train，trainSparse，predict，predictSparse和loadModel。

这几个函数具体使用方法，可用help命令查看，示例：

```
help(Classification.LinearSVM.train)
help(Classification.LinearSVM.trainSparse)
help(Classification.LinearSVM.predict)
help(Classification.LinearSVM.predictSparse)
help(Classification.LinearSVM.loadModel)
```

注意事项：

- 支持输入训练集格式为普通表和稀疏矩阵

训练

```
def train(inputTableName, featureColNames, labelColName, modelTableName, inputPartitions = None, cost = 1,
epsilon = 0.0001, positiveLabelValue = None, weight = [1.0, 1.0]):
```

参数:

- inputTableName: 输入表名, 普通表
- featureColNames: 特征列名, 字符数组
- labelColName: label列名
- modelTableName: 模型表名, 普通表
- inputPartitions: (可选)输入表分区
- cost: (可选)惩罚因子, 默认值为1
- epsilon: (可选)收敛系数, 默认值为0.001
- positiveLabelValue: (可选)从label中指定一类作为正类
- weight: (可选)惩罚因子权重, 实数数组, weight[0]为正类权重, weight[1]为负类权重, 默认值为1.0

返回:

- LinearSVMModel 线性svm模型

示例:

```
Classification.LinearSVM.train("train_input", ["height", "weight", "footsize"], "sex", "svm_model",
inputPartitions = None, cost = 100, epsilon = 0.0001, positiveLabelValue = "male", weight = [1.0, 2.0])
```

说明:

- 选择label列中值为“male”的类为正类, 其惩罚因子为 $100 \times 1.0 = 100$, 另一类为负类, 惩罚因子为 $100 \times 2.0 = 200$ 。

```
def trainSparse(featureMatrixName, labelTableName, labelColName, modelTableName, labelPartitions = None, cost = 1,
epsilon = 0.0001, positiveLabelValue = None, weight = [1.0, 1.0]):
```

参数:

- featureMatrixName: 输入 Xlib稀疏矩阵表 名
- labelTableName: label列所在表名
- labelColName: label列名
- modelTableName: 模型表名, 普通表
- labelPartitions: (可选)label列所在表分区
- cost: (可选)惩罚因子, 默认值为1

- epsilon: (可选) 收敛系数, 默认值为 0.001
- positiveLabelValue: (可选) 从 label 中指定一类作为正类
- weight: (可选) 惩罚因子权重, 实数数组, weight[0] 为正类权重, weight[1] 为负类权重, 默认值为 1.0

返回:

- LinearSVMModel 线性 SVM 模型

示例:

```
Classification.LinearSVM.trainSparse("train_input_sparse", "train_input", "sex", "svm_model_sparse",
labelPartitions = None, cost = 1, epsilon = 0.0001, positiveLabelValue = "male", weight = [1.0, 2.0])
```

预测

```
def predict(inputTableName, model, outputTableName, inputPartitions = None, appendColNames = None, outputPartition = None):
```

参数:

- inputTableName: 输入表名, 普通表
- model: 模型, 类型为 LinearSVMModel
- outputTableName: 输出表名, 普通表
- inputPartitions: (可选) 输入表分区
- appendColNames: (可选) 输入表中插入到输出表中的列名
- outputPartition: (可选) 输出表分区

返回:

- void 无

示例:

```
svmmode=Classification.LinearSVM.train("train_input", ["height", "weight", "footsize"], "sex",
"svm_model", inputPartitions = None, cost = 1, epsilon = 0.0001)
```

```
Classification.LinearSVM.predict("predict_input", svmmode, "svm_model_predict_table_out",
inputPartitions = None, appendColNames = None, outputPartition = None)
```

```
def predictSparse(inputTableName, model, outputTableName, outputPartition = None):
```

参数:

- inputTableName: 输入表名, Xlib 稀疏矩阵表
- model: 模型, 类型为 LinearSVMModel

- outputTableName: 输出表名, 普通表
- outputPartition: (可选)输出表分区

返回:

- void 无

示例:

```
svmmode=Classification.LinearSVM.trainSparse("train_input_sparse", "train_input", "sex",
    "svm_model_sparse", labelPartitions = None, cost = 1, epsilon = 0.001)

Classification.LinearSVM.predictSparse("predict_sparse_input", svmmode, "svm_model_predict_sparse_out",
    outputPartition = None)
```

加载模型

```
def loadModel(modelTableName):
```

参数:

- modelTableName: 模型表表名, 普通表

返回:

- LinearSVMModel 线性svm模型

示例:

```
Classification.LinearSVM.loadModel("svm_model")
```

8.3.2 界面

界面包括: 训练和预测。

- 只支持输入训练集格式为普通表。

训练

通过UI界面(只支持输入训练集格式为普通表)选择数据表, 点击模型→线性支持向量机→训练



其中：

- Partition筛选：选择参与线性支持向量机模型训练的partition。
- 目标变量：数据表中目标分类列，支持Long, Integer, String, Boolean类型，目前只支持两类。
- - 输入表达式中列：参与线性支持向量机计算的列，只支持数值型。
- 模型输出表：线性支持向量机模型输出表。
- 惩罚因子：C，软间隔SVM中的用来衡量对噪声点的重视程度。

- 收敛系数：目标函数梯度的相对变化低于该值，认为算法收敛。
- 目标变量正例值：从label中选择一类作为正类，可对不同类样本使用不同的惩罚因子权重。
- 正、负例惩罚因子权重：weightP和weightN分别表示对正类和负类的惩罚因子权重。实际的惩罚因子 = C × 相应权重。
- 注意，对惩罚因子加权重适用于数据偏斜非常严重（比如正类有10000个而负类只有100个）的情况，此时对样本数少的类建议使用较大的惩罚因子。

选择完毕单击“确定”开始训练。训练结果：模型表，其中最后一行表示截距，如下图：

model_info		coeff
0	{"featuresList": [], "featuresSize": 3, ...}	0
1	0	-0.9702073775480459
2	1	-0.002882458868165536
3	2	0.6696097694063672
4	Intercept	-0.25708196852794357

预测

通过UI界面(只支持输入预测数据集为普通表，且输入模型表由普通表训练得到)选择数据表，点击模型→线性支持向量机→预测



其中：

- Partition筛选：选择参与线性SVM预测的partition。
- 使用模型：选择进行线性SVM预测的模型表。
- 预测输出表：选择预测结果的输出表。
- 保留列选择：选择会保留在预测结果表中的列。

选择完毕单击“确定”开始预测。预测结果：说明：预测结果包括两列，第一列表示判决结果，第二列表示具体的预测值。

预测结果表

表格 数据处理 全表描述 统计 分析 模型 工具 画图		
	conclusion	predict_values
0	male	1.0356289611152147
1	male	0.992300709545787
2	male	1.1318222532979976
3	male	0.9329360036848089
4	female	-0.6001097698770546
5	female	-0.8414691558352153
6	female	-0.9160571806261968
7	female	-0.8416340183914186

数据大小: 8 行, 当前显示前 行 (最多10000行)

8.4 非线性支持向量机(Nonlinear SVM)

基于随机梯度下降(SGD)方法开发的分布式非线性二分类SVM算法，支持大规模数据(百亿非零元规模)的非线性SVM训练。目前支持RBF, Polynomial和Sigmoid三种核函数，核心算法可参考文献Z. A. Zhu, W. Chen, G. Wang, C. Zhu, and Z. Chen. P-packsvm: parallel primal gradient descent kernel svm. In IEEE International Conference on Data Mining, 2009. Note: 输入特征值只支持数值类型。下文使用数据表: train_input, 进行介绍函数和界面的操作方法。

8.4.1 函数

非线性SVM包括以下几个函数: train, trainSparse, predict, predictSparse和loadModel。

这几个函数具体使用方法, 可用help命令查看, 示例:

```
help(Classification.NonlinearSVM.train)
help(Classification.NonlinearSVM.trainSparse)
help(Classification.NonlinearSVM.predict)
```

```
help(Classification.NonlinearSVM.predictSparse)
help(Classification.NonlinearSVM.loadModel)
```

注意事项:

- 支持输入训练集格式为普通表和稀疏矩阵

训练

```
def train(inputTableName, featureColNames, labelColName, modelTableName, inputPartitions = None,
          cost = 1.0, iters = None, kernel_type = "rbf", gamma = 1.0, coef0 = 0.0, degree = 2):
```

参数:

- inputTableName: 输入表名, 普通表
- featureColNames: 特征列名, 字符数组
- labelColName: label列名
- modelTableName: 模型表名, 普通表
- inputPartitions: (可选)输入表分区
- cost: (可选)惩罚因子, 默认值为1
- iters: (可选)迭代次数, 控制收敛精度, 其默认值满足iters = k * 样本数量m, 其中 k 的默认值由以下经验公式给出: k = 10, if log10(m) <= 3;

$k = (-9) * \log_{10}(m) + 37$, if $\log_{10}(m) > 3 \&& \log_{10}(m) \leq 4$;

$k = (-0.5) * \log_{10}(m) + 3$, if $\log_{10}(m) > 4 \&& \log_{10}(m) \leq 5$;

$k = (-0.25) * \log_{10}(m) + 1.75$, if $\log_{10}(m) > 5 \&& \log_{10}(m) \leq 6$;

$k = (-0.125) * \log_{10}(m) + 1$, if $\log_{10}(m) > 6 \&& \log_{10}(m) \leq 7$;

$k = -(7.0 / 64.0) * \log_{10}(m) + (57.0 / 64.0)$, if $\log_{10}(m) > 7 \&& \log_{10}(m) \leq 8$;

$k = 1.0 / 64.0$, else

注意: 迭代次数控制收敛精度, 默认值综合考虑了收敛精度与运行时间, 通常情况下用默认值就能达到很好的效果。用户可根据需要改变iters值。出于运行时间的考虑, iters的上限目前定为 $2*10^6$ 次。

- kernel_type: (可选)目前提供以下三种核函数: “rbf” (高斯径向基, 表达式 $k(u, v) = \exp(-\gamma * |u - v|^2)$)
- “poly” (多项式, 表达式 $k(u, v) = (\gamma * u' v + c)^d$)
- “sigmoid” (sigmoid, 表达式 $k(u, v) = \tanh(\gamma * u' v + c)$) 默认值为” rbf”
- gamma: (可选)核函数系数(权重), 默认值0.01

- coef0: (可选)核函数系数(截距), 默认值0
- degree: (可选)核函数系数(指数), 默认值2

注意: Sigmoid核是条件正定 (conditionally positive definite) 的, 其参数取值范围可参考文献: H.-T. Lin, C.-J. Lin, A study on sigmoid kernels for SVM and the training of non-PSD kernels by SM0-type methods.

返回:

- NonlinearSVMModel 非线性svm模型

示例:

```
Classification.NonlinearSVM.train("train_input", ["height", "weight", "footsize"], "sex",
"svm_model", inputPartitions = None, cost = 1.0, iters = 10, kernel_type = "rbf", gamma = 0.01)

def trainSparse((featureMatrixName, labelTableName, labelColName, modelTableName, labelPartitions = None,
cost = 1.0, iters = None, kernel_type = "rbf", gamma = 1.0, coef0 = 0.0, degree = 2):
```

参数:

- featureMatrixName: 输入 Xlib稀疏矩阵表 名
- labelTableName: label列所在表名
- labelColName: label列名
- modelTableName: 模型表名, 普通表
- labelPartitions: (可选)label列所在表分区
- cost: (可选)惩罚因子, 默认值为1
- iters: (可选)迭代次数, 控制收敛精度, 其默认值满足iters = k * 样本数量m, 其中 k的默认值由以下经验公式给出: k = 10, if log10(m) <= 3;

```
k = (-9) * log10(m) + 37, if log10(m) > 3 && log10(m) <= 4;
k = (-0.5) * log10(m) + 3, if log10(m) > 4 && log10(m) <= 5;
k = (-0.25) * log10(m) + 1.75, if log10(m) > 5 && log10(m) <= 6;
k = (-0.125) * log10(m) + 1, if log10(m) > 6 && log10(m) <= 7;
k = -(7.0 / 64.0) * log10(m) + (57.0 / 64.0), if log10(m) > 7 && log10(m) <= 8;
k = 1.0 / 64.0, else
```

注意: 迭代次数控制收敛精度, 默认值综合考虑了收敛精度与运行时间, 通常情况下用默认值就能达到很好的效果。用户可根据需要改变iters值。出于运行时间的考虑, iters的上限目前定为 2×10^6 次。

- kernel_type: (可选)目前提供以下三种核函数: “rbf” (高斯径向基, 表达式 $k(u, v) = \exp(-\gamma \|u - v\|^2)$)

- “poly” (多项式, 表达式 $k(u, v) = (\gamma * u' \cdot v + \text{coef0}) ^ \text{degree}$)
- “sigmoid” (sigmoid, 表达式 $k(u, v) = \tanh(\gamma * u' \cdot v + \text{coef0})$) 默认值为 “rbf”
- gamma: (可选)核函数系数(权重), 默认值1.0
- coef0: (可选)核函数系数(截距), 默认值0
- degree: (可选)核函数系数(指数), 默认值2

返回:

- NonlinearSVMModel 非线性svm模型

示例:

```
Classification.NonlinearSVM.trainSparse("train_input_sparse", "train_input", "sex",
"svm_model_sparse", labelPartitions = None, cost = 1, iters = 10, kernel_type = "rbf", gamma = 0.01)
```

预测

```
def predict(inputTableName, model, outputTableName, inputPartitions = None,
appendColNames = None, outputPartition = None):
```

参数:

- inputTableName: 输入表名, 普通表
- model: 模型, 类型为NonlinearSVMModel
- outputTableName: 输出表名, 普通表
- inputPartitions: (可选)输入表分区
- appendColNames: (可选)输入表中插入到输出表中的列名
- outputPartition: (可选)输出表分区

返回:

- void 无

示例:

```
svmmode=Classification.NonlinearSVM.train("train_input", ["height", "weight", "footsize"], "sex",
"svm_model", inputPartitions = None, cost = 1)
Classification.NonlinearSVM.predict("predict_input", svmmode, "svm_model_predict_table_out",
inputPartitions = None, appendColNames = None, outputPartition = None)
```

```
def predictSparse(inputTableName, model, outputTableName, outputPartition = None):
```

参数:

- inputTableName: 输入表名, Xlib稀疏矩阵表

- model: 模型, 类型为NonlinearSVMModel
- outputTableName: 输出表名, 普通表
- outputPartition: (可选)输出表分区

返回:

- void 无

示例:

```
svmmode1=Classification.NonlinearSVM.trainSparse("train_input_sparse", "train_input", "sex",
"svm_model_sparse", labelPartitions = None, cost = 1)

Classification.NonlinearSVM.predictSparse("predict_sparse_input", svmmode1,
"svm_model_predict_sparse_out", outputPartition = None)
```

加载模型

```
def loadModel(modelTableName):
```

参数:

- modelTableName: 模型表表名, 普通表

返回:

- NonlinearSVMModel 非线性SVM模型

示例:

```
Classification.NonlinearSVM.loadModel("svm_model")
```

8.4.2 界面

界面包括: 训练和预测。

- 只支持输入训练集格式为普通表。

训练

通过UI界面(只支持输入训练集格式为普通表)选择数据表, 点击模型→非线性支持向量机→训练



其中：

- Partition筛选：选择参与非线性支持向量机模型训练的partition。
- 目标变量：数据表中目标分类列，支持Long, Integer, String, Boolean类型，目前只支持两类。

- - 输入表选中列: 参与非线性支持向量机计算的列, 只支持数值型。
- 模型输出表: 非线性支持向量机模型输出表。
- 惩罚因子: C, 软间隔SVM中的用来衡量对噪声点的重视程度。
- iters: (可选) 迭代次数, 控制收敛精度, 其默认值满足 $\text{iters} = k * \log_{10}(m)$, 其中 k 的默认值由以下经验公式给出


```

k = 10, if log10(m) <= 3;
k = (-9) * log10(m) + 37, if log10(m) > 3 && log10(m) <= 4;
k = (-0.5) * log10(m) + 3, if log10(m) > 4 && log10(m) <= 5;
k = (-0.25) * log10(m) + 1.75, if log10(m) > 5 && log10(m) <= 6;
k = (-0.125) * log10(m) + 1, if log10(m) > 6 && log10(m) <= 7;
k = -(7.0 / 64.0) * log10(m) + (57.0 / 64.0), if log10(m) > 7 && log10(m) <= 8;
k = 1.0 / 64.0, else
      
```
- kernel_type: (可选) 目前提供以下三种核函数: “rbf” (高斯径向基, 表达式 $k(u, v) = \exp(-\gamma \|u - v\|^2)$)
 “poly” (多项式, 表达式 $k(u, v) = (\gamma * u' v + \gamma_0)^{\gamma}$)
 “sigmoid” (sigmoid, 表达式 $k(u, v) = \tanh(\gamma * u' v + \gamma_0)$) 默认值为 “rbf”
- gamma: (可选) 核函数系数(权重), 默认值 0.01
- coef0: (可选) 核函数系数(截距), 默认值 0
- degree: (可选) 核函数系数(指数), 默认值 2

选择完毕单击“确定”开始训练。训练结果: 第0行第0列记录模型基本信息, 从第1行开始, 每行记录一条支持向量, 其中第0列表示对应支持向量的权重

	model_info	height	weight	footsize
0	{ "coef0": "0.0", ...	0	0	0
1	0.5	6	180	12
2	0.8	5.92	190	11
3	-0.6	5	100	6
4	-0.3	5.5	150	8
5	0.5	5.58	170	12
6	0.6	5.92	165	10
7	-0.9	5.42	130	7
8	-0.6	5.75	150	9

预测

通过UI界面(只支持输入预测数据集为普通表, 且输入模型表由普通表训练得到)选择数据表, 点击模型→非线性支持向量机→预测



其中：

- Partition筛选：选择参与非线性SVM预测的partition。
- 使用模型：选择进行非线性SVM预测的模型表。
- 预测输出表：选择预测结果的输出表。

- 保留列选择：选择会保留出现在预测结果表中的列。

选择完毕单击“确定”开始预测。预测结果：说明：预测结果包括两列，第一列表示判决结果，第二列表示具体的预测值

The screenshot shows a software window titled "预测结果表" (Prediction Results Table). The menu bar includes "表格" (Table), "数据处理" (Data Processing), "全表描述" (Full Table Description), "统计" (Statistics), "分析" (Analysis), "模型" (Model), "工具" (Tools), and "画图" (Plot). The main area displays a table with two columns: "conclusion" and "predict_values". The "conclusion" column contains categorical values ("male" or "female") for rows 0 through 7. The "predict_values" column contains numerical values for each row. At the bottom of the table, there are buttons for "Whole Table" (显示整个表格), "位置" (Position), "数据大小: 8 行, 当前显示前 50 行" (Data Size: 8 rows, Current Display Before 50 rows), "行 (最多10000行)" (Rows (Up to 10000 rows)), and "GO!".

	conclusion	predict_values
0	male	1.0356289611152147
1	male	0.992300709545787
2	male	1.1318222532979976
3	male	0.9329360036848089
4	female	-0.6001097698770546
5	female	-0.8414691558352153
6	female	-0.9160571806261968
7	female	-0.8416340183914186

8.5 朴素贝叶斯(Naive Bayes)

朴素贝叶斯分类是一种应用基于独立假设的贝叶斯定理的简单概率分类算法。更精确的描述这种潜在的概率模型为独立特征模型。 算法详见：[Naive Bayes classifier](#)

为便于说明，先约定如下：

- 训练输入表：train_input
- 模型表：naive_bayes_model
- 预测输入表：predict_input
- 预测输出表：naive_bayes_predict_output
- 特征列名：["height", "weight", "footsize"]
- Label列名：“sex”

朴素贝叶斯模型的执行途径有两种：函数和界面。其主要涉及两个过程：训练和预测，详见下文。

8.5.1 函数

朴素贝叶斯包括以下几个函数：train，predict和loadModel。三个函数具体使用方法，可用help命令查看，示例：

```
help(Classification.NaiveBayes.train)
help(Classification.NaiveBayes.predict)
help(Classification.NaiveBayes.loadModel)
```

训练

```
def train(inputTableName, featureColNames, labelColName, modelTableName, inputPartitions=None, categoryColNames = None):
```

参数：

- inputTableName: 输入表名，普通表
- featureColNames: 连续型特征列名，字符数组
- labelColName: label列名
- modelTableName: 模型表名，普通表
- inputPartitions: (可选)输入表分区
- categoryColNames: (可选)离散型特征列名，字符数组

返回：

- NaiveBayesModel 朴素贝叶斯模型

示例：

```
Classification.NaiveBayes.train("train_input", ["height", "weight", "footsize"], "sex",
"naive_bayes_model", inputPartitions = None)
```

预测

```
def predict(inputTableName, model, outputTableName, inputPartitions=None, appendColNames=None,
outputPartition=None):
```

参数：

- inputTableName: 输入表名，普通表
- model: 模型，类型为朴素贝叶斯NaiveBayesModel
- outputTableName: 输出表名，普通表

- inputPartitions: (可选)输入表分区
- appendColNames: (可选)输入表中插入到输出表中的列名
- outputPartition: (可选)输出表分区

返回:

- void 无

示例:

```
nvmodel=Classification.NaiveBayes.train("train_input", ["height", "weight", "footsize"], "sex", "naive_bayes_model")
Classification.NaiveBayes.predict("predict_input", nvmodel, "naive_bayes_model_predict_out")
```

加载模型

```
def loadModel(modelTableName):
```

参数:

- modelTableName: 模型表表名, 普通表

返回:

- NaiveBayesModel 朴素贝叶斯模型

示例:

```
Classification.NaiveBayes.loadModel("naive_bayes_model")
```

注意事项:

- 该分类算法, 对于训练和预测的输入输出, 仅支持普通表(table);
- 特征值(feature)仅支持数值类型: Double, Bigint, label值支持Double, Bigint, Boolean, String;
- 对于预测输出结果, 可以选择将预测输入的若干列附加到预测输出表中。

8.5.2 界面

界面包括: 训练和预测。

训练

- 输入: 普通表, 包含label列和数值类型feature列的表, 如train_input。
- 输出: 模型表, 包括预测用到的一些参数, 共五列, 如naive_bayes_model。

训练步骤如下：

首先，打开训练输入表train_input，然后点击”模型”—”朴素贝叶斯(Naïve Bayes)”—“训练”，如下图：



然后，在训练窗口，在对应提示区域填写相应参数，“Partition筛选”，“选择连续列”，“选择离散列”，“Label列”，“输出表表名”；点击界面右下角按钮“训练”开始训练，如下图：



- 对于” Partition筛选”，默认为全选，如果需要对某个分区操作，可以选择” Partition筛选” 对应的修改按钮，没有分区则不用修改。
- 对于” 选择连续列”，点击” 修改” 后，候选列表均为数值类型，用户可根据需要选择feature，如：train_input的特征列为[” height”， “weight”， ” footsize”]。
- 对于” 选择离散列”，点击” 修改” 后，候选列表为所有类型，用户可根据需要选择离散列。
- 对于” Label列” 选择，不能跟feature列有交集，如：train_input的目标列(即label列)为sex。

- 对于“输出表表名”，如：naive_bayes_model。

最后，模型训练成功，自动打开model表naive_bayes_model，如下图：

朴素贝叶斯模型表:naive_bayes_model					
	vallabel	plabel	namefeature	mean	variance
0	female	0.5	height	5.4175	0.0972249999999457
1	female	0.5	weight	132.5	558.3333333333334
2	female	0.5	footsize	7.5	1.6666666666666667
3	male	0.5	height	5.855	0.0350333333333103
4	male	0.5	weight	176.25	122.91666666666667
5	male	0.5	footsize	11.25	0.9166666666666666
6	{“version”:”v1.0...”}				

朴素贝叶斯模型表包括两种信息：conf和data信息

- conf信息包括：version, featureSize和 featureList
- data信息包括以下数据：
 - valLabel: label的值
 - pLabel: label的概率
 - nameFeature: 特征名
 - mean: feature相对于label的均值
 - variance: feature相对于label的方差
- 对于0.13.4及以后版本的XLab, NaiveBayes支持离散型feature, 包含离散型feature的模型还会包括：
 - double value/long value/boolean value/string value/date value: 分别对应某一离散列的某个真实值

- probability: 离散列的某个真实值在当前label下的概率
- valueindex: 真实值所处列的索引

预测

- 输入: 包括输入表和模型表, 输入表, 如predict_input, 模型表, 如naive_bayes_model。
- 输出: 预测结果输出表, 如naive_bayes_predict_output。输出表一般只有一列conclusion, 也可以append输入表中的若干列列在输出表中。

预测步骤如下:

首先, 可打开”输入表”或者”模型表”

- 如果打开的是”输入表” predict_input, 点击”模型” —”朴素贝叶斯(Naïve Bayes)” —”预测”, 预测表表名会被自动加载, 用户可在” Partition筛选” 区域选择修改Partition, 在” Model 表名” 处填写模型表名, 如下图:



- 如果打开的是”模型表” naive_bayes_model, 点击”模型” —”朴素贝叶斯(Naïve Bayes)” —”预测”, Model表名会被自动加载, 用户需在”预测表”区域, 填写表名, 点击”加载Partition”后, 可进行” Partition筛选” 如下图:



- 对于” Partition筛选”，默认为全选，如果需要对某个分区操作，可以选择” Partition筛选” 对应的修改按钮，没有分区则不用修改。

然后，在”输出表名”处，填写预测输出表如naive_bayes_predict_output，点击”训练”按钮开始训练，如下图：



最后，预测成功，会自动打开输出表，如下图：

The screenshot shows a software interface for data analysis. The main window title is "判别分析-预测结果:naive_bayes_predict_output". The menu bar includes "表格" (Table), "数据处理" (Data Processing), "全表统计" (Full Table Statistics), "统计对比" (Statistical Comparison), "分析" (Analysis), "模型" (Model), and "工具" (Tools). Below the menu is a table with the header "conclusion". The data rows are:

	conclusion
0	male
1	male
2	male
3	male
4	female
5	female
6	female
7	female

At the bottom of the window, there are buttons for "Whole Table" (Whole Table), "位置" (Position), "数据大小: 8 行, 当前显示前 50 行 (最多10000行)" (Data Size: 8 rows, Current Display Before 50 Rows (Up to 10000 rows)), and "GO!".

8.6 贝叶斯判别(Bayes)

Bayes的统计思想总是假定对所研究的对象已有一定的认识, 常用先验概率分布来描述这种认识。然后我们抽取一个样本, 用样本来修正已有的认识(先验概率分布), 得到后验概率分布。各种统计推断都通过后验概率分布来进行, 将贝叶斯思想用于判别分析就得到贝叶斯判别法。

在正态总体的假设下, 按Bayes判别的思想, 在错判造成的损失认为相等情况下得到的判别函数其实就是马氏距离判别在考虑先验概率及协差阵不等情况下的推广。

所谓判别方法, 就是给出空间 R_m 的一种划分: $D = \{D_1, D_2, \dots, D_k\}$ 。一种划分对应一种判别方法, 不同的划分就是不同的判别方法。Bayes判别法也是给出空间 R_m 的一种划分。

为便于说明, 先约定如下: * 训练输入表: train_input * 模型表: bayes_model * 预测输入表: predict_input * 预测输出表: bayes_predict_output * 特征列名: ["height", "weight", "footsize"] * Label列名: "sex"

贝叶斯模型的执行途径有两种: 函数和界面。其主要涉及两个过程: 训练和预测, 详见下文。

8.6.1 函数

贝叶斯判别包括以下几个函数: train , predict和loadModel。

三个函数具体使用方法, 可用help命令查看, 示例:

```
help(Classification.BayesDiscriminant.train)
help(Classification.BayesDiscriminant.predict)
help(Classification.BayesDiscriminant.loadModel)
```

训练

```
def train(inputTableName, featureColNames, labelColName, modelTableName, inputPartitions=None):
```

参数:

- inputTableName: 输入表名, 普通表
- featureColNames: 特征列名, 字符数组
- labelColName: label列名
- modelTableName: 模型表名, 普通表
- inputPartitions: (可选)输入表分区

返回:

- BayesModel 贝叶斯模型

示例:

```
Classification.BayesDiscriminant.train("train_input", ["height", "weight", "footsize"], "sex", "bayes_model")
```

预测

```
def predict(inputTableName, model, outputTableName, inputPartitions=None, appendColNames=None, outputPartition=None):
```

参数:

- inputTableName: 输入表名, 普通表
- model: 模型, 类型为贝叶斯BayesModel
- outputTableName: 输出表名, 普通表
- inputPartitions: (可选)输入表分区
- appendColNames: (可选)输入表中插入到输出表中的列名
- outputPartition: (可选)输出表分区

返回:

- void 无

示例:

```
bayesmodel=Classification.BayesDiscriminant.train("train_input", ["height", "weight", "footsize"], "sex", "bayes_model")
Classification.BayesDiscriminant.predict("predict_input", bayesmodel, "bayes_model_predict_out")
```

加载模型

```
def loadModel(modelTableName):
```

参数:

- modelTableName: 模型表表名, 普通表

返回:

- BayesModel 贝叶斯模型

示例:

```
Classification.BayesDiscriminant.loadModel("bayes_model")
```

注意事项:

- 该分类算法, 对于训练和预测的输入输出, 仅支持普通表(table);
- 特征值(feature)仅支持数值类型: Double, Bigint, label值支持Double, Bigint, Boolean, String;
- 对于预测输出结果, 可以选择将预测输入的若干列附加到预测输出表中。

8.6.2 界面

界面包括: 训练和预测。

训练

- 输入: 普通表, 包含label列和数值类型feature列的表, 如train_input。
- 输出: 模型表, 包括预测用到的一些参数, 如bayes_model。

训练步骤如下:

首先, 打开训练输入表train_input, 然后点击”模型” — ”贝叶斯(Bayes)” – “训练”, 如下图:



然后，在训练窗口，在对应提示区域填写相应参数，” Partition筛选”，”选择feature列”，”Label列”，”输出表表名”，点击界面右下角按钮”训练”开始训练，如下图：



其中：

- 对于” Partition筛选”，默认为全选，如果需要对某个分区操作，可以选择” Partition筛选” 对应的” 修改” 按钮，没有分区则不用修改。
- 对于” 选择feature列”，点击” 修改” 后，候选列表均为数值类型，用户可根据需要选择feature，如：train_input的特征列为[” height”， “weight”， ” footsize”]。
- 对于” Label列” 选择，不能跟feature列有交集，如：train_input的目标列(即label列)为sex。
- 对于” 输出表表名”，如：bayes_model。

最后，模型训练成功，自动打开model表bayes_model，如下图：

贝叶斯模型表:bayes_model

	vallabel	restlabel	namefeature	mean	inv0	inv1	inv2
0	female	-0.9842954...	height	5.4175	191.003460...	-0.8470588...	-30.103806...
1	female	-0.9842954...	weight	132.5	-0.8470588...	0.01680000...	-0.0882352...
2	female	-0.9842954...	footsize	7.5	-30.103806...	-0.0882352...	9.11418685...
3	male	-2.1858431...	height	5.855	46.8262226...	-0.4183142...	4.65140478...
4	male	-2.1858431...	weight	176.25	-0.4183142...	0.01253694...	-0.0695525...
5	male	-2.1858431...	footsize	11.25	4.65140478...	-0.0695525...	1.64203954...
6			{"version":...				

Whole Table ▾ 位置 数据大小: 7 行, 当前显示前 50 行 (最多10000行) GO!

贝叶斯模型表包括两种信息：conf和data信息

- conf信息包括：version, featureSize和 featureList
- data信息包括以下数据：
 - valLabel: label的值
 - restLabel: label的差值: $2 * \text{Math.log}(p) - \text{Math.log}(\det(\text{COV}))$
 - nameFeature: 特征名
 - mean: feature相对于label的均值
 - inv: feature的协方差的逆

预测

- 输入：包括输入表和模型表，输入表，如predict_input，模型表，如bayes_model。
- 输出：预测结果输出表，如bayes_predict_output。输出表一般只有一列conclusion，也可以append 输入表中的若干列列在输出表中

预测步骤如下：

首先, 可打开”输入表”或者”模型表”

- 如果打开的是”输入表” predict_input, 点击”模型” —”贝叶斯(Bayes)” —”预测”, 预测表表名会被自动加载, 用户可在” Partition筛选” 区域选择修改Partition, 在” Model表名” 处填写模型表名, 如下图:



- 如果打开的是”模型表” bayes_model, 点击”模型” —”贝叶斯(Bayes)” —”预测”, Model表名会被自动加载, 用户需在”预测表”区域, 填写表名, 点击”加载Partition”后, 可进行”Partition筛选”如下图:



- 对于” Partition筛选”，默认为全选，如果需要对某个分区操作，可以选择” Partition筛选” 对应的修改按钮，没有分区则不用修改。

然后，在” 输出表名”处，填写预测输出表如bayes_predict_output，点击” 训练” 按钮开始训练，如下图：



最后，预测成功，会自动打开输出表，如下图：

conclusion	
0	male
1	male
2	male
3	male
4	female
5	female
6	female
7	female

8.7 费希尔判别(Fisher)

费希尔判别的基本思想是投影(或降维)。

Fisher方法是要找到一个(或一组)投影轴w使得样本投影到该空间后能在保证方差最小的情况下, 将不同类的样本很好的分开。并将度量类别均值之间差别的量称为类间方差(或类间散布矩阵);而度量这些均值周围方差的量称为类内方差(或类内散布矩阵)。

Fisher判决的目标就是:寻找一个或一组投影轴, 能够在最小化类内散布的同时最大化类间布。

为便于说明, 先约定如下:

- 训练输入表: train_input
- 模型表: fisher_model
- 预测输入表: predict_input
- 预测输出表: fisher_predict_output
- 特征列名: ["height", "weight", "footsize"]

- Label列名: “sex”

费希尔判别模型的执行途径有两种: 函数和界面。其主要涉及两个过程: 训练和预测, 详见下文。

8.7.1 函数

费希尔判别包括以下几个函数: train , predict和loadModel。

三个函数具体使用方法, 可用help命令查看, 示例:

```
help(Classification.FisherDiscriminant.train)
help(Classification.FisherDiscriminant.predict)
help(Classification.FisherDiscriminant.loadModel)
```

训练

```
def train(inputTableName, featureColNames, labelColName, modelTableName, inputPartitions=None):
```

参数:

- inputTableName: 输入表名, 普通表
- featureColNames: 特征列名, 字符数组
- labelColName: label列名
- modelTableName: 模型表名, 普通表
- inputPartitions: (可选)输入表分区

返回:

- FisherDiscriminantModel 费希尔判别模型

示例:

```
Classification.FisherDiscriminant.train("train_input", ["height", "weight", "footsize"], "sex", "fisher_model")
```

预测

```
def predict(inputTableName, model, outputTableName, inputPartitions=None, appendColNames=None,
           outputPartition=None):
```

参数:

- inputTableName: 输入表名, 普通表
- model: 模型, 类型为费希尔判别模型FisherDiscriminantModel
- outputTableName: 输出表名, 普通表

- inputPartitions: (可选)输入表分区
- appendColNames: (可选)输入表中插入到输出表中的列名
- outputPartition: (可选)输出表分区

返回:

- void 无

示例:

```
fishermodel=Classification.FisherDiscriminant.train("train_input", ["height", "weight", "footsize"],  
"sex", "fisher_model")  
  
Classification.FisherDiscriminant.predict("predict_input", fishermodel, "fisher_model_predict_out")
```

加载模型

```
def loadModel(modelTableName):
```

参数:

- modelTableName: 模型表表名, 普通表

返回:

- FisherDiscriminantModel 费希尔判别模型

示例:

```
Classification.FisherDiscriminant.loadModel("fisher_model")
```

注意事项:

- 该分类算法, 对于训练和预测的输入输出, 仅支持普通表(table);
- 特征值(feature)仅支持数值类型: Double, Bigint, label值支持Double, Bigint, Boolean, String;
- 对于预测输出结果, 可以选择将预测输入的若干列附加到预测输出表中。

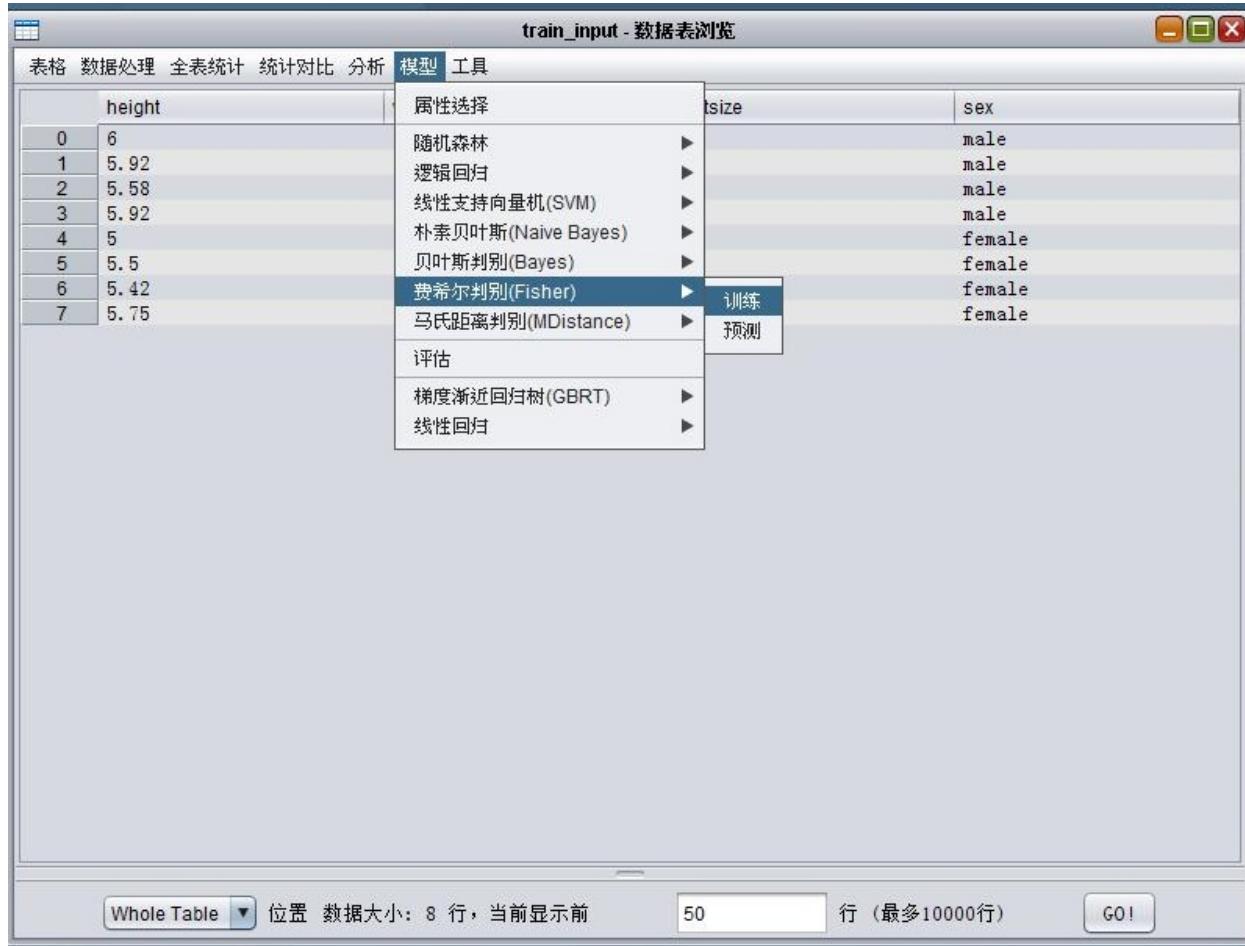
8.7.2 界面

界面包括: 训练和预测。

训练

- 输入: 普通表, 包含label列和数值类型feature列的表, 如train_input。
- 输出: 模型表, 包括预测用到的一些参数, 如fisher_model。

训练步骤如下：首先，打开训练输入表train_input，然后点击”模型”—”费希尔判别(Fisher)”—“训练”，如下图：



然后，在训练窗口，在对应提示区域填写相应参数，“Partition筛选”，“选择feature列”，“Label列”，“输出表表名”，最后点击界面右下角按钮”训练”开始训练，如下图：



其中：

- 对于” Partition筛选 ”，默认为全选，如果需要对某个分区操作，可以选择” Partition筛选 ” 对应的修改按钮，没有分区则不用修改。
- 对于” 选择feature列 ”，点击” 修改 ” 后，候选列表均为数值类型，用户可根据需要选择feature，如： train_input 的特征列为 [” height ”， “weight ”， ” footsize ”]。
- 对于” Label列 ” 选择，不能跟feature列有交集，如： train_input 的目标列(即label列)为 sex。
- 对于” 输出表表名 ”，如： fisher_model。

最后，模型训练成功，自动打开model表fisher_model，如下图：

费希尔模型表:fisher_model					
表格 数据处理 全表统计 统计对比 分析 模型 工具					
vallabel	namefeature	c0	c1	c2	
0	female				
1	male				
2	height				
3	weight				
4	footsize				
5		-0.0882561810114...	-6.5576394960746...	0.5028498536589378	
6		5.63625	154.375	9.375	
7	{"version": "v1.0..."}				

费希尔判别模型表包括两种信息，conf和data信息：

- conf信息包括：version, featureSize和 featureList
- data信息包括以下数据：
- valLabel: lable的值
- nameFeature: 特征名
- c: 系数和均值

预测

- 输入：包括输入表和模型表，输入表，如predict_input，模型表，如fisher_model。
- 输出：预测结果输出表，如fisher_predict_output。输出表一般只有一列conclusion，也可以append输入表中的若干列列在输出表中

预测步骤如下：首先，可打开”输入表”或者”模型表”

- 如果打开的是”输入表”predict_input，点击”模型”—”费希尔判别(Fisher)”—”预测”，预测表表名会被自动加载，用户可在”Partition筛选”区域选择修改Partition，在”Model表名”处填

写模型表名, 如下图:



- 如果打开的是”模型表”fisher_model, 点击”模型” — ”费希尔判别(Fisher)” — ”预测”, Model表名会被自动加载, 用户需在”预测表”区域, 填写表名, 点击”加载Partition”后, 可进行”Partition筛选”如下图:



- 对于” Partition筛选 ”，默认为全选，如果需要对某个分区操作，可以选择” Partition筛选 ” 对应的修改按钮，没有分区则不用修改。



最后, 预测成功, 会自动打开输出表, 如下图:

conclusion
0 male
1 male
2 male
3 male
4 female
5 female
6 female
7 female

8.8 马氏距离判别(MDistance)

因为同一总体的样品应具有相似性或者说距离较小，不同总体的样品距离较大或者说相似性较小，由此可以看出，我们可以用距离统计量作为定量识别或者说归类的依据。该模型是用马氏距离Mahalanobis)进行类别判断。

为便于说明，先约定如下：

- 训练输入表： train_input
- 模型表： mdist_model
- 预测输入表： predict_input
- 预测输出表： mdist_predict_output
- 特征列名： [”height”， “weight”， ”footsize”]
- Label列名： “sex”

马氏距离判别模型的执行途径有两种：函数和界面。其主要涉及两个过程：训练和预测，详见下文。

8.8.1 函数

马氏距离判别包括以下几个函数: train , predict和loadModel。

三个函数具体使用方法, 可用help命令查看, 示例:

```
help(Classification.MDistanceDiscriminant.train)
help(Classification.MDistanceDiscriminant.predict)
help(Classification.MDistanceDiscriminant.loadModel)
```

训练

```
def train(inputTableName, featureColNames, labelColName, modelTableName, inputPartitions, cost, epsilon):
```

参数:

- inputTableName: 输入表名, 普通表
- featureColNames: 特征列名, 字符数组
- labelColName: label列名
- modelTableName: 模型表名, 普通表
- inputPartitions: (可选)输入表分区
- cost: (可选)惩罚因子, 默认值为1
- epsilon: (可选)收敛系数, 默认值为0.001

返回:

- MDistanceDiscriminantModel 马氏距离模型

示例:

```
Classification.MDistanceDiscriminant.train("train_input", ["height", "weight", "footsize"], "sex", "mdist_model")
```

预测

```
def predict(inputTableName, model, outputTableName, inputPartitions=None,
appendColNames=None, outputPartition=None):
```

参数:

- inputTableName: 输入表名, 普通表
- model: 模型, 类型为马氏距离模型MDistanceDiscriminantModel
- outputTableName: 输出表名, 普通表
- inputPartitions: (可选)输入表分区

- appendColNames: (可选)输入表中插入到输出表中的列名
- outputPartition: (可选)输出表分区

返回:

- void 无

示例:

```
mdistmodel=Classification.MDistanceDiscriminant.train("train_input", ["height", "weight", "footsize"], "sex", "mdist_model")
Classification.MDistanceDiscriminant.predict("predict_input", mdistmodel, "mdist_model_predict_out")
```

加载模型

```
def loadModel(modelTableName):
```

参数:

- modelTableName: 模型表表名, 普通表

返回:

- MDistanceDiscriminantModel 马氏距离模型

示例:

```
Classification.MDistanceDiscriminant.loadModel("mdist_model")
```

注意事项:

- 该分类算法, 对于训练和预测的输入输出, 仅支持普通表(table);
- 特征值(feature)仅支持数值类型: Double, Bigint, label值支持Double, Bigint, Boolean, String;
- 对于预测输出结果, 可以选择将预测输入的若干列附加到预测输出表中。

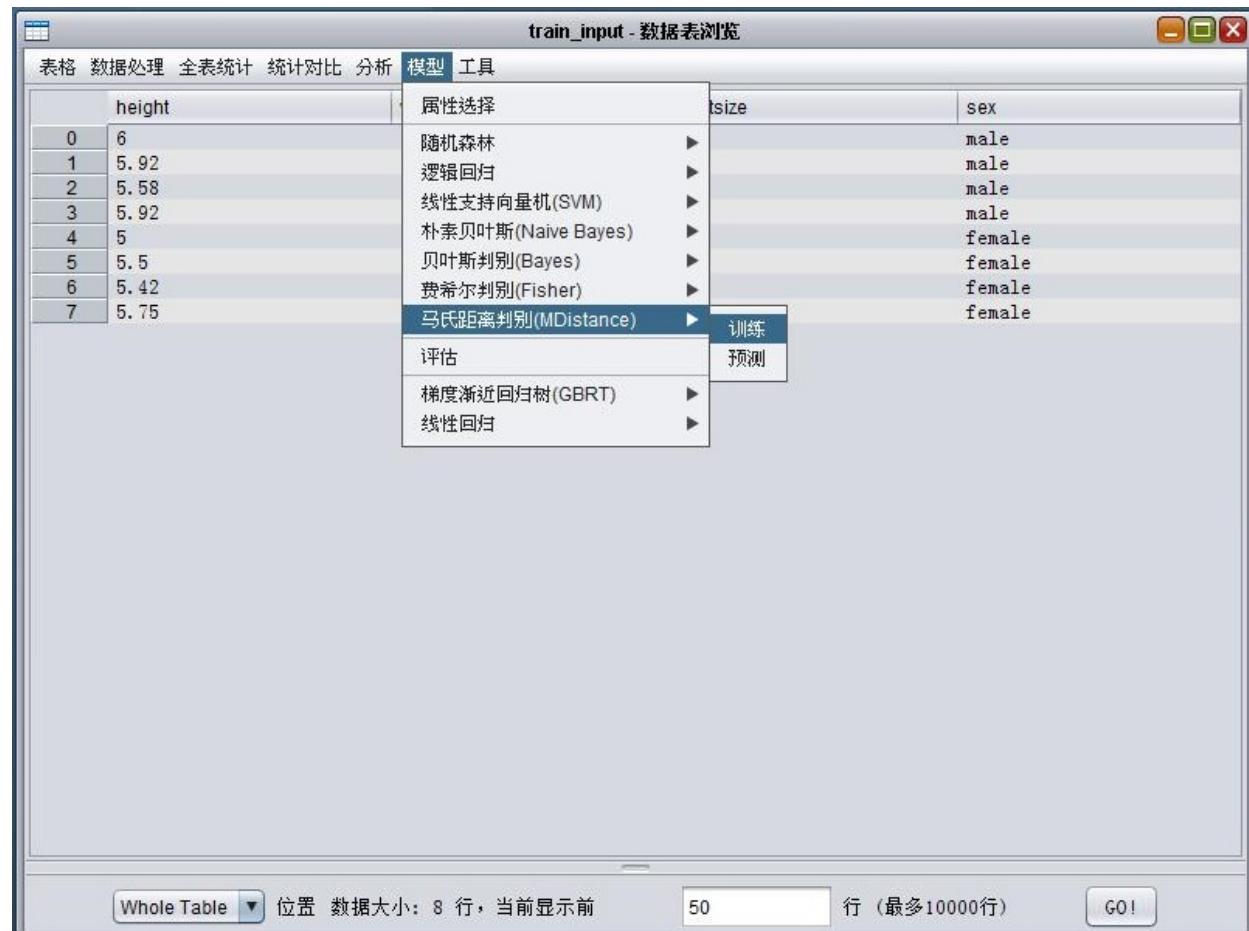
8.8.2 界面

界面包括: 训练和预测。

训练

- 输入: 普通表, 包含label列和数值类型feature列的表, 如train_input。
- 输出: 模型表, 包括预测用到的一些参数, 如mdist_model。

训练步骤如下：首先，打开训练输入表train_input，然后点击”模型” — ”马氏距离判别(MDistance)”—“训练”，如下图：



然后，在训练窗口，在对应提示区域填写相应参数，” Partition筛选”，”选择feature列”，”Label列”，”输出表表名”，最后点击界面右下角按钮”训练”开始训练，如下图：



其中：

- 对于” Partition筛选”，默认为全选，如果需要对某个分区操作，可以选择” Partition筛选” 对应的修改按钮，没有分区则不用修改。
- 对于” 选择feature列”，点击” 修改” 后，候选列表均为数值类型，用户可根据需要选择feature，如：train_input的特征列为[” height”， “weight”， ” footsize”]。
- 对于” Label列” 选择，不能跟feature列有交集，如： train_input的目标列(即label列)为sex。
- 对于” 输出表表名”，如： mdist_model。

最后，模型训练成功，自动打开model表mdist_model，如下图：

马氏距离模型表:mdist_model

	vallabel	namefeature	mean	inv0	inv1	inv2
0	female	height	5.4175	191.00346020...	-0.8470588235...	-30.10380622...
1	female	weight	132.5	-0.847058823...	0.0168000000...	-0.088235294...
2	female	footsize	7.5	-30.10380622...	-0.0882352941...	9.1141868512141
3	male	height	5.855	46.826222684...	-0.4183142559...	4.6514047866...
4	male	weight	176.25	-0.418314255...	0.01253694068...	-0.069552549...
5	male	footsize	11.25	4.6514047866...	-0.0695525494...	1.6420395421...
6	{ "version": "v..." }					

Whole Table ▾ 位置 数据大小: 7 行, 当前显示前 50 行 (最多10000行) GO!

马氏距离判别模型表包括两种信息: conf和data信息

- conf信息包括: version, featureSize和 featureList
- data信息包括以下数据:
 - valLabel: label的值
 - pLabel: label的概率
 - nameFeature: 特征名
 - mean: feature相对于label的均值
 - inv: feature之间的协方差的逆

预测

- 输入: 包括输入表和模型表, 输入表, 如predict_input, 模型表, 如mdist_model。
- 输出: 预测结果输出表, 如mdist_predict_output。输出表一般只有一列conclusion, 也可以append 输入表中的若干列列在输出表中。

预测步骤如下:

首先, 可打开”输入表”或者”模型表”

- 如果打开的是”输入表” predict_input, 点击”模型” —” 马氏距离判别(MDistance) ” —” 预测”, 预测表表名会被自动加载, 用户可在” Partition筛选” 区域选择修改Partition, 在” Model 表名” 处填写模型表名, 如下图:



- 如果打开的是”模型表” mdist_model, 点击”模型” —” 马氏距离判别(MDistance) ” —” 预测”, Model表名会被自动加载, 用户需在”预测表”区域, 填写表名, 点击”加载Partition”后, 可进行” Partition筛选”如下图:



- 对于” Partition筛选”，默认为全选，如果需要对某个分区操作，可以选择” Partition筛选” 对应的修改按钮，没有分区则不用修改。

然后，在” 输出表名”处，填写预测输出表如mdist_predict_output，点击” 训练” 按钮开始训练，如下图：



最后, 预测成功, 会自动打开输出表, 如下图:

conclusion	
0	male
1	male
2	male
3	male
4	female
5	female
6	female
7	female

8.9 决策树C5.0

C5.0是经典的决策树模型算法之一，采用Boosting方式提高模型准确率，又称为BoostingTrees。C5.0可生成多分支的决策树，目标变量为分类变量，使用C5.0算法可以生成决策树或者规则集。C5.0模型根据能偶带来的最大信息增益的字段拆分样本。第一次拆分确定的样本子集随后再次拆分，通常是根据另一个字段进行拆分，这一过程重复进行指导样本子集不能在被拆分为止。最后，重新从最低层次的拆分，哪些对模型值没有显著贡献的样本子集被提出或者修剪。

- C5.0算法介绍： C5.0 。

8.9.1 函数

C5.0模型包括以下几个函数：train, predict, isC5Model, loadModel和exportModel。

这几个函数具体使用方法，可用help命令查看，示例：

```
help(Classification.C5.train)
help(Classification.C5.predict)
```

```
help(Classification.C5.isC5Model)
help(Classification.C5.loadModel)
help(Classification.C5.exportModel)
```

训练

```
def train(inputTableName,
          modelTableName,
          featureColNames=None,
          isFeatureContinuous=None,
          labelColName=None,
          nameConfTableName=None,
          nameConfPartition=None,
          inputPartitions = None,
          isRule = False,
          utility = -1,
          isWinnow = False,
          boostTrials = -1,
          isProbThresh = False,
          isSubset = False,
          isGlobPruning = True,
          minSplits = -1,
          cf = -1.0,
          sample = -1.0,
          folds = -1,
          isSetRandomSeed = False,
          randomSeed = -1,
          testTableName = None,
          testPartitions = None,
          costTableName = None,
          costPartitions = None,
          costColNames = None
) :
```

参数:

- inputTableName: 训练输入表的表名。
- modelTableName: 输出的模型名。
- featureColNames: (可选) 输入表中选择的用于训练的特征列名。
- isFeatureContinuous: (可选) 特征对应的类型, 表示对应特征是否为连续值。True为连续, False为离散。[True, False, False]表示
 - 三列特征中第一列为连续, 第二和第三列为离散。

- labelColName: (可选) 输入表中标签列的列名。
- featureColNames, isFeatureContinuous, labelColName 如果输入这三个参数, 系统会自动生成下面的name配置文件,
- nameConfTableName: (可选) name配置输入表名, 会读取nameConf第一行cell的文本, 其格式如下:

```

class. | the target variable
       #第一行是对目标列的描述, 即指定目标列的列名
age :continuous.
       #对feature的描述, 名字是age, 其是连续类型
workclass :Federal-gov, Local-gov, Never-worked
       #对feature的描述, 名字是workclass, 是discrete类型, 一个有三个分类值, 以逗号分隔
class :0, 1.
       #对目标列的描述, 是分类类型, 一个有两个值, 分别是0和1

```

注意事项:

- 第一行必须执行目标列的列名, 格式是 XXX | the target variable, XXX是目标列名
- feature以及描述, 用冒号: 分隔, 其中continuous表示是连续属性, 其列类型必须是double, long或者boolean; 非连续类型, 需要列举其所有分类值, 并以其逗号划分, 类型是string, long, boolean
- 如何属性值有一下字符(逗号, 冒号, 点, 空白字符, 反斜杠)需要转义, 比如属性有个分类值是”Filch, Grabbit, and Co.” , 需要写成”Filch\, Grabbit\, and Co\.”
- namesconf大小不可以超过1.5M
- namesconf不支持字符 ‘\0’
- nameConfPartition: (可选) names配置输入partition
- inputPartitions: (可选) 输入表对应的输入分区, 选中全表则为None。
- isRule : (可选) use rule-based classifiers, 默认False。
- utility : (可选) order rules by utility in bands, 默认-1, 不设置, 范围[2, 10000]
- isWinnow : (可选) invoke attribute winnowing, 默认False
- boostTrials : (可选) number of boosting trials, 默认-1, 不指定, 范围[3, 1000]
- isProbThresh : (可选) use soft thresholds, 默认False
- isSubset : (可选) find subset tests for discrete attrs, 默认False
- isGlobPruning : (可选) use global tree pruning, 默认True
- minSplits : (可选) 最小可分数据大小, 默认-1, 不限制, 范围[1, 1000000]

- cf = : (可选) confidence level (CF) for pruning, 值越小剪枝越多, 默认-1.0, 不设置 (按默认25.0剪枝), 范围[0.0, 100.0]
- sample = : (可选) 训练数据百分比, 默认-1.0, 不设置, 全集用来训练, 范围[0.1, 99.9]
- folds = : (可选) 交叉验证cross-validate, 如果设置该属性, 则输出的模型只显示信息不能做预测, 默认 -1, 不交叉验证, 范围[2, 100]
- isSetRandomSeed : (可选) 是否设置随机种子, 默认False, 不设置
- randomSeed : (可选) 采样和交叉验证的随机种子, 在isSetRandomSeed = True, 生效
- testTableName : (可选) 测试输入表, 默认None, 如果不为None, 则需要和inputTableName一样的表结构
- testPartitions : (可选) 测试输入表对应的输入分区, 选中全表则为None。
- costTableName : (可选) cost 输入表, 默认None, 需要3列, 分别为predict_class, true_class, cost
- costPartitions : (可选) cost输入表对应的输入分区, 选中全表则为None。
- costColNames : (可选) cost输入表选择的列名, 分别对应predict_class, true_class, cost

返回:

- C5Model类型, 表示C5的模型。

示例:

- 输入featureColNames, isFeatureContinuous, labelColumnName

```
Classification.C5.train("weka_contact_lenses", "weka_contact_lenses_c5_model", \
    featureColNames=["age", "spectacle_prescrip", "astigmatism", "tear_prod_rate"], \
    isFeatureContinuous=[False, False, False, False], \
    labelColumnName="contact_lenses", \
    inputPartitions=["pdate=201315", "pdate=201405"], isRule=False, utility=3, isWinnow = False, \
    boostTrials=10, isProbThresh=False, isSubset=False, isGlobPruning=True, minSplits=100, \
    cf=25.0, sample = 90.0, isSetRandomSeed=True, randomSeed=10)
```

- 输入nameConf表

```
Classification.C5.train("weka_contact_lenses", "weka_contact_lenses_c5_model", \
    nameConfTableName="weka_contact_lenses_names", nameConfPartition="20140506", \
    inputPartitions=["pdate=201315", "pdate=201405"], isRule=False, utility=3, isWinnow = False, \
    boostTrials=10, isProbThresh=False, isSubset=False, isGlobPruning=True, minSplits=100, \
    cf=25.0, sample = 90.0, isSetRandomSeed=True, randomSeed=10)
```

注意事项:

- 输入数据中, Feature属性列, 存在一下三个值会被当成缺失值: 问好(‘?’), 空串(‘’), 缺失值

- 输入数据中， 目标列，如果存在：问好(‘?’)，空串(‘’)或缺失值，则这条记录会被忽略
- 如果设置folds参数，则输出的模型不能做预测

预测

```
def predict(inputTableName,
            model,
            outputTableName,
            inputPartitions = None,
            appendColNames = None,
            outputPartition = None,
            labelValueToPredict = None):
```

参数：

- inputTableName: 预测输入表名。
- model: C5Model类型。C5模型
- outputTableName: 预测输出表。
- inputPartitions: (可选)。输入表对应的输入分区，选中全表则为None。
- appendColNames: (可选)。输出表中需要增加的预测输入表的列。
- outputPartition: (可选)。指定输出到输出表的分区。
- labelValueToPredict: (可选)。目标是二分时，输出的概率对应的分类值。比如有两个分类：good, bad。如果想输出good的概率，则labelValueToPredict=' good'

示例：

```
c5Model = Classification.C5.loadModel("adult_c5_model")
Classification.C5.predict("adult", c5Model, "out_table_name",
    inputPartitions = ['p=1'], appendColNames = ['label'], outputPartition = "p=1",
    labelValueToPredict = ' good')
```

注意事项：

- C5.0预测根据输入表和模型，输出预测结果。
- 本方法在二分类情况下，可以有概率输出。在多分类情况下，无概率输出。在二分有概率输出的情况下，输出两列，第一列为预测的结论，第二列为labelValueToPredict 所指定的分类对应的概率。

是否为C5.0模型

```
def isC5Model(modelTableName):
```

参数:

- modelTableName: 输入模型表名,

返回:

- True|False: 输出是否C5.0模型

示例:

```
isModel=Classification.C5.isC5Model("adult_model")
```

加载模型

```
def loadModel(modelTableName):
```

参数:

- modelTableName: 输入模型表名

返回:

- C5Model: 输出C5模型

示例:

```
c5Model=Classification.C5.loadModel("adult_model")
```

导出模型

```
def exportModel(model, outputFolderPath):
```

参数:

- model: 输入C5模型
- outputFolderPath: 输出模型的文件路径, 其路径下outputFolderPath/model+.names”, outputFolderPath/model+.tree” or outputFolderPath/model+.rules”, outputFolderPath/model+.out不存在

示例:

```
c5Model=Classification.C5.loadModel("weka_contact_lenses_c5_model");
Classification.C5.exportModel(c5Model,"D: \\ ")
```

注意事项:

- 导出C5模型目前支持小于1000M模型导出
- 一共输出三个文件, 其中.names 文件是对属性以及label的描述, .tree or .rules是对树的描述, .out文件是建树的过程以及评估内容

8.9.2 界面

使用训练集: adult

双击打开数据表, 模型菜单中选择决策树C5, 如:



- “打开模型”，如果当前表是C5模型，打开模型。
- “训练”，打开C5的训练界面
- “预测”，打开C5的预测界面
- “导出模型”，打开导出模型界面

训练

由菜单→决策树C5>训练, 进入训练界面如下:



训练界面一共有二个tab页，分别为：属性选择，参数配置。

- 属性选择tab页如上图，一共分为四个部分，
- 第一部分为属性选择(默认为除去最后一列的所有其他列)，显示属性的表一共三类，第一列表示当前属性是否选择，第二列是属性名称，第三列表示属性是否连续的，默认double和long会当成连续的。
- 第二部分为目标列选择(默认为最后一列)。
- 第三部分为模型输出表，用于存储模型数据，
- 第四部分为进度信息。
- 参数配置tab页如下图：

- 是否按规则集输出, 默认是false, 如果选中, 可以设置是否规则排序以及设置分组个数
- 是否boosting, 默认是false, 如果选中, 可以设置boost的次数
- 是否交叉验证, 默认是false, 如果选中, 可以设置交叉折数
- 是否对训练数据采样, 默认是false, 如果选中, 可以设置采样的比例。
- 是否设置随机种子, 默认false, 随机, 如果选中, 可以设置随机初始种子。采样和交叉验证会用到。
- 是否离散属性可分租, 默认false, 如果选中, 则对于分类属性, 同一条表可以有多个分类值。
- 是否Softening thresholds, 默认false。如果选中, 对于连续属性, 会生成两个分割点low, high, 左边 \leq low, 右边 \geq high, 对于 (low, high) 之间的数值会同时符合两条边。
- 测试表, 需要和训练表同样的表结构
- cost表, 需要3列, 分别为predict_class, true_class, cost
- 是否预先筛选属性, 默认false
- 是否全局剪枝, 默认true
- 是否设置最小可分数据, 默认false, 不限制, 如果选中, 可以设置最小可分数据大小
- 剪枝置信水平, 默认是25%。值越小, 剪枝越



参数配置完成以后，回到属性选择tab页，单击”开始训练”，界面如下：



训练结束后会自动显示训练的模型，后面会详细介绍模型：

C5 模型adult_c5_model

标准输出 模型 特征描述

```
C5.0 [Release 2.07 GPL Edition] Tue Sep 2 19:31:32 2014

Options:
    Rule-based classifiers
    Tests on discrete attribute groups
    Pruning confidence level 25%

Class specified by attribute 'class'

Read 32561 cases (15 attributes) from train table

Rules:

Rule 1: (430, lift 1.3)
    capital_gain > 594
    capital_gain <= 2993
    -> class 0 [0.998]

Rule 2: (309, lift 1.3)
    capital_gain > 3103
    capital_gain <= 4101
    -> class 0 [0.997]

Rule 3: (130, lift 1.3)
    relationship in {Husband, Wife}
    capital_loss > 1504
    capital_loss <= 1762
    -> class 0 [0.992]
```

预测

由菜单→C5→预测，进入预测界面如下：



其中：

- 预测输入表：要预测的表
- 结果附加列：可以选择预测输入表中需要输出到预测结果表的列，默认为无。
- 模型名：随机森林模型名
- 概率：如果目标列是二分的，且输入主分类，预测结果输出的是结论和输入分类的概率

- 输出表: 预测输出表

单击“预测”，预测完成后，即可打开预测输出表查看预测结果：



预测成功，可以单击表adult_c5_model_pre查看预测结果：

adult_c5_model_pre - 数据表浏览

conclusion		probability
0	0	0.001999974250793457
1	0	0.2070000171661377
2	0	0.2070000171661377
3	0	0.2070000171661377
4	0	0.2070000171661377
5	1	0.7149999737739563
6	0	0.2070000171661377
7	0	0.2070000171661377
8	1	0.9850000143051147
9	1	0.9909999966621399
10	1	0.7549999952316284
11	0	0.2070000171661377
12	0	0.2070000171661377
13	0	0.2070000171661377
14	0	0.2070000171661377
15	0	0.2070000171661377
16	0	0.2070000171661377
17	0	0.2070000171661377
18	0	0.2070000171661377
19	0	0.2070000171661377
20	1	0.7549999952316284
21	0	0.2070000171661377
22	0	0.2070000171661377
23	0	0.010999977588653564
24	0	0.2070000171661377
25	1	0.7850000262260437
26	0	0.2070000171661377
27	0	0.2070000171661377
28	0	0.2070000171661377
29	0	0.2070000171661377

Whole Table ▾ 位置 数据大小: -- 行, 当前显示前 50 行 (最多10000行) GO!

其中conclusion为预测的结果, probability为分类1的概率。

打开模型

双击模型表, 在菜单->决策树C5>打开模型, 进入界面如下:

C5 模型adult_c5_model

标准输出 模型 特征描述

C5.0 [Release 2.07 GPL Edition] Tue Sep 2 19:31:32 2014

Options:

- Rule-based classifiers
- Tests on discrete attribute groups
- Pruning confidence level 25%

Class specified by attribute 'class'

Read 32561 cases (15 attributes) from train table

Rules:

Rule 1: (430, lift 1.3)
capital_gain > 594
capital_gain <= 2993
-> class 0 [0.998]

Rule 2: (309, lift 1.3)
capital_gain > 3103
capital_gain <= 4101
-> class 0 [0.997]

Rule 3: (130, lift 1.3)
relationship in {Husband, Wife}
capital_loss > 1504
capital_loss <= 1762
-> class 0 [0.992]

Tab页，分为三种：

第一个是标准输出，输出的是树以及评估的结果，人可读，如图：

C5 模型adult_c5_model

标准输出 模型 特征描述

```
C5.0 [Release 2.07 GPL Edition] Tue Sep 2 19:31:32 2014
-----
Options:
    Rule-based classifiers
    Tests on discrete attribute groups
    Pruning confidence level 25%
Class specified by attribute 'class'
Read 32561 cases (15 attributes) from train table
Rules:
Rule 1: (430, lift 1.3)
    capital_gain > 594
    capital_gain <= 2993
    -> class 0 [0.998]
Rule 2: (309, lift 1.3)
    capital_gain > 3103
    capital_gain <= 4101
    -> class 0 [0.997]
Rule 3: (130, lift 1.3)
    relationship in {Husband, Wife}
    capital_loss > 1504
    capital_loss <= 1762
    -> class 0 [0.992]
```

第二个是模型，这个是真正的模型，机器可读，可用来做预测，如图：

C5 模型adult_c5_model

标准输出 模型 特征描述

```

id="See5/C5.0 2.07 GPL Edition 2014-09-02"
entries="1"
rules="42" default="0"
conds="2" cover="430" ok="430" lift="1.31414" class="0"
type="2" att="capital_gain" cut="594" result=">"
type="2" att="capital_gain" cut="2993" result="<"
conds="2" cover="309" ok="309" lift="1.31296" class="0"
type="2" att="capital_gain" cut="3103" result=">"
type="2" att="capital_gain" cut="4101" result="<"
conds="3" cover="130" ok="130" lift="1.30721" class="0"
type="3" att="relationship" elts="Husband","Wife"
type="2" att="capital_loss" cut="1504" result=">"
type="2" att="capital_loss" cut="1762" result="<"
conds="2" cover="88" ok="88" lift="1.30256" class="0"
type="2" att="capital_loss" cut="1980" result=">"
type="2" att="capital_loss" cut="2163" result="<"
conds="3" cover="82" ok="82" lift="1.30151" class="0"
type="3" att="relationship" elts="Husband","Wife"
type="2" att="capital_gain" cut="4416" result=">"
type="2" att="capital_gain" cut="5060" result="<"
conds="2" cover="34" ok="34" lift="1.2806" class="0"
type="2" att="capital_gain" cut="6514" result=">"
type="2" att="capital_gain" cut="6849" result="<"
conds="2" cover="503" ok="443" lift="1.15809" class="0"
type="2" att="capital_loss" cut="625" result=">"
type="2" att="capital_loss" cut="1762" result="<"
conds="1" cover="31162" ok="24700" lift="1.04402" class="0"
type="2" att="capital_gain" cut="6849" result="<"
conds="3" cover="109" ok="109" lift="4.11525" class="1"
type="3" att="relationship" elts="Husband","Wife"

```

第三个是特种描述，是对特征以及目标列的描述，如图：

```
class. |the target variable
age :continuous.
workclass :Federal-gov, Local-gov, Never-worked, Private, Self-emp-inc, Self-emp-not-inc, State-gov, Without-pay
fnlwgt :continuous.
education :10th, 11th, 12th, 1st-4th, 5th-6th, 7th-8th, 9th, Assoc-acdm, Assoc-voc, Bachelors, Doctorate, HS-grad, Masters
education_num :continuous.
marital_status :Divorced, Married-AF-spouse, Married-civ-spouse, Married-spouse-absent, Never-married, Separated,
occupation :Adm-clerical, Armed-Forces, Craft-repair, Exec-managerial, Farming-fishing, Handlers-cleaners, Machine-op-trainee
relationship :Husband, Not-in-family, Other-relative, Own-child, Unmarried, Wife
race :Amer-Indian-Eskimo, Asian-Pac-Islander, Black, Other, White
sex :Female, Male
capital_gain :continuous.
capital_loss :continuous.
hours_per_week :continuous.
native_country :Cambodia, Canada, China, Columbia, Cuba, Dominican-Republic, Ecuador, El-Salvador, England, France, Germany, Greece, Honduras, Ireland, Italy, Mexico, Portugal, Puerto-Rico, Spain, Switzerland, Taiwan, United-Kingdom, United States, Yugoslavia
class :0, 1.
```

导出模型

“导出模型”模块用于导出C5模型到特定格式的文本

选择模型数据表，右击表名选择→决策树C5.0→导出模型，如：



- 保存路径：输出模型的文件路径，其路径下outputFolderPath/model+” . names” , outputFolderPath/model+” . tree” or outputFolderPath/model+” . rules” , outputFolderPath/model+” . out” 不存在

单击“开始”，则开始导出模型，并显示导出的进度

8.10 梯度渐近树

梯度渐近树算法介绍：Gradient boosting算法介绍。本算法目前支持Gradient boosting decision tree。使用的损失函数是negative binomial log-likelihood，目前只支持二分类。

算法详细可参见：Greedy Function Approximation: A Gradient Boosting Machine Author(s): Jerome H. Friedman Source: The Annals of Statistics, Vol. 29, No. 5 (Oct., 2001), pp. 1189–1232 Published by: Institute of Mathematical Statistics

在XLab中，对于梯度渐近树模型，提供两种执行方式：函数和界面，主要包括两个过程：训练和预测。

8.10.1 函数

GBDT包括以下几个函数：trainSparse, predictSparse, loadModel和isModel。

这几个函数具体使用方法，可用help命令查看，示例：

```
help(Classification.GradBoostTree.trainSparse)
help(Classification.GradBoostTree.predictSparse)
```

```
help(Classification.GradBoostTree.loadModel)
help(Classification.GradBoostTree.isModel)
```

训练

```
def trainSparse(independentMatrixName, dependentTableName, dependentColName,
               modelTableName, dependentPartitions = None, treeDepth = 5,
               treesNum = 100, learningRate = 0.1, minSampleLeaf = 2,
               colTypes = None) :
```

参数:

- independentMatrixName: 输入的自变量(x)矩阵。
- dependentTableName: 输入的因变量(y)表名。
- dependentColName: 因变量的列名。
- modelTableName: 输出的模型名。
- dependentPartitions: (可选)因变量表对应的输入矩阵。选择全表则为None。默认为选择全表。
- treeDepth: (可选)单颗树的最大深度。默认为5.
- treesNum: (可选)树的棵数。默认为100.
- learningRate: (可选)学习速率。默认为0.1.
- minSampleLeaf: (可选)叶结点最小样本数。要求大于0, 默认为2。
- colTypes: (可选)列类型。True为连续型, False为离散型。默认None的情况下, 都为连续型。

返回:

- 返回GradBoostTreeModel类型, 表示梯度渐近树模型。

示例:

```
Classification.GradBoostTree.trainSparse("gbdt_train_sparse", "gbdt_train_dependent_table", "dependent",
                                         "gbdt_model", dependentPartitions = ["p=1"], treeDepth = 5, treesNum = 100,
                                         learningRate = 0.1, minSampleLeaf = 2, colTypes = [True, False])
```

注意事项:

- 本方法实现了梯度渐近树的训练过程。
- 自变量只支持 Xlib稀疏矩阵表 输入

预测

```
def predictSparse(inputMatrixName, model, outputTableName, outputPartition = None) :
```

参数:

- inputMatrixName: 预测输入矩阵名。
- model: GradBoostTreeModel类型。注意这里不是String类型, 见example。梯度渐近树模型。
- outputTableName: 预测输出表。
- outputPartition: (可选)指定输出到输出表的分区。None表示无分区。默认为无分区。

示例:

```
gbdtModel = Classification.GradBoostTree.trainSparse("gbdt_train_sparse", "gbdt_train_dependent_table", "dependent",
    "gbdt_model", dependentPartitions = ["p=1"], treeDepth = 5, treesNum = 100,
    learningRate = 0.1, minSampleLeaf = 2, colTypes = [True, False])
#如果训练和预测分开使用, 请使用Classification.GradBoostTree.loadModel(modelTableName)作为预测方法模型参数
Classification.GradBoostTree.predictSparse("gbdt_train_sparse", gbdtModel, "gbdt_prediction",
    outputPartition = "p=1", labelValueToPredict = "0")
```

注意事项:

- 本方法实现了梯度渐近树的预测过程。

加载模型

```
def loadModel(modelTableName) :
```

参数:

- modelTableName: 模型表名。

返回:

- GradBoostTreeModel: 梯度渐近树模型

示例:

```
gbdtModel = Classification.GradBoostTree.loadModel("table_name")
```

判断模型

```
def isModel(modelTableName) :
```

参数:

- modelTableName: 检查的模型表名。

返回:

- Boolean: 是否为梯度渐近树模型。

示例:

```
if Classification.GradBoostTree.isModel("table_name") :  
    print "gbdt model."
```

8.10.2 界面

在本例子中使用bank_full作为输入表。 bank_full的结构如下:

bank_full信息

数据列定义:

Name	Type	Comment
0 f0	double	
1 f1	double	
2 f2	double	
3 f3	double	
4 f4	double	
5 f5	double	
6 f6	double	
7 f7	double	
8 f8	double	

分区定义:

Name	Type	Comment

分区列表:

Name	RecordNumber

刷新行数 行数: 未知 打开

Project: projectxlib4 Owner: ALIYUN\$odpstest1@aliyun.c
生命周期: -1 Comment:
创建时间: 2013-12-31 11:21:32 修改时间: 2013-12-31 11:23:24

f0-f15为特征, label为目标

使用表转稀疏矩阵工具将f0-f15转为稀疏矩阵bank_full_sparce



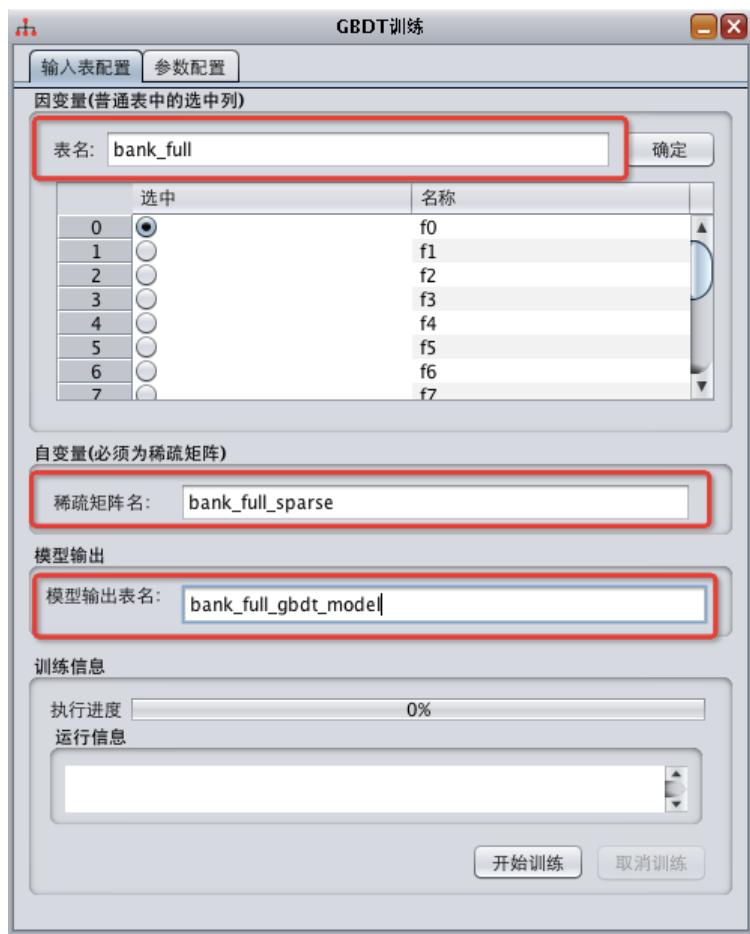
打开bank_full表，在菜单→模型→梯度渐近树，打开模型相关功能：

bank_full - 数据类浏览									
	f0	f1	f2	属性选择	f6	f7	f8	f9	
0	58	3	0	随机森林	1	0	0	5	
1	44	10	2	决策树C5	1	0	0	5	
2	33	5	0	梯度渐近树(GBDT)	1	1	0	5	
3	47	7	0	逻辑回归	0	0	0	5	
4	33	1	2	线性支持向量机(LinearSVM)	0	0	0	5	
5	35	3	0	非线性支持向量机(NonlinearSVM)	0	0	0	5	
6	28	3	2	朴素贝叶斯(Naive Bayes)	1	0	0	5	
7	42	5	1	贝叶斯判别(Bayes)	1	0	0	5	
8	58	9	0	费希尔判别(Fisher)	1	0	0	5	
9	43	10	2	马氏距离判别(MDistance)	1	0	0	5	
10	41	0	0	评估	1	0	0	5	
11	29	0	0	梯度渐近回归树(GBRT)	1	0	0	5	
12	53	10	0	线性回归	1	0	0	5	
13	58	10	0		1	0	0	5	
14	57	11	0		1	0	0	5	
15	51	9	0		1	0	0	5	
16	45	0	2		1	0	0	5	
17	57	7	0		1	0	0	5	
18	60	9	0		60	1	0	5	
19	33	11	0		0	1	0	5	
20	28	7	0		723	1	1	5	
21	56	3	0		779	1	0	5	
22	32	7	2		23	1	1	5	
23	25	11	0		50	1	0	5	
24	40	9	0		0	1	1	5	
25	44	0	0		-372	1	0	5	
26	39	3	2		255	1	0	5	
27	52	5	0		113	1	1	5	
28	46	2	2		246	1	0	5	

训练

在训练界面中的参数输入是和函数中的参数相对应的，参见[训练](#).

打开训练界面为：



在本界面中：

- 第一个红色方框中代表目标所在的表，其中选中的行代表目标在表中所在的列。
- 第二个红色方框中代表特征的 Xlib 稀疏矩阵表 名。
- 第三个红色方框中代表输出模型表名。

点击开始训练按钮，即开始训练。在本界面中有两个tab页，输入表配置页由上所述，参数配置页可以选择算法的参数，如下：



开始训练界面如下：



训练结束会弹出模型显示窗口，参见[模型显示](#)。

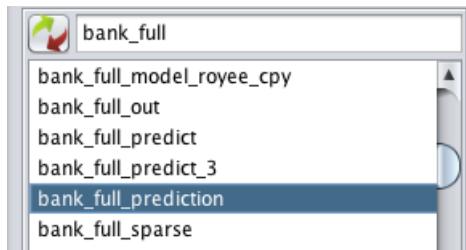
预测

在预测界面中的参数输入是和函数中的参数相对应的，参见[预测](#)。

打开预测界面为：



输入表为要预测的表，model为训练时的输出表，输出表为预测的结果。点击预测按钮，开始预测。在预测中，主界面上选择GBDT模型时，预测会自动识别这个表是否为GBDT模型。预测结束，选择预测结果表，如下图：

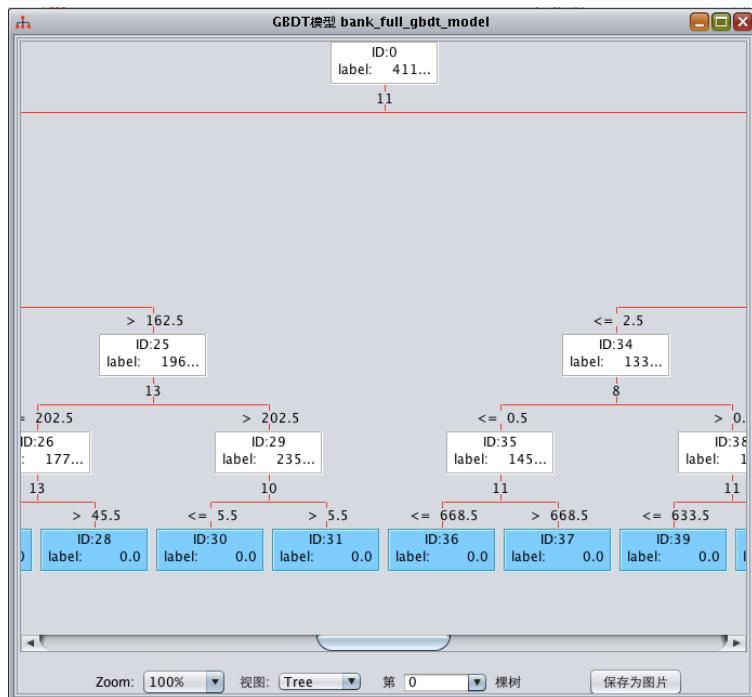


双击打开，查看预测结果，如下图：

bank_full_prediction - 数据表浏览		
conclusion	probability	
0	0.009031171924951198	
1	0.0048756564211969115	
2	0.0029548181077105103	
3	0.004642192051371856	
4	0.007176542479322161	
5	0.006209172462302292	
6	0.008275059084995918	
7	0.014832963474633296	
8	0.0026367948764018114	
9	0.0031704194371355195	
10	0.008754511768515646	
11	0.006374042519175588	
12	0.06052189521395393	
13	0.003343542525358373	
14	0.005384251078408278	
15	0.007297014396603868	
16	0.005157813820549573	
17	0.002487380231858754	
18	0.0060154815364539494	
19	0.00279052203318781	
20	0.006363951270878584	
21	0.00655630803466011	
22	0.00426586606298459	
23	0.008002245183729809	
24	0.004087859225969351	
25	0.0055043541131044845	
26	0.009635989742164508	
27	0.004641235530693879	
28	0.005859640716214959	
...	...	

模型显示

对于训练出来的模型，可以通过XLab提供的树的图形表示来展示，如下图：



模型显示提供了以下功能：

- 放大缩小；
- tree, folder, print三种视图；
- 选择树的编号。

8.11 评估

用于对分类算法进行评估，可以计算正确率，混淆矩阵，ROC，Lift，Precision/Recall。

8.11.1 界面

- 选择数据表(任意表均可)，双击打开，菜单模型->评估如下图：



- 评估界面如下图：



评估模块是用来计算混淆矩阵和画图的。 页面一共有四部分，分别为：

- 预测原始表，即预测的输入表，单击”加载列信息”之后，可以选择目标列和权重为列，如下图：



- 预测结果表, 即预测的输出表, 单击”加载列信息”之后, 可以选择目标列和概率列, 以及填写概率列对应的目标分类值, 即是谁的概率, 如下图:

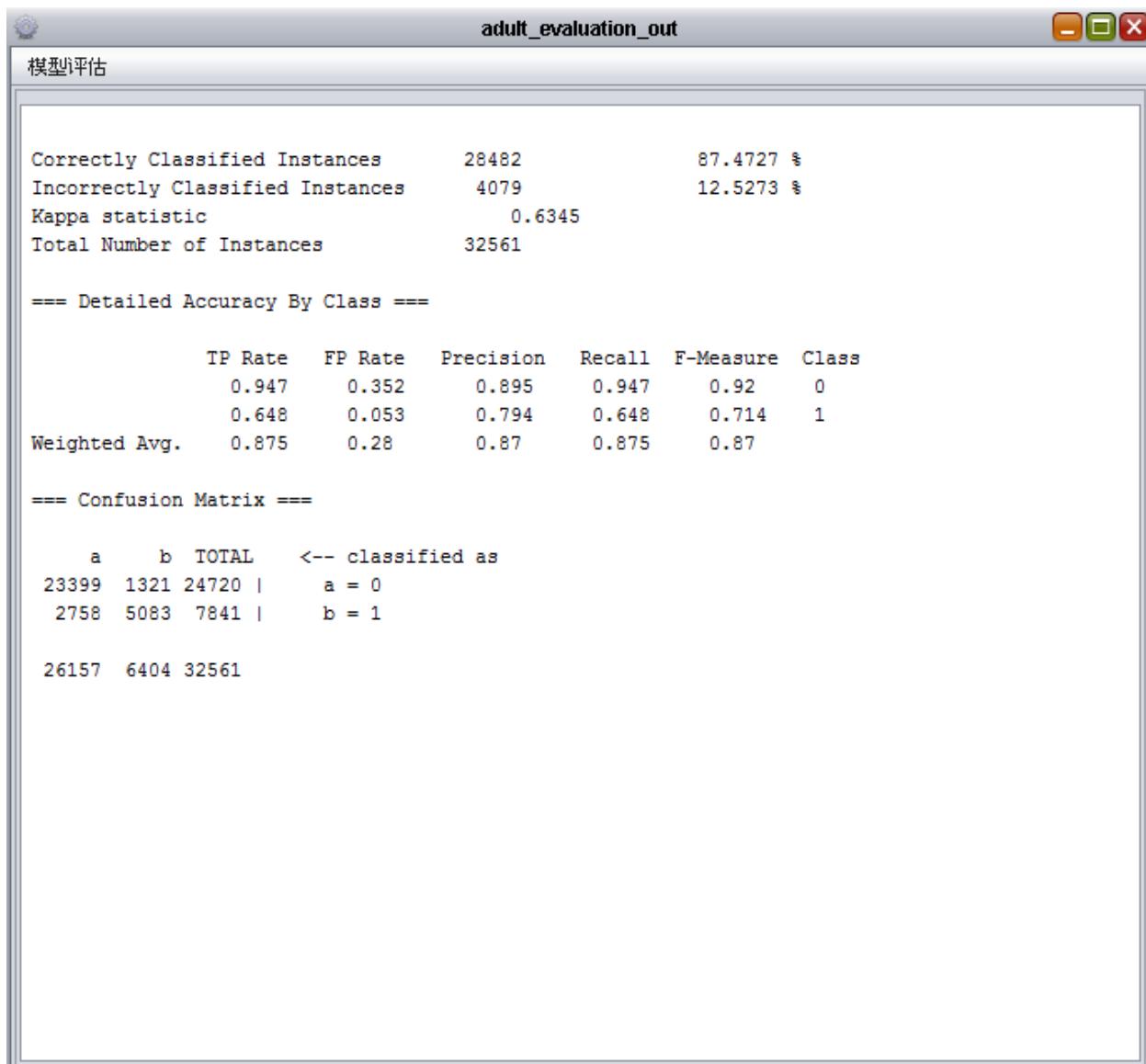


- 混淆矩阵输出表。
- 概率评估输出表, 如果目标是二分且选择概率列, 则可以计算概率评估, 包括有阈值混淆矩阵, ROC, lift and Recall-FP。

单击”开始评估”, 如下图:



- 评估结束后会自动弹出结果, 如下图:



The screenshot shows a window titled "adult_evaluation_out" containing the output of a model evaluation script. The output includes:

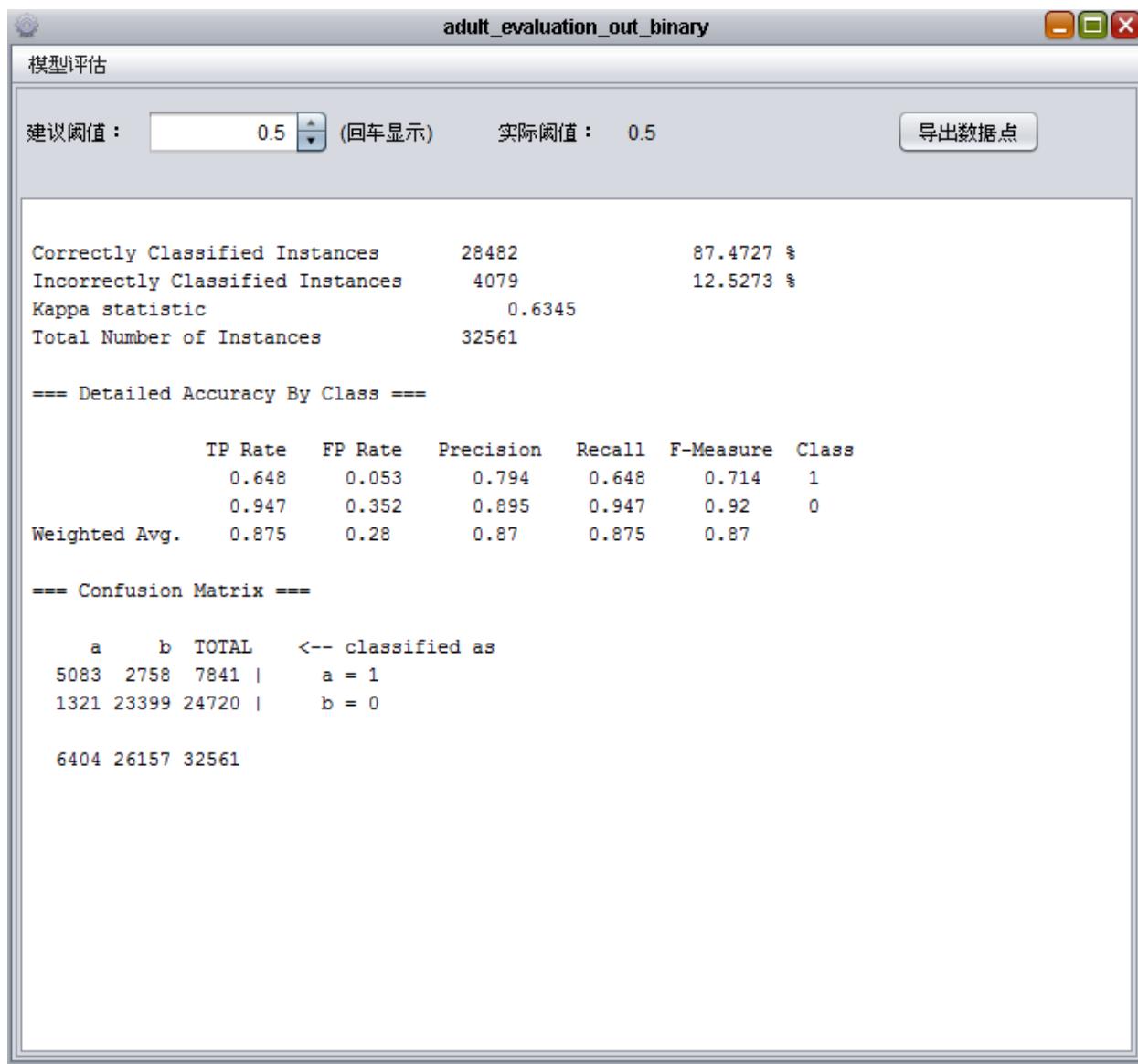
- Summary statistics:
 - Correctly Classified Instances: 28482 (87.4727 %)
 - Incorrectly Classified Instances: 4079 (12.5273 %)
 - Kappa statistic: 0.6345
 - Total Number of Instances: 32561
- Detailed Accuracy By Class:

	TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0	0.947	0.352	0.895	0.947	0.92	0
1	0.648	0.053	0.794	0.648	0.714	1
Weighted Avg.	0.875	0.28	0.87	0.875	0.87	
- Confusion Matrix:

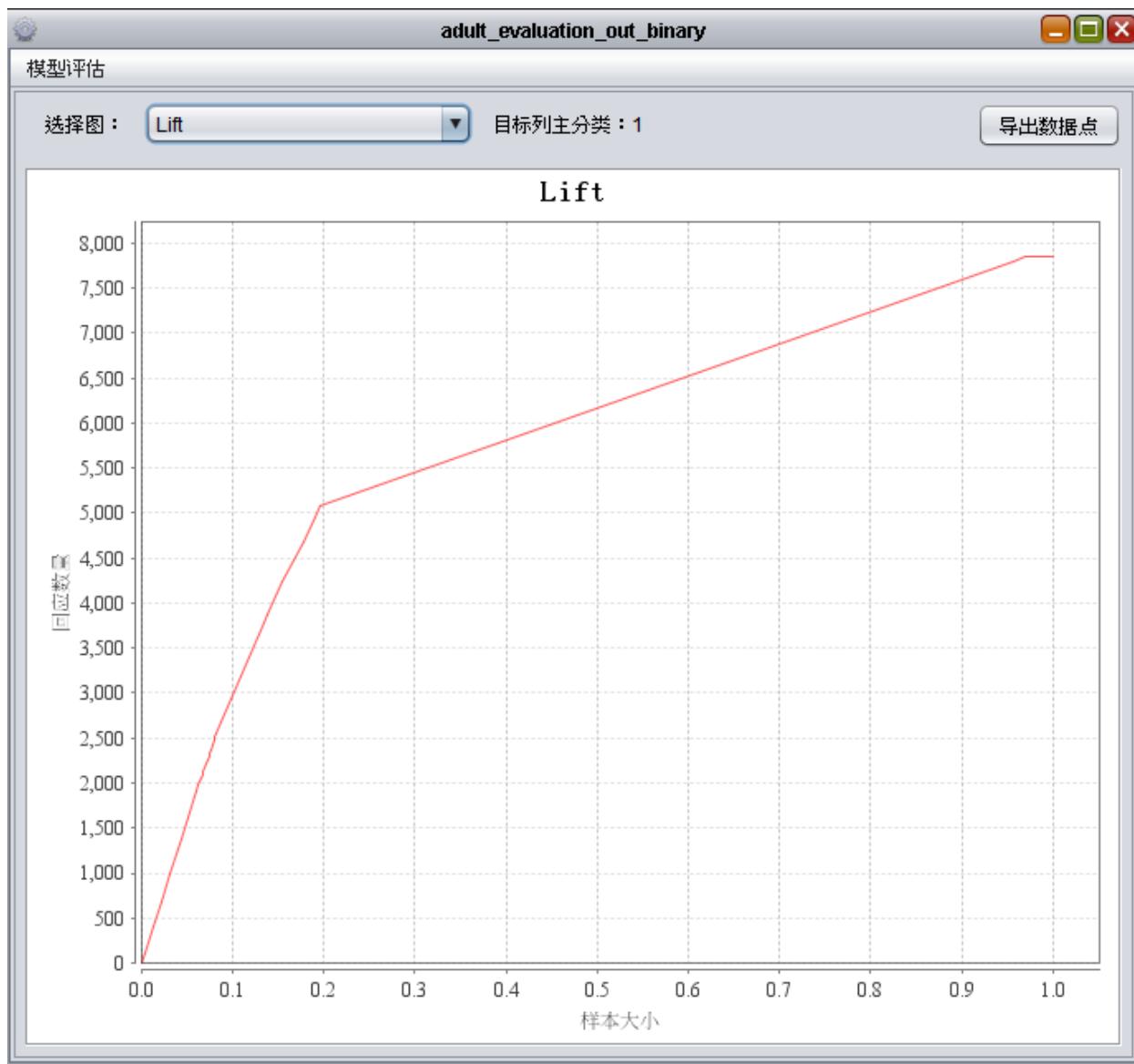
	a	b	TOTAL	<-- classified as
a = 0	23399	1321	24720	
b = 1	2758	5083	7841	
	26157	6404	32561	

- 概率评估的结果如下图：

阈值混淆矩阵：



ROC & Lift & Precision-Recall:



8.11.2 打开混淆矩阵或概率评估表

- “打开为...”模块，提供以何种方式打开数据，现在提供两种方式：混淆矩阵和ROC，如下图：



- 以”混淆矩阵”方式打开, 如果一个数据表是混淆矩阵表, 打开的界面如下图:

评估表adult_evaluation_out

模型评估

```

Correctly Classified Instances      28482          87.4727 %
Incorrectly Classified Instances   4079           12.5273 %
Kappa statistic                   0.6345
Total Number of Instances        32561

==== Detailed Accuracy By Class ====

      TP Rate    FP Rate    Precision    Recall    F-Measure  Class
      0.947     0.352      0.895       0.947     0.92       0
      0.648     0.053      0.794       0.648     0.714      1
Weighted Avg.      0.875     0.28       0.87       0.875     0.87

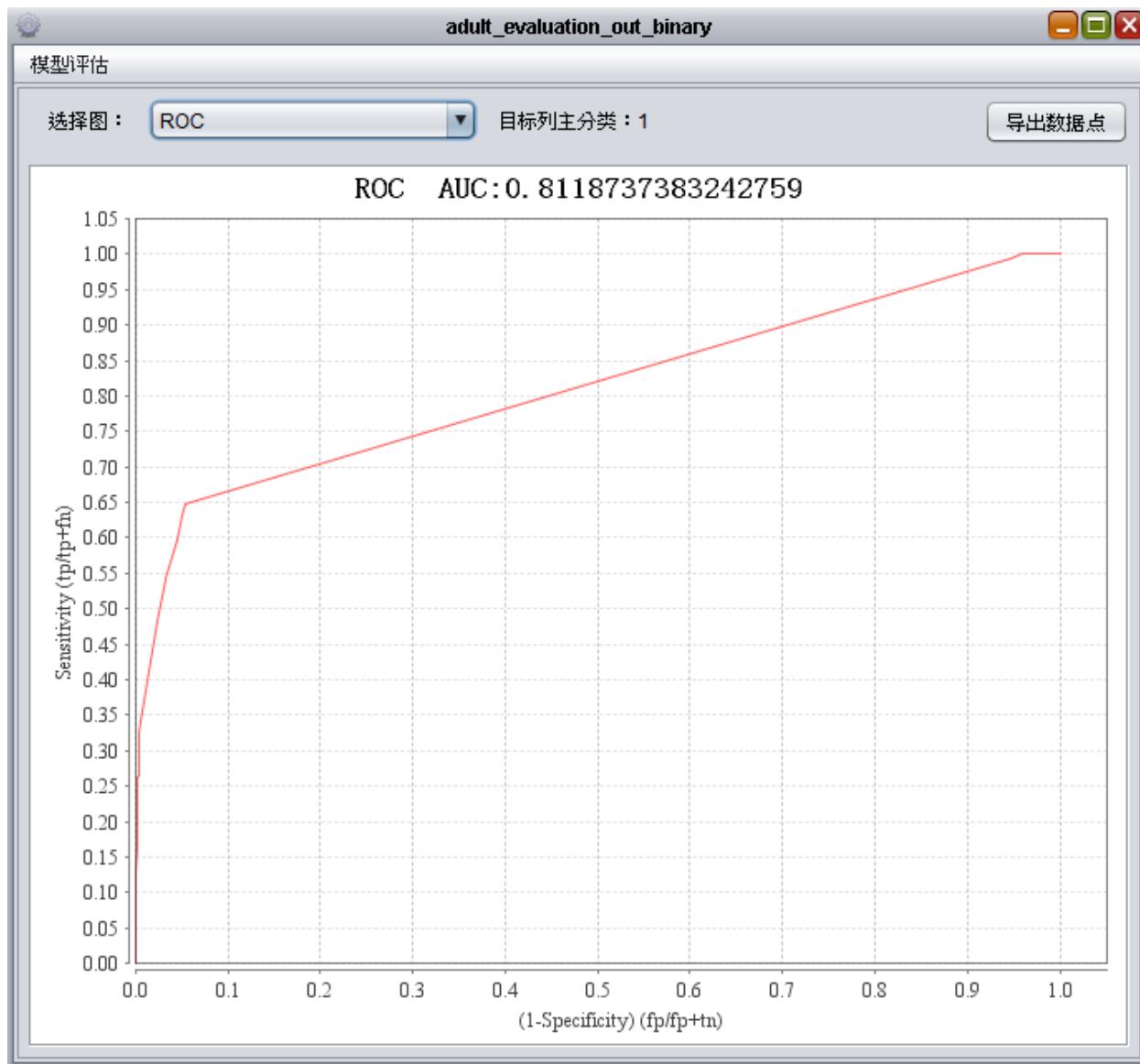
==== Confusion Matrix ===

      a      b      TOTAL    <-- classified as
23399  1321  24720 |      a = 0
 2758   5083  7841 |      b = 1
26157  6404  32561

```

否则：提示数据不合法。

- 以”画图ROC”方式打开，如果一个数据表是画图表格，打开的界面如下图：



否则：提示数据不合法。

8.11.3 计算正确率

```
def calcPrecision(referenceTableName, referenceLabelColName, targetTableName, targetLabelColName,
    referenceWeightColName=None, referencePartitions = None, targetPartitions=None):
```

参数：

- `referenceTableName`: 评估参考表
- `referenceLabelColName`: 评估参考表的结论列名
- `targetTableName`: 目标表
- `targetLabelColName`: 目标表的结论列名

- referenceWeightColName: (可选)评估参考表的权重列名
- referencePartitions: (可选)评估表输入partitions列表, 默认选择整表
- targetPartitions: (可选)目标表输出partitions列表, 默认选择整表

返回:

- 正确率P, Double类型, $p \in [0, 1]$

示例:

```
p=Classification.Evaluation.calcPrecision("adult", "class", "adult_model_test_pre", "conclusion")
p=Classification.Evaluation.calcPrecision("tree_adult_with_weight", "class",
    "tree_adult_with_weight_model_test_pre", "conclusion", referenceWeightColName="col1")
p=Classification.Evaluation.calcPrecision("tree_adult_with_weight_split", "class",
    "tree_adult_with_weight_model_test_split", "conclusion", referenceWeightColName="col1",
    referencePartitions=[{"pdate=20140403", "pdate=20140404"}, targetPartitions=[{"pdate=20140405", "pdate=20140406"}])
```

8.11.4 计算混淆矩阵

```
def calcConfusionMatrix(referenceTableName, referenceLabelColName, targetTableName,
    targetLabelColName, outputTableName, referenceWeightColName=None, referencePartitions = None,
    targetPartitions=None):
```

参数:

- referenceTableName: 评估参考表
- referenceLabelColName: 评估参考表的结论列名
- targetTableName: 目标表
- targetLabelColName: 目标表的的结论列名
- outputTableName: 评估输出表名
- referenceWeightColName: (可选)评估参考表的权重列名
- referencePartitions: (可选)评估表输入partitions列表, 默认选择整表
- targetPartitions: (可选)目标表输出partitions列表, 默认选择整表

返回:

- ConfusionMatrix, 混淆矩阵数据结构
 - classNames(), 目标列分类值的数组
 - matrix(), 混淆矩阵的二维数组, [actual][predict], 其中分类值得Index即为classNames() 的数组索引
 - numInstances(), 总的记录数

- threshold(), 当前混淆矩阵的阈值
- correct(), 正确分类的记录数组
- pctCorrect(), 正确分类的概率, 范围[0, 100]
- incorrect(), 错误分类的记录数
- pctIncorrect(), 错误分类的记录概率, 范围[0, 100]
- kappa(), kappa系数
- truePositiveRate()
- trueNegativeRate()
- falsePositiveRate()
- falseNegativeRate()
- recall(), recall
- precision()
- fMeasure()

示例:

```
cm=Classification.Evaluation.calcConfusionMatrix("adult", "class", "adult_model_test_pre",
                                                "conclusion", "adult_evaluation_out")
cm=Classification.Evaluation.calcConfusionMatrix("tree_adult_with_weight", "class",
                                                "tree_adult_with_weight_model_test_pre", "conclusion", "adult_evaluation_weight_out",
                                                referenceWeightColName="col1")
cm=Classification.Evaluation.calcConfusionMatrix("tree_adult_with_weight_split", "class",
                                                "tree_adult_with_weight_model_test_pre_split", "conclusion", "adult_evaluation_weight_out",
                                                referenceWeightColName="col1", referencePartitions=["pdate=20140403", "pdate=20140404"],
                                                targetPartitions=["pdate=20140405", "pdate=20140406"])

print cm
show(cm)
```

8.11.5 概率评估

计算有阈值混淆矩阵, ROC图, Lift图, Precision/Recall图。

```
calcProbEvaluation(referenceTableName, referenceLabelColName, targetTableName, probilityColName,
                    goodValue, outputTableName, referenceWeightColName=None, referencePartitions = None, targetPartitions=None):
```

参数:

- referenceTableName: 评估参考表
- referenceLabelColName: 评估参考表的结论列名

- targetTableName: 目标表
- probilityColName: 目标表的概率列
- goodValue: 目标表概率列对应的目标分类值, 即预测为goodValue的概率是probilityColName
- outputTableName: 评估输出表名
- referenceWeightColName: (可选)评估参考表的权重列
- referencePartitions: (可选)评估表输入partitions列表, 默认选择整表
- targetPartitions: (可选)目标表输出partitions列表, 默认选择整表

返回:

- ProbEvaluation, 概率评估数据结构
 - 计算AUC auc=ProbEvaluation.getRocCurve().calcAUC(), auc $\in [0, 1]$
 - 获取某个阈值混淆矩阵: ConfusionMatrix=ProbEvaluation.getProbConfusionMatrix().calcConfusionMatrix(threshold 建议 阈值 threshold $\in [0, 1]$) print ConfusionMatrix.threshold() 打印实际阈值 ConfusionMatrix 的数据结构可见帮助 help(Classification.Evaluation.calcConfusionMatrix)

示例:

```
probe=Classification.Evaluation.calcProbEvaluation("tree_adult_with_weight", "class",
    "tree_adult_with_weight_model_test_pre", "probability", "1", "adult_evaluation_weight_out",
    referenceWeightColName="col1")
probe=Classification.Evaluation.calcProbEvaluation("tree_adult_with_weight_split", "class",
    "tree_adult_with_weight_model_test_split", "probability", "1", "adult_evaluation_weight_out2",
    referenceWeightColName="col1", referencePartitions=[{"pdate=20140403", "pdate=20140404"},
    targetPartitions=[{"pdate=20140405", "pdate=20140406"}]
auc=probe.getRocCurve().calcAUC()
print auc
cm=probe.getProbConfusionMatrix().calcConfusionMatrix(0.5)
print cm.threshold()
show(probe)
```

8.11.6 加载混淆矩阵

```
def loadConfusionMatrix(tableName):
```

参数:

- tableName: 输入表名字

返回:

- ConfusionMatrix 混淆矩阵

- ConfusionMatrix的数据结构可见帮助 `help(Classification.Evaluation.calcConfusionMatrix)`

示例:

```
cm=Classification.Evaluation.loadConfusionMatrix("adult_model_test_cm")
```

8.11.7 加载概率评估数据

```
def loadProbEvaluation(tableName):
```

参数:

- `tableName`: 输入表名字

返回:

- `ProbEvaluation` 混淆矩阵
- `ProbEvaluation`的数据结构可见帮助 `help(Classification.Evaluation.calcProbEvaluation)`

示例:

```
pe=Classification.Evaluation.loadProbEvaluation("adult_evaluation_weight_out")
```

第 9 章

回归分析

回归分析(Regression Analysis)是一种统计学上分析数据的方法，目的在于了解两个或多个变量间是否相关、相关方向与强度，并建立数学模型以便观察特定变量来预测研究者感兴趣的变量。回归分析是建立因变量Y(或称依变量，反应变量)与自变量X(或称独变量，解释变量)之间关系的模型。简单线性回归使用一个自变量X，复回归使用超过一个自变量($X_1, X_2 \dots X_i$)。详细介绍见：[Regression analysis](#)。在Xlab中，回归分析主要支持两种：[线性回归](#) 和 [梯度渐近回归树](#)。

9.1 线性回归

线性回归(Linear Regression)是利用称为线性回归方程的最小二乘函数对一个或多个自变量和因变量之间关系进行建模的一种回归分析。这种函数是一个或多个称为回归系数的模型参数的线性组合。只有一个自变量的情况称为简单回归，大于一个自变量情况的叫做多元回归。

线性回归是回归分析中第一种经过严格研究并在实际应用中广泛使用的类型。这是因为线性依赖于其未知参数的模型比非线性依赖于其位置参数的模型更容易拟合，而且产生的估计的统计特性也更容易确定。详细介绍见：[Linear regression](#)。

在XLab中，对于线性回归模型，提供两种执行方式：函数和界面，主要包括两个过程：训练和预测。

9.1.1 函数

线性回归包括以下几个函数：train, predict, loadModel和isModel。这几个函数具体使用方法，可用help命令查看，示例：

```
help(Regression.LinearReg.train)
help(Regression.LinearReg.predict)
help(Regression.LinearReg.loadModel)
help(Regression.LinearReg.isModel)
```

训练

```
def train(independentColNames, dependentColName, inputTableName = None,
          inputPartitions = None, srt = None, outModelTableName = None):
```

参数:

- independentColNames: 自变量列名列表(x).
- dependentColName: 因变量列名(y).
- inputTableName: (可选)输入表名.
- inputPartitions: (可选)输入表的分区.
- srt: (可选)inputTableName和inputPartitions下的SummaryResultTable.
- outModelTableName: (可选)输出模型表的名字.

返回:

- LinearRegressionModel

示例:

```
linearRegressionModel = Regression.LinearReg.train(["balance"], "age", "bank_marketing")
```

预测

```
def predict(inputTableName, model, outputTableName,
            appendColNames = None, inputPartitions = None):
```

参数:

- inputTableName: 输入预测表.
- model: 线性回归模型.
- outputTableName: 输出表.
- appendColNames: (可选)追加列列表
- inputPartitions: (可选)输入表的分区列表.

示例:

```
Regression.LinearReg.predict("bank_marketing", model, "outTableName",
    appendColNames=["age", "balance"])
```

加载模型

```
def loadModel(modelTableName):
```

参数:

- modelTableName: 模型表

返回:

- LinearRegressionModel

示例:

```
linearRegressionModel = Regression.LinearReg.loadModel("modelTableName")
```

判断模型

```
def isModel(modelTableName):
```

参数:

- modelTableName: 模型表

返回:

- 布尔值, true: 是线性回归模型, false: 不是线性回归模型

示例:

```
linearRegressionModel = Regression.LinearReg.isModel("modelTableName")
```

9.1.2 界面

界面操作包括两个过程: 训练和预测, 详见下文。

训练

打开数据表->模型->线性回归->训练, 如下图:

kddcup - 数据表浏览

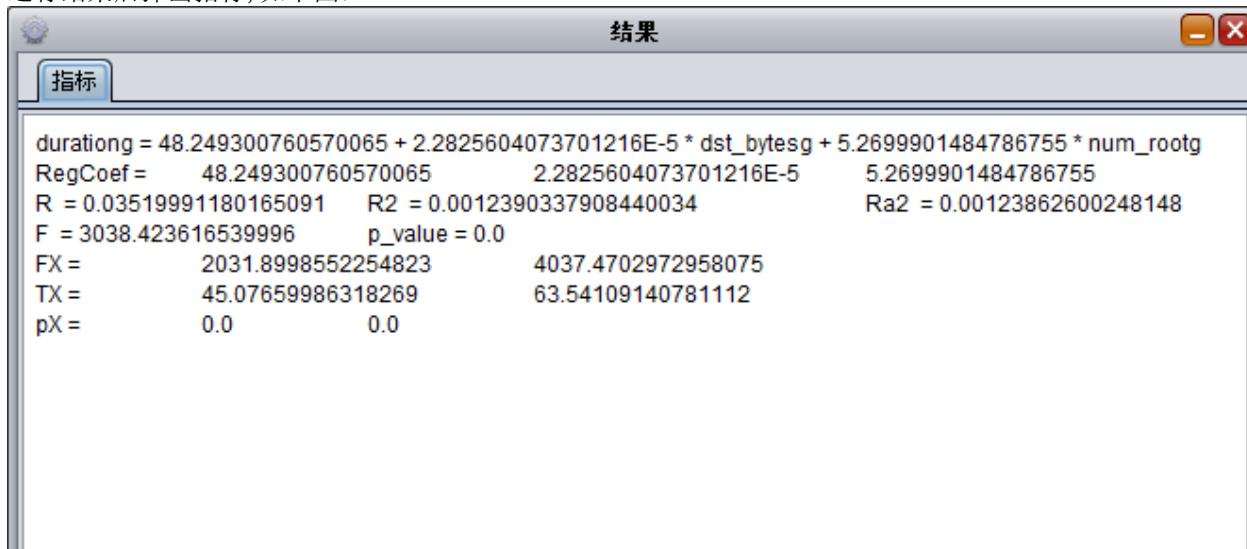
表格 数据处理 全表描述 统计 分析 模型 工具 画图

	duration	protocol_type	service	属性选择	byteseg	landg	wrong_fra...	urgencg	hotg
0	0	tcp	http	随机森林	6	0	0	0	0
1	0	tcp	http	逻辑回归	0	0	0	0	0
2	0	tcp	http	线性支持向量机(LinearSVM)	0	0	0	0	0
3	0	tcp	http	非线性支持向量机(NonlinearSVM)	0	0	0	0	0
4	0	tcp	http	朴素贝叶斯(Naive Bayes)	0	0	0	0	0
5	0	tcp	http	贝叶斯判别(Bayes)	0	0	0	0	0
6	0	tcp	http	费希尔判别(Fisher)	0	0	0	0	0
7	0	tcp	http	马氏距离判别(MDistance)	0	0	0	0	0
8	0	tcp	http	评估	0	0	0	0	0
9	0	tcp	http	梯度渐近回归树(GBRT)	0	0	0	0	0
10	0	tcp	http	线性回归	训练	0	0	0	0
11	0	tcp	http		预测	0	0	0	0
12	0	tcp	http			0	0	0	0
13	0	tcp	http			0	0	0	0
14	0	tcp	http	SF	236	1228	0	0	0
15	0	tcp	http	SF	233	2032	0	0	0
16	0	tcp	http	SF	238	1282	0	0	0
17	0	tcp	http	SF	235	1337	0	0	0
18	0	tcp	http	SF	234	1364	0	0	0
19	0	tcp	http	SF	239	486	0	0	0
20	0	tcp	http	SF	185	9020	0	0	0
21	0	tcp	http	SF	184	124	0	0	0
22	0	tcp	http	SF	181	5450	0	0	0
23	0	tcp	http	SF	239	1295	0	0	0
24	0	tcp	http	SF	236	1228	0	0	0

选择因变量和协变量, 进行训练, 如下图:



运行结束后弹出指标, 如下图:



预测

打开预测表或者线性回归模型表->模型->线性回归->预测, 如下图:



9.2 梯度渐近回归树

梯度渐近回归树算法介绍: Gradient boosting 算法介绍。本算法目前支持Gradient boosting regression tree. 也就是只支持回归。使用的损失函数只支持最小二乘。

在XLab中, 对于梯度渐近回归树模型, 提供两种执行方式: 函数和界面, 主要包括两个过程: 训练和预测。

9.2.1 函数

GBRT包括以下几个函数: trainSparse, predictSparse, loadModel和isModel。

这几个函数具体使用方法, 可用help命令查看, 示例:

```
help(Regression.GradBoostRegTree.trainSparse)
help(Regression.GradBoostRegTree.predictSparse)
help(Regression.GradBoostRegTree.loadModel)
help(Regression.GradBoostRegTree.isModel)
```

训练

```
def trainSparse(independentMatrixName, dependentTableName, dependentColName,
               modelTableName, dependentPartitions = None, treeDepth = 5,
               treesNum = 100, learningRate = 0.1, minSampleLeaf = 2,
               colTypes = None) :
```

参数:

- independentMatrixName: 输入的自变量(x)矩阵。
- dependentTableName: 输入的因变量(y)表名。
- dependentColName: 因变量的列名。
- modelTableName: 输出的模型名。
- dependentPartitions: (可选)因变量表对应的输入矩阵。选择全表则为None。默认为选择全表。
- treeDepth: (可选)单颗树的最大深度。默认为5.
- treesNum: (可选)树的棵数。默认为100.
- learningRate: (可选)学习速率。默认为0.1.
- minSampleLeaf: (可选)叶结点最小样本数。要求大于0, 默认为2。
- colTypes: (可选)列类型。True为连续型, False为离散型。默认None的情况下, 都为连续型。

返回:

- 返回GradBoostTreeModel类型, 表示梯度渐近树模型。

示例:

```
Regression.GradBoostRegTree.trainSparse("gbrt_train_sparse", "gbrt_train_dependent_table", "dependent",
                                         "gbrt_model", dependentPartitions = ["p1"], treeDepth = 5, treesNum = 100,
                                         learningRate = 0.1, minSampleLeaf = 2, colTypes = [True, False])
```

注意事项:

- 本方法实现了梯度渐近回归树的训练过程。
- 对于因变量y, 只支持double和bigint类型
- 自变量只支持 Xlib稀疏矩阵表 输入

预测

```
def predictSparse(inputMatrixName, model, outputTableName, outputPartition = None) :
```

参数:

- inputMatrixName: 预测输入矩阵名。
- model: GradBoostTreeModel类型。注意这里不是String类型, 见example。梯度渐近树模型。
- outputTableName: 预测输出表。
- outputPartition: (可选)指定输出到输出表的分区。None表示无分区。默认为无分区。

示例:

```
gbrtModel = Regression.GradBoostRegTree.trainSparse("gbrt_train_sparse", "gbrt_train_dependent_table", "dependent",
                                                 "gbrt_model", dependentPartitions = ["p=1"], treeDepth = 5, treesNum = 100,
                                                 learningRate = 0.1, minSampleLeaf = 2, colTypes = [True, False])
#如果训练和预测分开使用, 请使用Regression.GradBoostRegTree.loadModel(modelTableName)作为预测方法模型参数
Regression.GradBoostRegTree.predictSparse("gbrt_train_sparse", gbrtModel, "gbrt_prediction", outputPartition = "p=1")
```

注意事项:

- 本方法实现了梯度渐近回归树的预测过程。

加载模型

```
def loadModel(modelTableName) :
```

参数:

- modelTableName: 模型表名。

返回:

- GradBoostTreeModel: 梯度渐近树模型

示例:

```
gbrtModel = Regression.GradBoostRegTree.loadModel("table_name")
```

判断模型

```
def isModel(modelTableName) :
```

参数:

- modelTableName: 检查的模型表名。

返回:

- Boolean: 是否为梯度渐近回归树模型。

示例:

```
if Regression.GradBoostRegTree.isModel("table_name") :
    print "gbrt model."
```

9.2.2 界面

在本例子中使用pgbrt_train_label表的label列作为因变量, 使用稀疏矩阵pgbrt_train_sparse作为自变量, pgbrt_train_model_d5作为模型输出。使用步骤如下:

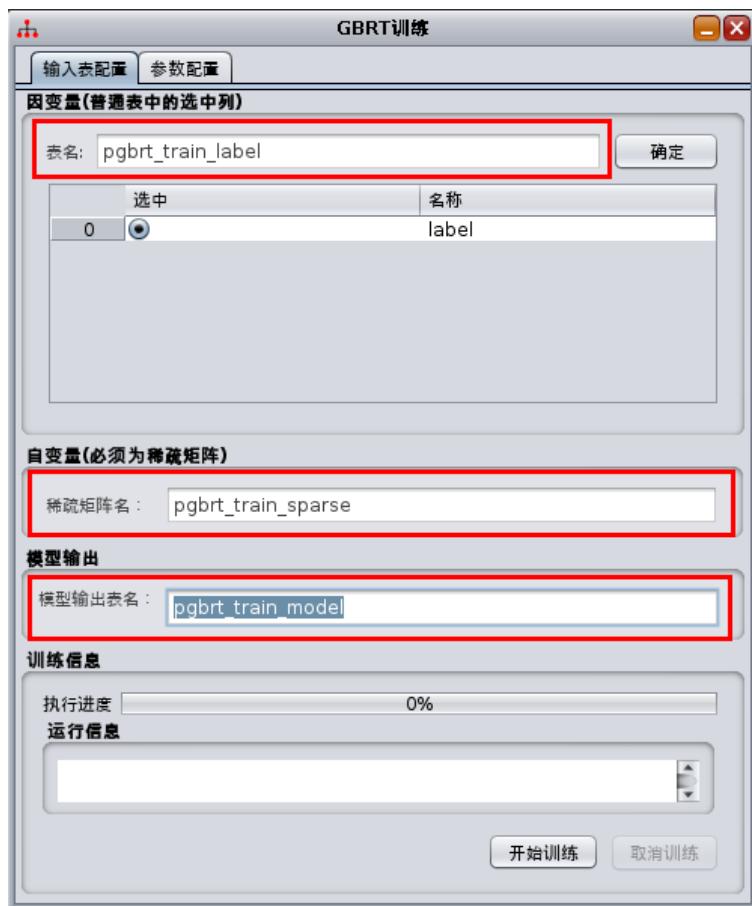
打开pgbrt_train_label表, 在菜单->模型->梯度渐近回归树, 打开模型相关功能:



训练

在训练界面中的参数输入是和函数中的参数相对应的, 参见[训练](#).

打开训练界面为:



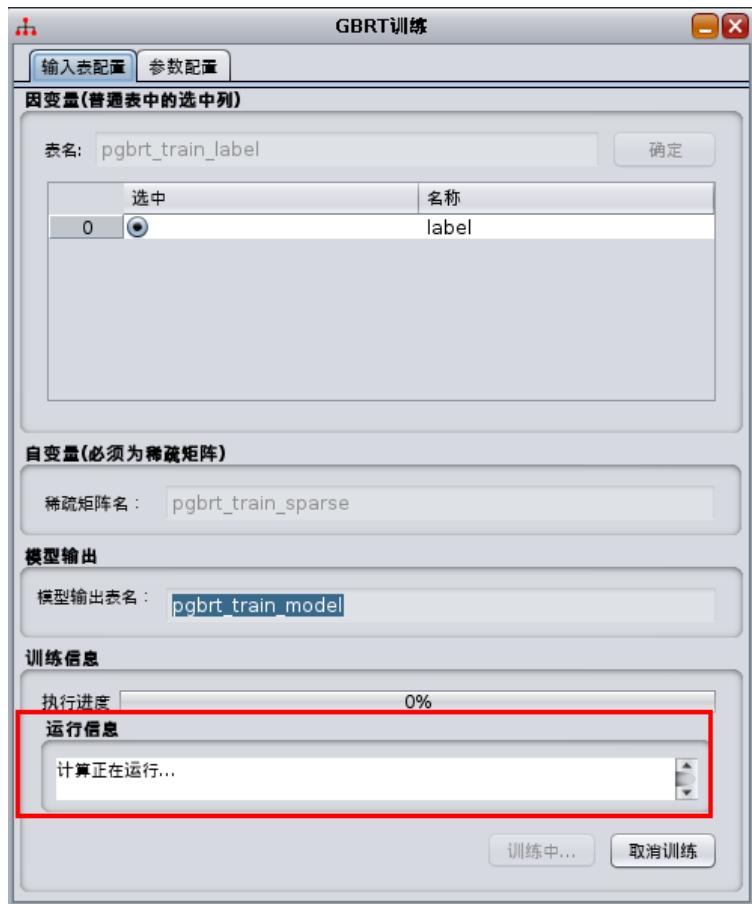
在本界面中：

- 第一个红色方框中代表因变量所在的表，其中选中的行代表因变量在表中所在的列。
- 第二个红色方框中代表自变量的 Xlib 稀疏矩阵表名。
- 第三个红色方框中代表输出模型表名。

点击开始训练按钮，即开始训练。在本界面中有两个tab页，输入表配置页由上所述，参数配置页可以选择算法的参数，如下：



开始训练界面如下：



训练结束会弹出模型显示窗口，参见[模型显示](#)。

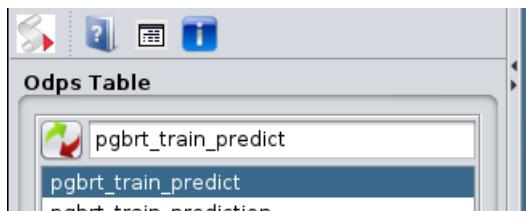
预测

在预测界面中的参数输入是和函数中的参数相对应的，参见[预测](#)。

打开预测界面为：



输入表为要预测的表，model为训练时的输出表，输出表为预测的结果。点击预测按钮，开始预测。在预测中，主界面上选择GBRT模型时，预测会自动识别这个表是否为GBRT模型。预测结束，选择预测结果表，如下图：



双击打开，查看预测结果，如下图：

pgbrt_train_predict - 数据表浏览	
	Y_var
0	0.031069748311264987
1	0.9575569505813396
2	0.06040185311225389
3	0.9921976373997257
4	0.06343798622790846
5	0.967377190019007
6	0.03207699139611464
7	0.9778725522541086
8	0.9469349167479941
9	-0.03488824310308909
10	0.9763666893210456
11	0.038567188758896614
12	0.9936777179149722
13	0.9957329789028606
14	0.9838245715533454
15	1.0132808318707203
16	1.0349587921761227
17	0.9616630799068329
18	0.9746016088702452
19	0.9462626327929477
20	1.0183131166268027
21	0.0658223939689425
22	1.000316810876327
23	0.9735553179006371
24	0.9665889599993732
25	0.9911029180851766
26	1.9370496680614755
27	0.015884006873773285
28	0.9853684286563512
29	0.07320010073202027

根据预测结果, 可以调用矩阵计算计算rmse值等。rmse的计算过程如下作为参考, 在脚本中执行:

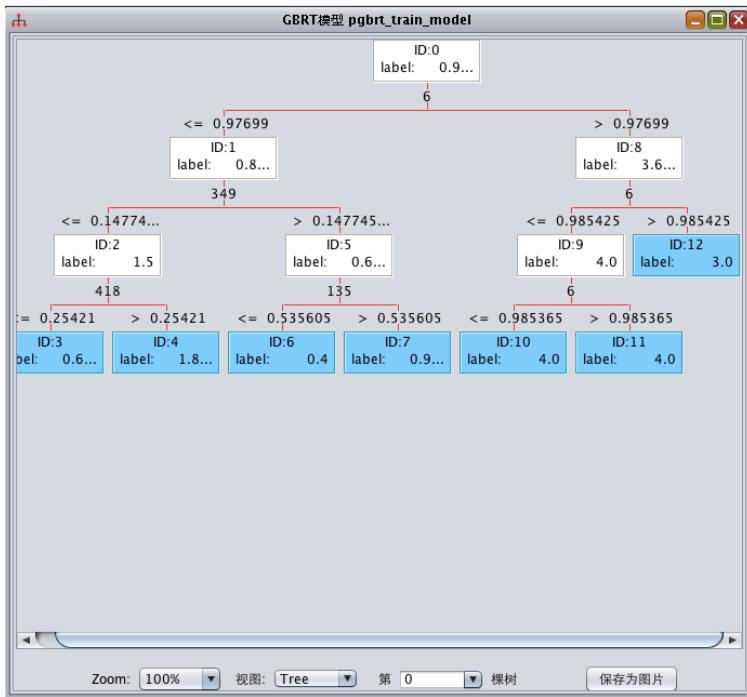
```
import math

before = Matrix("pgbrt_train_label", colNames = ['label'])
after = Matrix("pgbrt_train_predict", colNames = ['y_var'])
minus = after - before
norm = minus.norm('2')
row = before.rowSize()
rmse = norm / math.sqrt(float(row))

print rmse
```

模型显示

对于训练出来的模型, 可以通过XLab提供的树的图形表示来展示, 如下图:



模型显示提供了以下功能：

- 放大缩小；
- tree, folder, print三种视图；
- 选择树的编号。

9.3 线性支持向量回归(Linear SVR)

支持向量回归是采用支持向量方法处理回归问题。SVR算法介绍：Support vector machine。

9.3.1 函数

线性SVR包括以下几个函数：train，trainSparse，predict，predictSparse和loadModel。

这几个函数具体使用方法，可用help命令查看，示例：

```
help(Regression.LinearSVR.train)
help(Regression.LinearSVR.trainSparse)
help(Regression.LinearSVR.predict)
help(Regression.LinearSVR.predictSparse)
help(Regression.LinearSVR.loadModel)
```

注意事项；

- 支持输入训练集格式为普通表和稀疏矩阵

训练

```
def train(inputTableName, independentColNames, dependColName, modelTableName, inputPartitions = None, cost = 1.0, epsilon =
```

参数:

- inputTableName: 输入表名, 普通表
- independentColNames: 特征列名, 字符数组
- dependColName: 目标列名
- modelTableName: 模型表名, 普通表
- inputPartitions: (可选)输入表分区
- cost: (可选)惩罚因子, 默认值为1
- epsilon: (可选)收敛系数, 默认值为0.0001
- tau: (可选)不敏感因子, 默认值为0.1

返回:

- LinearSVRModel 线性svr模型

示例:

```
selectedColNames = ['0' for i in range(8)]  
  
for i in range(0, 8):  
  
    selectedColNames[i] = ('col' + str(i+1))  
  
Regression.LinearSVR.train("cadata_append", selectedColNames, "target_col", "cadata_model", cost = 100, epsilon =
```

```
def trainSparse(independentMatrixName, dependentTableName, dependentColName, modelTableName, dependentPartitions = None, cos
```

参数:

- featureMatrixName: 输入稀疏矩阵名
- dependentTableName: 目标列所在表名
- dependentColName: 目标列名
- modelTableName: 模型表名, 普通表
- dependentPartitions: (可选)label列所在表分区
- cost: (可选)惩罚因子, 默认值为1
- epsilon: (可选)收敛系数, 默认值为0.0001
- tau: (可选)不敏感因子, 默认值为0.1

返回:

- LinearSVRModel 线性svr模型

示例:

```
Regression.LinearSVR.trainSparse("bodyfat_sparse", "bodyfat_append", "target_col", "bodyfat_model_sparse", cost = 100, epsilon = 0.0001)
```

预测

```
def predict(inputTableName, model, outputTableName, inputPartitions = None, appendColNames = None, outputPartition = None):
```

参数:

- inputTableName: 输入表名, 普通表
- model: 模型, 类型为LinearSVMModel
- outputTableName: 输出表名, 普通表
- inputPartitions: (可选)输入表分区
- appendColNames: (可选)输入表中插入到输出表中的列名
- outputPartition: (可选)输出表分区

返回:

- void 无

示例:

```
selectedColNames = [0 for i in range(8)]
for i in range(0, 8):
    selectedColNames[i] = ('col' + str(i+1))

svr_model = Regression.LinearSVR.train("cadata_append", selectedColNames, "target_col",
                                         "cadata_model", cost = 100, epsilon = 0.0001, tau = 0.0001)

Regression.LinearSVR.predict("cadata_append", svr_model, "cadata_predict")
```

```
def predictSparse(inputTableName, model, outputTableName, outputPartition = None):
```

参数:

- inputTableName: 输入表名, Xlib稀疏矩阵表

- model: 模型, 类型为LinearSVMModel
- outputTableName: 输出表名, 普通表
- outputPartition: (可选)输出表分区

返回:

- void 无

示例:

```
for i in range(0, 14):

    selectedColNames[i] = ('col' + str(i+1))

    svr_model = Regression.LinearSVR.trainSparse("bodyfat_sparse", "bodyfat_append", "target_col", "bodyfat_model_sp

    cost = 100, epsilon = 0.0001, tau = 0.0001)

    Regression.LinearSVR.predictSparse("bodyfat_sparse", svr_model, "bodyfat_predict_sparse")
```

加载模型

```
def loadModel(modelTableName):
```

参数:

- modelTableName: 模型表表名, 普通表

返回:

- LinearSVRModel 线性svr模型

示例:

```
Regression.LinearSVR.loadModel("svr_model")
```

9.3.2 界面

界面包括: 训练和预测。

- 只支持输入训练集格式为普通表。

训练

通过UI界面(只支持输入训练集格式为普通表)选择数据表, 点击模型→线性支持向量回归→训练



其中：

- Partition筛选：选择参与线性支持向量机模型训练的partition。
- 目标变量：数据表中目标列，支持Long, double类型。
- - 输入表选中列：参与线性支持向量回归计算的列，只支持数值型。
- 模型输出表：线性支持向量回归模型输出表。
- 惩罚因子：C，软间隔SVM中的用来衡量对噪声点的重视程度。
- 收敛系数：目标函数梯度的相对变化低于该值，认为算法收敛。
- 不敏感因子：损失函数中的不敏感系数。

选择完毕单击“确定”开始训练。训练结果：模型表，其中最后一行表示截距，如下图：

model_info		coeff
0	{"featuresList": ["col1", "col2", ...]	0
1	col1	43358.698359442205
2	col2	1712.8740730403626
3	col3	-11.310880565854287
4	col4	60.8260904781327
5	col5	-43.55369826475666
6	col6	139.51341616770992
7	col7	-13119.527791631277
8	col8	-3740.9405434531814
9	Intercept	-30.825514552850848

预测

通过UI界面(只支持输入预测数据集为普通表, 且输入模型表由普通表训练得到)选择数据表, 点击模型→线性支持向量机→预测



其中：

- Partition筛选: 选择参与线性SVR预测的partition。
- 使用模型: 选择进行线性SVR预测的模型表。
- 预测输出表: 选择预测结果的输出表。

- 保留列选择：选择会保留出现在预测结果表中的列。

选择完毕单击“确定”开始预测。预测结果：

predict_values	
0	392901.6873110269
1	397561.4126070442
2	362483.6245989373
3	300606.495450079
4	226715.34435862178
5	236324.5443136956
6	233712.51527033022
7	231756.91721707987
8	165072.55366207575
9	244893.8274446732
10	206656.62906800673
11	234653.80389171725
12	201286.36913705026
13	178881.62034393832
14	174831.2183892843
15	149563.58762991687
16	180700.0077967772
17	159780.46367848958
18	150557.61132097305
19	172316.25345081746
20	96230.54521086309
21	121420.85511428563
22	152628.2686459628
23	154144.65426947552
24	179218.3549597011
25	139516.59553684827
26	160384.3598139805
27	139829.61916575013
28	140932.27862816027
29	180112.1100100110

第 10 章

聚类分析

聚类(Clustering)，简单地说就是把相似的东西分到一组，同分类(Classification)不同，对于一个classifier，通常需要你告诉它“这个东西被分为某某类”这样一些例子，理想情况下，一个classifier会从它得到的训练集中进行“学习”，从而具备对未知数据进行分类的能力，这种提供训练数据的过程通常叫做监督学习(supervised learning)，而在聚类的时候，我们并不关心某一类是什么，我们需要实现的目标只是把相似的东西聚到一起，因此，一个聚类算法通常只需要知道如何计算相似度就可以开始工作了，因此clustering通常并不需要使用训练数据进行学习，这在机器学习中被称作无监督学习(unsupervised learning)。

10.1 KMeans

K均值聚类是一种得到最广泛使用的聚类算法，把n个对象分为k个簇，使簇内具有较高的相似度。相似度的计算根据一个簇中对象的平均值来进行。它与处理混合正态分布的最大期望算法很相似，因为他们都试图找到数据中自然聚类的中心。

算法首先随机地选择k个对象，每个对象初始地代表了一个簇的平均值或中心。对剩余的每个对象根据其与各个簇中心的距离，将它赋给最近的簇，然后重新计算每个簇的平均值。这个过程不断重复，直到准则函数收敛。

它假设对象属性来自于空间向量，并且目标是使各个群组内部的均方误差总和最小。KMeans的更多详细介绍请参考 [相关文档链接](#)

10.1.1 函数

执行`help(Cluster.kmeans)`可以查看帮助信息

```
def kmeans(inputTableName, centerCount,
           idxTableName, centerTableName, clusterCountTableName,
           selectedColNames = None, selectedPartitions = None,
           loop = 100, accuracy = 0.0, distanceType = 'euclidean',
           initCenterTableName = None, initCentersMethod = 'sample', appendColsIndex=None)
```

参数:

- `inputTableName`: 输入表名
- `centerCount`: 聚类数
- `idxTableName`: 输出聚类编号表, 行数等于输入表总行数, 每行的值表示输入表对应行表示的点的聚类编号
- `centerTableName`: 输出聚类质心表, 列数等于输入表(有选中则等于选中列总数), 行数等于聚类数, 每行表示一个聚类质心位置
- `clusterCountTableName`: 输出聚类点总数表名, 行数等于聚类数, 每行聚类质心的类中所包含的聚类点的总数
- `selectedColNames`: (可选) 输入表选中列, 默认为全选, 目前只支持double类型
- `inputPartitions`: (可选) 输入表选择分区, 默认为全选
- `loop`: (可选) 计算最大迭代次数, 默认为100
- `accuracy`: (可选) 计算最小迭代精度, 默认为0.0
- `distanceType`: (可选) 距离测度方法, 支持欧式距离(euclidean, 默认), 绝度误差和(city-block), 夹角余弦(cosine)
- `initCenterTableName`: (可选) 输入起始center值的表格, 当`initCentersMethod`为matrix时候必须输入
- `initCentersMethod`: (可选) 初始质心位置选择方法, 支持sample(随机选取, 默认), topk(前K行), uniform(分布范围均匀随机生成), matrix(指定表作为初始质心位置), kmpp(kmeans++初始化)
- `appendColsIndex`: (可选) 输出聚类编号表附加列, 默认为不选

示例:

```
Cluster.kmeans("kmeans_input", 4, "idxTable", "centerTable", "clusterCountTable", loop=100)
```

- 读取表`kmeans_init`所有列进行kmeans计算
- 使用默认的距离计算方法和质心选择方法, 当迭代次数到100次时候结束计算

- 结果存储到idxTable, centerTable, 和clusterCountTable三张表中

10.1.2 界面

KMeans界面参数如下图:



其中：

- 最上层为选择输入表的partition;
- 下方输入表达式中列为选择参与KMeans计算的特征列，对应脚本参数selectedColNames，目前只支持double类型的特征列；
- 初始质心位置选择模块分别对应脚本参数initCentersMethod和initCenterTableName

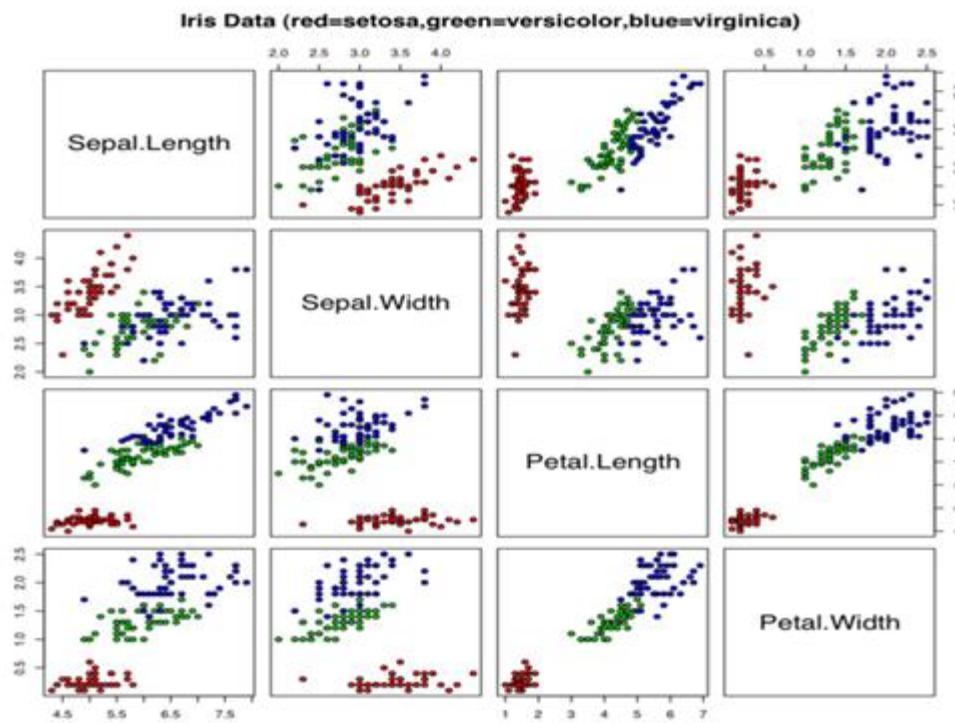
- 迭代参数对应脚本loop和accuracy;
- 计算模块中聚类数对应脚本参数centerCount;
- 距离计算方式对应脚本参数distanceType;
- 聚类编号表名, 质心表名, 个数表名分别对应脚本参数dxTableName, centerTableName, clusterCountTableName;
- 参数填写完毕后点击确定可以执行。执行结束后界面会一次弹出三张输出表。

10.1.3 例子

这里我们采用了iris数据集(Iris flower data set)进行实例, 数据如下:

	sepal_length	sepal_width	petal_length	petal_width	category
0	5.1	3.5	1.4	0.2	Iris Setosa
1	4.9	3	1.4	0.2	Iris Setosa
2	4.7	3.2	1.3	0.2	Iris Setosa
3	4.6	3.1	1.5	0.2	Iris Setosa
4	5	3.6	1.4	0.2	Iris Setosa
5	5.4	3.9	1.7	0.4	Iris Setosa
6	4.6	3.4	1.4	0.3	Iris Setosa
7	5	3.4	1.5	0.2	Iris Setosa
8	4.4	2.9	1.4	0.2	Iris Setosa
9	4.9	3.1	1.5	0.1	Iris Setosa
10	5.4	3.7	1.5	0.2	Iris Setosa
11	4.8	3.4	1.6	0.2	Iris Setosa
12	4.8	3	1.4	0.1	Iris Setosa
13	4.3	3	1.1	0.1	Iris Setosa
14	5.8	4	1.2	0.2	Iris Setosa
15	5.7	4.4	1.5	0.4	Iris Setosa
16	5.4	3.9	1.3	0.4	Iris Setosa
17	5.1	3.5	1.4	0.3	Iris Setosa
...

Iris以鸢尾花的特征作为数据来源, 数据集包含150个数据集, 分为3类, 每类50个数据, 每个数据包含4个属性, 该数据集数据分布情况如下, 可以看到在二维关系下, setosa较为独立, versicolor, virginica中间有部分交杂情况。



首先在界面上选中k均值聚类:

iris - 数据表浏览

	sepal_length	sepal_width	k均值聚类	petal_length	petal_width	category
128	6.4	2.8	关联分析	5.6	2.1	Iris Virginica
129	7.2	3	对应分析	5.8	1.6	Iris Virginica
130	7.4	2.8	多维对应分析	6.1	1.9	Iris Virginica
131	7.9	3.8	主成分分析	6.4	2	Iris Virginica
132	6.4	2.8		5.6	2.2	Iris Virginica
133	6.3	2.8		5.1	1.5	Iris Virginica
134	6.1	2.6		5.6	1.4	Iris Virginica
135	7.7	3		6.1	2.3	Iris Virginica
136	6.3	3.4		5.6	2.4	Iris Virginica
137	6.4	3.1		5.5	1.8	Iris Virginica
138	6	3		4.8	1.8	Iris Virginica
139	6.9	3.1		5.4	2.1	Iris Virginica
140	6.7	3.1		5.6	2.4	Iris Virginica
141	6.9	3.1		5.1	2.3	Iris Virginica
142	5.8	2.7		5.1	1.9	Iris Virginica
143	6.8	3.2		5.9	2.3	Iris Virginica
144	6.7	3.3		5.7	2.5	Iris Virginica
145	6.7	3		5.2	2.3	Iris Virginica
146	6.3	2.5		5	1.9	Iris Virginica
147	6.5	3		5.2	2	Iris Virginica
148	6.2	3.4		5.4	2.3	Iris Virginica
149	5.9	3		5.1	1.8	Iris Virginica

显示参数窗口，并按照上文说明填写对应参数:



这里我们设置了最大迭代100次，最小精度为0，聚类数为3(由于原表category列为类别，可以对比下我们的聚类结果和原表值)，点击右下角执行开始计算。结果聚类编号表为：

kmeans结果-Id表:iris_idx

cluster_index	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
...	0

Whole Table ▾ 位置 数据大小: 150 行, 当前显示前 10000 行 (最多10000行)

其中的0, 1, 2为分类标识

聚类质心表为:

kmeans结果-Center表:iris_center

	col0	col1	col2	col3
0	5.00599999999999	3.4180000000000006	1.464	0.243999999999999
1	5.901612903225806	2.7483870967741932	4.393548387096774	1.433870967741935
2	6.850000000000005	3.073684210526315	5.742105263157893	2.0710526315789473

Whole Table ▾ 位置 数据大小: 3 行, 当前显示前 50 行 (最多10000行) GO!

每行对应一个分类的质心位置。

聚类个数表为:

cluster_count	
0	50
1	62
2	38

Whole Table ▾ 位置 数据大小: 3 行, 当前显示前 50 行 (最多10000行) GO!

分别对应每个聚类中的个数。

对于setosa类别, KMeans得到较好的聚类结果, versicolor, virginica两个类别中出现交杂情况, 结果符合数据实际情况, 提高聚类迭代次数和更好的选择初始点选择可以提升运行结果。三张数据表在xlab中是有序的, 可以通过AppendColumn函数对结果进行贴合。

注意事项:

- 当前KMeans支持最大2000亿(输入表行*列)规模的计算。

10.2 Canopy

与传统的聚类算法(比如K-means)不同, Canopy聚类最大的特点是不需要事先指定k值(即clustering的个数), 因此具有很大的实际应用价值。与其他聚类算法相比, Canopy聚类虽然精度较低, 但其在速度上有很大优势, 因此可以使用Canopy聚类先对数据进行“粗”聚类, 得到k值后再使用K-means进行进一步“细”聚类。

(注意: 当前仅用于辅助计算kmeans的k值选择和初始质心, 未来可能会对调用接口和输入输出有所调整)

Canopy的更多详细介绍请参考 [维基百科](#)

10.2.1 函数

执行help(Cluster.canopy)可以查看帮助信息

```
def canopy(inputTableName, t1, t2, centerTableName,
           selectedColNames = None, inputPartitions = None, distanceType = 'euclidean')
```

参数:

- inputTableName: 输入表名(普通表)
- t1: t1值(the loose distance), t1>t2(此参数当前无意义)
- t2: t2值(the tight distance), t2>=0
- centerTableName: 输出canopy质心表, 列数等于输入表(有选中则等于选中列总数), 行数等于聚类数, 每行表示一个质心位置

- selectedColNames (可选) 输入表选中列 (只支持double类型), 默认为全选
- inputPartitions: (可选) 输入表选择分区, 默认为全选
- distanceType: (可选) 距离测度方法, 支持欧式距离 (euclidean, 默认), 绝度误差和 (cityblock), 夹角余弦 (cosine)

示例:

```
Cluster.canopy("canopy_input", 1, 0.5, "centerTable", selectedColNames = ['col0','col1'], inputPartitions = ['ds=20140505'],
```

10.3 EM聚类

EM 算法是一种循环优化算法, 用来处理包含缺失观测的数据。在很多实际应用中, 我们可能会收集到来自随机变量 X 的观测数据, 同时伴随着来自随机变量 Z 的不可观测(缺失) 数据。给定观测数据 X, 希望通过最大化似然函数 $L(\theta|X)$ 来给出参数 θ 的估计。而直接最大化似然函数 $L(\theta|X)$ 往往是件困难的事情。EM 算法假设完全数据来自随机变量 $Y=(X, Z)$, 考虑 使用 $Y|\theta$ 和 $Z|(X, \theta)$ 的条件密度函数来求解。 http://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm

10.3.1 函数

执行`help(Cluster.emcluster)`可以查看帮助信息

```
def emcluster(inputTableName, centerCount, resultTableName, centerTableName,
    selectedColNames ,appendCols ,selectedPartitions = None, loop = 100,
    epsilon = 0.0001, initCenterSampleSize = 10):
```

参数:

- inputTableName: 输入表名
- centerCount: 聚类数
- resultTableName: 聚类结果表, 会包含保留列与聚类结果以及到每个聚类的概率
- centerTableName: 输出聚类质心表, 每行数据包含了一个聚类的信息, 包括聚类编号, 聚类权重以及聚类中心点
- selectedColNames: 输入表选中列, 只接受数值类型
- appendCols: 输出至聚类结果表的附加列
- selectedPartitions: (可选) 输入表选择分区, 默认为全选
- loop: (可选) 计算最大迭代次数, 默认为 100
- epsilon (可选) 计算最小迭代精度, 默认为 0.0001, 会与损失函数的变化率对比, 当损失函数的变化率小于 epsilon 时, 迭代停止

- initCenterSampleSize (可选) 初始化聚类中心点时的抽样大小, 默认 10, 会根据抽到的样本计算初始聚类中心点

例子:

```
Cluster.emcluster("iris", 2, "resultTable", "centerTable",
['x1', 'x2', 'x3', 'x4'], ['y'])
```

10.3.2 界面

Em Cluster 界面参数如下图: 点击分析, EM 聚类



其中最上层为选择输入表的 partition; 下方输入表选中列为选择参与 emcluster 计算的数值列, 只支持 double 类型的特征列; 聚类结果表附加列会保留在结果输出表中; 聚类数为用户指定的聚类数目; 初始质心抽样数为在初始化每个聚类时进行抽样的样本条数; 最大迭代次数, 达到此数目时停止迭代; 迭代终止精度:当损失函数变化值得百分比小于此数时, 停止迭代; 聚类结果表: 输出的聚类结果, 包含每行数据到各个聚类的概率和每行数据最终被分到哪一个聚类的信息; 聚类质心表: 每个聚类的质心信息;

例子 输入:这里我们采用了 iris 数据集进行实例, 数据如下

The screenshot shows a window titled "iris - 数据表浏览" (Iris Data Table Browser). The window contains a table with 150 rows of data. The columns are labeled: sepal_length, sepal_width, petal_length, petal_width, and category. The "category" column shows three distinct values: "Iris Setosa" (the vast majority), "Iris Versicolor", and "Iris Virginica". The table has scroll bars on the right and bottom. At the bottom of the window, there is a toolbar with buttons for "Whole Table" (with a dropdown menu), "位置" (Position), "数据大小: 150 行, 当前显示前" (Data Size: 150 rows, Current Display Before), and a numeric input field set to "5000".

	sepal_length	sepal_width	petal_length	petal_width	category
0	5.1	3.5	1.4	0.2	Iris Setosa
1	4.9	3	1.4	0.2	Iris Setosa
2	4.7	3.2	1.3	0.2	Iris Setosa
3	4.6	3.1	1.5	0.2	Iris Setosa
4	5	3.6	1.4	0.2	Iris Setosa
5	5.4	3.9	1.7	0.4	Iris Setosa
6	4.6	3.4	1.4	0.3	Iris Setosa
7	5	3.4	1.5	0.2	Iris Setosa
8	4.4	2.9	1.4	0.2	Iris Setosa
9	4.9	3.1	1.5	0.1	Iris Setosa
10	5.4	3.7	1.5	0.2	Iris Setosa
11	4.8	3.4	1.6	0.2	Iris Setosa
12	4.8	3	1.4	0.1	Iris Setosa
13	4.3	3	1.1	0.1	Iris Setosa
14	5.8	4	1.2	0.2	Iris Setosa
15	5.7	4.4	1.5	0.4	Iris Setosa
16	5.4	3.9	1.3	0.4	Iris Setosa
17	5.1	3.5	1.4	0.3	Iris Setosa
18	5.7	2.8	1.7	0.3	Iris Versicolor
19	6.4	3.9	5.1	1.5	Iris Virginica
20	7.0	3.4	6.4	2.0	Iris Virginica
21	7.9	3.8	6.5	2.4	Iris Virginica
22	4.3	3.0	1.0	0.2	Iris Setosa
23	5.8	4.0	1.2	0.2	Iris Setosa
24	7.7	3.8	6.7	2.6	Iris Virginica
25	6.9	3.5	5.4	1.6	Iris Virginica
26	5.4	3.4	1.7	0.2	Iris Setosa
27	7.9	3.8	6.4	2.0	Iris Virginica
28	4.9	3.4	1.4	0.2	Iris Setosa
29	6.4	3.2	4.5	1.5	Iris Virginica
30	4.3	3.0	1.3	0.1	Iris Setosa
31	6.9	3.1	5.1	1.8	Iris Virginica
32	5.1	3.5	1.7	0.3	Iris Setosa
33	7.7	3.0	6.3	2.3	Iris Virginica
34	6.4	3.6	4.3	1.5	Iris Virginica
35	4.8	3.4	1.6	0.2	Iris Setosa
36	7.2	3.5	5.5	1.5	Iris Virginica
37	5.7	3.0	4.2	1.3	Iris Virginica
38	6.3	3.4	4.4	1.3	Iris Virginica
39	4.9	3.1	1.8	0.1	Iris Setosa
40	6.9	3.1	5.1	1.8	Iris Virginica
41	5.6	3.0	4.6	1.5	Iris Virginica
42	6.5	3.0	4.5	1.5	Iris Virginica
43	4.9	3.1	1.8	0.1	Iris Setosa
44	7.0	3.0	5.5	2.0	Iris Virginica
45	5.4	3.4	4.5	1.5	Iris Virginica
46	6.9	3.1	5.1	1.8	Iris Virginica
47	5.7	3.0	4.6	1.4	Iris Virginica
48	6.4	3.1	4.7	1.4	Iris Virginica
49	4.8	3.0	1.4	0.1	Iris Setosa
50	6.8	3.0	5.1	1.9	Iris Virginica
51	5.2	3.4	4.7	1.4	Iris Virginica
52	6.7	3.1	5.2	1.5	Iris Virginica
53	5.1	3.3	4.5	1.5	Iris Virginica
54	6.9	3.1	5.4	1.5	Iris Virginica
55	5.4	3.0	4.5	1.5	Iris Virginica
56	6.5	3.0	5.0	1.6	Iris Virginica
57	4.9	3.1	1.4	0.1	Iris Setosa
58	7.1	3.0	5.9	2.1	Iris Virginica
59	5.8	3.0	4.9	1.8	Iris Virginica
60	6.3	3.0	5.6	1.8	Iris Virginica
61	4.8	3.0	1.4	0.1	Iris Setosa
62	7.2	3.0	5.8	1.6	Iris Virginica
63	5.7	3.0	5.1	1.8	Iris Virginica
64	6.4	3.0	5.4	1.7	Iris Virginica
65	4.9	3.0	1.4	0.1	Iris Setosa
66	7.0	3.0	5.5	1.8	Iris Virginica
67	5.5	3.0	4.9	1.7	Iris Virginica
68	6.6	3.0	5.0	1.8	Iris Virginica
69	4.8	3.0	1.4	0.1	Iris Setosa
70	7.3	3.0	6.3	1.5	Iris Virginica
71	6.0	3.0	5.1	1.8	Iris Virginica
72	5.9	3.0	5.4	1.7	Iris Virginica
73	4.8	3.0	1.4	0.1	Iris Setosa
74	7.1	3.0	5.9	1.8	Iris Virginica
75	6.3	3.0	5.6	1.8	Iris Virginica
76	4.8	3.0	1.4	0.1	Iris Setosa
77	7.2	3.0	5.8	1.5	Iris Virginica
78	6.1	3.0	5.4	1.7	Iris Virginica
79	5.9	3.0	5.1	1.8	Iris Virginica
80	4.8	3.0	1.4	0.1	Iris Setosa
81	7.0	3.0	5.5	1.8	Iris Virginica
82	6.3	3.0	5.4	1.7	Iris Virginica
83	5.8	3.0	5.1	1.8	Iris Virginica
84	6.4	3.0	5.3	1.7	Iris Virginica
85	4.8	3.0	1.4	0.1	Iris Setosa
86	7.1	3.0	5.9	1.8	Iris Virginica
87	6.5	3.0	5.5	1.7	Iris Virginica
88	5.9	3.0	5.1	1.8	Iris Virginica
89	4.8	3.0	1.4	0.1	Iris Setosa
90	7.2	3.0	5.8	1.5	Iris Virginica
91	6.4	3.0	5.4	1.7	Iris Virginica
92	5.8	3.0	5.1	1.8	Iris Virginica
93	6.5	3.0	5.3	1.7	Iris Virginica
94	4.8	3.0	1.4	0.1	Iris Setosa
95	7.3	3.0	6.0	1.8	Iris Virginica
96	6.6	3.0	5.5	1.7	Iris Virginica
97	5.8	3.0	5.1	1.8	Iris Virginica
98	6.3	3.0	5.4	1.7	Iris Virginica
99	4.8	3.0	1.4	0.1	Iris Setosa
100	7.0	3.0	5.5	1.8	Iris Virginica
101	6.4	3.0	5.4	1.7	Iris Virginica
102	5.9	3.0	5.1	1.8	Iris Virginica
103	4.8	3.0	1.4	0.1	Iris Setosa
104	7.1	3.0	5.9	1.8	Iris Virginica
105	6.5	3.0	5.5	1.7	Iris Virginica
106	5.9	3.0	5.1	1.8	Iris Virginica
107	4.8	3.0	1.4	0.1	Iris Setosa
108	7.2	3.0	5.8	1.5	Iris Virginica
109	6.6	3.0	5.4	1.7	Iris Virginica
110	5.8	3.0	5.1	1.8	Iris Virginica
111	6.3	3.0	5.3	1.7	Iris Virginica
112	4.8	3.0	1.4	0.1	Iris Setosa
113	7.3	3.0	6.0	1.8	Iris Virginica
114	6.7	3.0	5.5	1.7	Iris Virginica
115	5.8	3.0	5.1	1.8	Iris Virginica
116	6.2	3.0	5.4	1.7	Iris Virginica
117	4.8	3.0	1.4	0.1	Iris Setosa
118	7.0	3.0	5.5	1.8	Iris Virginica
119	6.5	3.0	5.4	1.7	Iris Virginica
120	5.9	3.0	5.1	1.8	Iris Virginica
121	4.8	3.0	1.4	0.1	Iris Setosa
122	7.1	3.0	5.9	1.8	Iris Virginica
123	6.6	3.0	5.5	1.7	Iris Virginica
124	5.9	3.0	5.1	1.8	Iris Virginica
125	6.3	3.0	5.3	1.7	Iris Virginica
126	4.8	3.0	1.4	0.1	Iris Setosa
127	7.2	3.0	5.8	1.5	Iris Virginica
128	6.7	3.0	5.4	1.7	Iris Virginica
129	5.8	3.0	5.1	1.8	Iris Virginica
130	6.2	3.0	5.3	1.7	Iris Virginica
131	4.8	3.0	1.4	0.1	Iris Setosa
132	7.3	3.0	6.0	1.8	Iris Virginica
133	6.8	3.0	5.5	1.7	Iris Virginica
134	5.9	3.0	5.1	1.8	Iris Virginica
135	6.3	3.0	5.4	1.7	Iris Virginica
136	4.8	3.0	1.4	0.1	Iris Setosa
137	7.0	3.0	5.5	1.8	Iris Virginica
138	6.6	3.0	5.4	1.7	Iris Virginica
139	5.9	3.0	5.1	1.8	Iris Virginica
140	6.3	3.0	5.3	1.7	Iris Virginica
141	4.8	3.0	1.4	0.1	Iris Setosa
142	7.1	3.0	5.9	1.8	Iris Virginica
143	6.7	3.0	5.5	1.7	Iris Virginica
144	5.9	3.0	5.1	1.8	Iris Virginica
145	6.3	3.0	5.3	1.7	Iris Virginica
146	4.8	3.0	1.4	0.1	Iris Setosa
147	7.2	3.0	6.0	1.8	Iris Virginica
148	6.8	3.0	5.5	1.7	Iris Virginica
149	5.9	3.0	5.1	1.8	Iris Virginica
150	6.3	3.0	5.4	1.7	Iris Virginica

iris 以鸢尾花的特征作为数据来源，数据集包含 150 个数据集，分为 3 类，每类 50 个数据，每个数据包含 4 个属性，该数据集数据分布情况如下，可以看到在二维关系下，setosa 较为 独立，versicolor, virginica 中间有部分交杂情况。 (http://en.wikipedia.org/wiki/Iris_flower_data_set)

输出：1. 聚类结果表：

EM聚类结果表

	x1	x2	x3	x4	y	xlab_cluster...	prob_of_clu...	prob_of_clu...
0	5.1	3.5	1.4	0.2	1	0	1	4.275253...
1	4.9	3	1.4	0.2	1	0	1	1.897055...
2	4.7	3.2	1.3	0.2	1	0	1	6.196910...
3	4.6	3.1	1.5	0.2	1	0	1	7.075167...
4	5	3.6	1.4	0.2	1	0	1	1.201840...
5	5.4	3.9	1.7	0.4	1	0	1	4.467885...
6	4.6	3.4	1.4	0.3	1	0	1	8.381722...
7	5	3.4	1.5	0.2	1	0	1	2.729306...
8	4.4	2.9	1.4	0.2	1	0	1	1.498623...
9	4.9	3.1	1.5	0.1	1	0	1	9.053640...
10	5.4	3.7	1.5	0.2	1	0	1	4.364800...
11	4.8	3.4	1.6	0.2	1	0	1	6.024274...
12	4.8	3	1.4	0.1	1	0	1	1.857584...
13	4.3	3	1.1	0.1	1	0	1	4.883405...
14	5.8	4	1.2	0.2	1	0	1	2.279069...
15	5.7	4.4	1.5	0.4	1	0	1	3.146877...
16	5.4	3.9	1.3	0.4	1	0	1	6.459428...
17	5.1	3.5	1.4	0.3	1	0	1	1.580096...
18	5.7	3.8	1.7	0.3	1	0	1	1.861361...
19	5.1	3.8	1.5	0.3	1	0	1	1.602100...
20	5.4	3.4	1.7	0.2	1	0	1	2.874847...
21	5.1	3.7	1.5	0.4	1	0	1	9.567636...
22	4.6	3.6	1	0.2	1	0	1	4.345120...
23	5.1	3.3	1.7	0.5	1	0	1	7.066928...
24	4.8	3.4	1.9	0.2	1	0	1	4.161408...
25	5	3	1.6	0.2	1	0	1	3.207099...
26	5	3.4	1.6	0.4	1	0	1	2.996477...
27	5.2	3.5	1.5	0.2	1	0	1	1.129079...
28	5.2	3.4	1.4	0.2	1	0	1	1.998205...
29	5	3.2	1.6	0.2	1	0	1	2.574000...

Whole Table ▾ 位置 数据大小: -- 行, 当前显示前 50 行 (最多10000行) GO!

2, 聚类质心表:

EM聚类质心信息表

	cluster_id	cluster_weight	mu_x1	mu_x2	mu_x3	mu_x4
0	0	0.66667166318...	6.26198680300...	2.8719957529...	4.90597298450...	1.6759896833...
1	1	0.33332833681...	5.00600756744...	3.4180166786...	1.46400243741...	0.2439991685...

第 11 章

关联分析

关联式规则(Association Rules, AR)，又称关联规则，是数据挖掘的一个重要课题，用于从大量数据中挖掘出有价值的数据项之间的相关关系。关联规则解决的常见问题如：“如果一个消费者购买了产品A，那么他有多大机会购买产品B？”以及“如果他购买了产品C和D，那么他还将购买什么产品？”正如大多数数据挖掘技术一样，关联规则的任务在于减少潜在的大量杂乱无章的数据，使之成为少量的易于观察理解的静态资料。关联式规则多不考虑项目的次序，而仅考虑其组合。

关联规则一个经典的实例是购物篮分析(Market Basket Analysis)。超市对顾客的购买记录数据库进行关联规则挖掘，可以发现顾客的购买习惯，例如，购买产品X的同时也购买产品Y，于是，超市就可以调整货架的布局，比如将X产品和Y产品放在一起，增进销量。

11.1 函数

执行help(Association.analysis)可以查看帮助信息

```
def analysis(inputTableName, idxColName, sequenceColName, itemColName, outputTableName, frequentItemsetsTableName,
    minSupportCount = -1, minSupportPercent = 0.02, minConfidence = 0.05,
    maxItemInAssociation = 3, maxTransactionWindowLen = -1, maxRelation = -1)
```

参数：

- inputTableName: 输入表名
- idxColName: 输入表中id列名
- sequenceColName: 输入表中sequence列，指定后按照序列模式(sequence)规则生成

- itemColName: 输入表中item列
- outputTableName: 关联规则输出表
- frequentItemsetsTableName: 频繁项集输出表
- minSupportCount: (可选)最小支持度(个数), 小于该支持度的规则将被过滤, 默认为-1
- minSupportPercent: (可选)最小支持比(分数), 当minSupportCount小于等于0时生效, 默认为0.02
- minConfidence: (可选)最小置信度, 小于该置信度的规则将被过滤, 默认为0.05
- maxItemInAssociation: (可选)项集中最大item数, 大于该最大值的规则将被过滤, 如果为非正整数则表示无限制, 默认为3
- maxTransactionWindowLen: (可选)序列模式(sequence)下最大时序窗口长度, 如果sequence为时间类型则需要转换为对应时间差的毫秒级偏移量, 如果为非正整数则表示无限制, 默认为-1
- maxRelation: (可选)序列模式(sequence)下最长规则链长度, 大于该最大值的规则将被过滤, 如果为非正整数则表示无限制, 默认为-1

关联规则示例:

```
Association.analysis("association_input", "id", "", "item", "association_output", "output_frequent")
```

表示计算输入表为association_input, 以其中的id列为id列, item列为item列到association_output表内, 频繁项集到output_frequent表内

序列模式示例

```
Association.analysis("association_input", "id", "time", "item", "association_output", "output_frequent")
```

表示计算输入表为association_input, 以其中的id列为id列, time列为sequence列, item列为item列, 到association_output表内, 频繁项集到output_frequent表内

11.2 界面

关联规则挖掘界面分三个模块:

1. 输入输出模块 , 用于指定输入表中的:

- id列: 对应脚本参数idxColName
- target列: 对应脚本参数itemColName
- sequence列: 对应脚本参数sequenceColName
- 规则表名: 对应脚本参数outputTableName
- 频繁项级表名: 对应脚本参数frequentItemsetsTableName



2. 参数模块：

- 挖掘模式可以指定关联规则挖掘和序列模式挖掘。
- 当指定序列模式挖掘时，必须指定sequence列，对应参数模块可以编辑。
- 最小支持度可以按照事务总数2%，自制定最小支持度和最小支持个数，分别对应脚本参数minSupportCount，minSupport
- 规则中最大item数对应脚本参数maxItemInAssociation
- 最小置信度对应脚本参数minConfidence



3. 序列模式模块 :

- 最长规则链长度, 对应脚本参数maxRelation
- 时间窗口, 对应脚本参数maxTransactionWindowLen



11.3 例子

这里我们采用了一份市场销售记录数据集来进行演示。数据集如下：

The screenshot shows a window titled "market - 数据表浏览". The window contains a table with three columns: "customer", "product", and "time". The data consists of 30 rows, indexed from 0 to 29. The "customer" column contains IDs like "CSI0000" through "CSI0004". The "product" column lists various items such as "hering", "corned_b", "olives", "ham", "turkey", "bourbon", "ice_crea", "baguette", "soda", "heineken", "olives", "corned_b", "avocado", "cracker", "artichok", "heineken", "ham", "turkey", "sardines", "olives", "bourbon", "coke", "turkey", "ice_crea", "ham", "peppers", and "hering". The "time" column contains numerical values ranging from 0 to 6. At the bottom of the window, there are buttons for "Whole Table", "位置" (Position), "数据大小: 7007 行, 当前显示前 10000 行" (Data size: 7007 rows, currently displaying the first 10000 rows), and "GO!".

	customer	product	time
0	CSI0000	hering	0
1	CSI0000	corned_b	1
2	CSI0000	olives	2
3	CSI0000	ham	3
4	CSI0000	turkey	4
5	CSI0000	bourbon	5
6	CSI0000	ice_crea	6
7	CSI0001	baguette	0
8	CSI0001	soda	1
9	CSI0001	hering	2
10	CSI0001	cracker	3
11	CSI0001	heineken	4
12	CSI0001	olives	5
13	CSI0001	corned_b	6
14	CSI0002	avocado	0
15	CSI0002	cracker	1
16	CSI0002	artichok	2
17	CSI0002	heineken	3
18	CSI0002	ham	4
19	CSI0002	turkey	5
20	CSI0002	sardines	6
21	CSI0003	olives	0
22	CSI0003	bourbon	1
23	CSI0003	coke	2
24	CSI0003	turkey	3
25	CSI0003	ice_crea	4
26	CSI0003	ham	5
27	CSI0003	peppers	6
28	CSI0004	hering	0
...

当前表有三列，其中customer列标识用户id，product列标识商品名，time列标识用户购买时间。每行标识某用户在某时间内购买了某商品，输入表是一个典型的三元组格式数据。

首先我们进行不考虑时序关联规则挖掘，首先打开关联分析界面：

market - 数据表视图

customer	product	time
0 CST0000	ring	0
1 CST0000	corned_b	1
2 CST0000	ives	2
3 CST0000	m	3
4 CST0000	turkey	4
5 CST0000	bourbon	5
6 CST0000	ice_crea	6
7 CST0001	baguette	0
8 CST0001	soda	1
9 CST0001	hering	2
10 CST0001	cracker	3
11 CST0001	heineken	4
12 CST0001	olives	5
13 CST0001	corned_b	6
14 CST0002	avocado	0
15 CST0002	cracker	1
16 CST0002	artichok	2
17 CST0002	heineken	3
18 CST0002	ham	4
19 CST0002	turkey	5
20 CST0002	sardines	6
21 CST0003	olives	0
22 CST0003	bourbon	1
23 CST0003	coke	2
24 CST0003	turkey	3
25 CST0003	ice_crea	4
26 CST0003	ham	5
27 CST0003	peppers	6
28 CST0004	hering	0
...

定义参数如下：

1. 输入输出部分：



2. 参数部分：



点击确定进行计算，计算分为：

- 数据导入步骤
- 迭代计算频繁项级步骤
- 生成规则步骤
- 转换频繁项级步骤
- 每执行一步都会在下方的运行信息处显示：
- 执行结束后会跳出两张结果表

规则表如下图：

market_rules - 数据表视图						
relations	绩	support_by_percent	confidence_by_percent	transaction_count	rule	
0	3	1.02561475409...	4.695304695304695	50	baguette & bourbon => cracker	
1	3	1.02561475409...	4.695304695304695	9.631147540983607	cracker => baguette & bourbon	
2	3	1.17638888888...	5.4945054945054945	70.51282051282051	baguette & chicken => heineken	
3	3	1.17638888888...	5.4945054945054945	9.1666666666666666	heineken => baguette & chicken	
4	3	1.07085253456...	3.896103896103896	9.948979591836734	baguette => chicken & herring	
5	3	1.02983539094...	3.896103896103896	50	baguette & chicken => herring	
6	3	1.02983539094...	3.896103896103896	8.024691358024691	herring => baguette & chicken	
7	3	1.07085253456...	3.896103896103896	41.93548387096774	chicken & herring => baguette	
8	3	1.17060810810...	2.697302697302697	34.61538461538461	baguette & chicken => peppers	
9	3	1.17060810810...	2.697302697302697	9.121621621621621	peppers => baguette & chicken	
10	3	1.04054054054...	2.3976023976023977	30.76923076923077	baguette & chicken => sardines	
11	3	1.04054054054...	2.3976023976023977	8.108108108108109	sardines => baguette & chicken	
12	3	1.02561475409...	3.896103896103896	50	baguette & chicken => cracker	
13	3	1.02561475409...	3.896103896103896	7.991803278688526	cracker => baguette & chicken	
14	3	1.760366653091...	3.796203796203796	12.837837837837837	coke => baguette & ice_crea	
15	3	1.84132055378...	3.796203796203796	57.57575757575757	baguette & coke => ice_crea	
16	3	1.84132055378...	3.796203796203796	12.140575079872203	ice_crea => baguette & coke	
17	3	1.760366653091...	3.796203796203796	52.054794520547944	baguette & ice_crea => coke	
18	3	1.53716216216...	2.997002997002997	45.45454545454546	baguette & coke => sardines	
19	3	1.53716216216...	2.997002997002997	10.135135135135137	sardines => baguette & coke	
20	3	1.07184923439...	1.998001998001998	30.303030303030305	baguette & coke => turkey	
21	3	1.07184923439...	1.998001998001998	7.06713780918728	turkey => baguette & coke	
22	3	1.18861892583...	2.5974025974025974	6.649616368286446	corned_b => baguette & ham	
23	3	1.18861892583...	2.5974025974025974	46.42857142857143	baguette & ham => corned_b	
24	3	1.38065843621...	6.093906093906094	67.03296703296702	baguette & corned_b => herring	
25	3	1.38065843621...	6.093906093906094	12.551440329218105	herring => baguette & corned_b	
26	3	1.18918918918...	3.196803196803197	35.16483518483516	baguette & corned_b => peppers	
27	3	1.18918918918...	3.196803196803197	10.81081081081081	peppers => baguette & corned_b	
28	3	1.19460905349...	5.794205794205794	11.934156378600825	peppers => baguette & olives	
...

relation列表示规则中项级数;

- list列为改规则对应提升度;
- support_by_percentile为支持度;
- confidence_by_percentile为提升度;
- transaction_count为支持数;

rule为对应的规则:

- a & b =>c 表示a, b发生后发生c的情况;
- a=>b & c 表示a发生后发生b, c的情况。

我们对挖掘后的规则按照提升度先排序, 先把数据读取全部, 总共1476条规则, 点击lift表头, 看一下前几条数据:

market_rules - 数据表视图						
relations	绩	support_by_percent	confidence_by_percent	transaction_count	rule	
0	2	1.25125	36.5634365343656	75	366	cracker => heineken
1	2	1.25125	36.5634365343656	61	366	heineken => cracker
2	2	1.11080357142...	26.073926073926074	66.58163265306122	261	baguette => heineken
3	2	1.11080357142...	26.073926073926074	43.5	261	heineken => baguette
4	2	1.34830712788...	25.674325674325676	80.81761006289308	257	soda => heineken
5	2	1.34830712788...	25.674325674325676	42.83333333333333	257	heineken => soda
6	2	1.11474782275...	25.574425574425575	52.674897119341566	256	hering => olives
7	2	1.11474782275...	25.574425574425575	54.12262156448203	256	olives => hering
8	2	1.37842622990...	25.174825174825177	82.62295081967214	252	artichok => heineken
9	2	1.37842622990...	25.174825174825177	42.00000000000001	252	heineken => artichok
10	2	1.81905222187...	25.074925074925076	51.434426229508205	251	cracker => soda

Cracker是薄脆饼干，经常和Heineken一起被销售，其中置信度，支持度都很高，提升度也不错，因此可以将该规则作为推荐结果之一展示给用户，当用户购买了其中一个，推荐另外一个是一个不错的选择。

再看一下频繁项级表：

market_fsets - 数据表浏览	
表格	数据处理
itemset	supportcount
0 apples	314
1 artichok	305
2 ham	305
3 heineken	600
4 herring	486
5 ice_crea	313
6 olives	473
7 peppers	296
8 sardines	296
9 soda	318
10 steak	227
11 turkey	283
12 avocado	363
13 baguette	392
14 bordeaux	74
15 bourbon	403
16 chicken	315
17 coke	296
18 corned_b	391
19 cracker	488
20 apples&artichok	64
21 apples&ham	53
22 apples&heineken	105
23 apples&herring	157
24 apples&ice_crea	70
25 apples&olives	159
26 apples&peppers	130
27 apples&sardines	126
28 apples&soda	59
...	...

其中frequentItemset=k中存储的是频繁K项级的内容，itemset为频繁项，item之间以&分割，supportcount列为对应的支持度。

第 12 章

推荐算法

12.1 eTREC

eTREC是推荐系统中广泛使用的基于物品的协同过滤（item-based collaborative filtering）算法在MR上的高效实现(上亿的user和item矩阵可在20分钟左右计算完成)，支持常用的以及自定义的相似度计算方法。

目前应用在手机淘宝、PC淘宝首页及各行业、非搜广告、一淘和淘宝搜索等数十个上线场景中，是计算行为相关性的有效利器，大幅度提升了各业务指标。

该算法由一淘推荐团队研发，阿里妈妈推荐团队部分同学参与，后续会加入集团其他推荐团队的成果。

贡献者：

2012.6-7月，袁全调优ItemCF wbcos模型，设计并行化方案；人清做第一版实现

2012.8-9月，东流对eTREC实现进行算子抽象，重构第二版，即年底在集团发布的eTRECV1

2013.1-3月，叶完、三余进行第三版优化，引入ProtocolBuffer，加入算法增量更新，鸿博指导，杨焜优化流程，将核心步骤简化为2步

2013.12月，绍成将算法从云梯1迁移至ODPS

2014.6月，品七、舍神、绍成将eTREC集成进入XLib算法库

12.1.1 函数

执行`help(Recommendation, etrec)`可以查看帮助信息

```
def etrec(inputTableName, outputTableName,
          inputPartitions=None, inputColNames=None,
          outputPartition=None, operator='add',
          minUserBehavior=2, maxUserBehavior=500, topN=50,
          itemValueDelimiter=':', itemDelimiter=',',
          isAscii=False, similarityType='wbcosine',
          normExpression=None, dotExpression=None, similarityExpression=None,
          alpha=0.5, localWeight=1)
```

etrec接受一个两列或者三列的表输入，也就是inputTableName为两列或者三列，对应的inputColNames为两列或者三列也可。如果是三列的情况下，要求第三列为数值类型。

第一列对应user，第二列对应item，第三列对应payload。

输出表第一列对应main item，后续的列对应main item的topN个推荐item list。这个list以kv形式存储。两个kv之间用itemDelimiter分割，k-v之间用itemValueDelimiter分割。

参数：

- inputTableName： 输入表名。
- outputTableName： 输出表名。
- inputPartitions：（可选）。输入表对应的输入分区。选中全表为None。默认选中全表。
- inputColNames：（可选）。选中的输入列名。选中全表为None。默认选中全表。
- outputPartition：（可选）。输出到partition名。输出到表则为None。默认为None,
- operator：（可选）。当同一个user对应出现相同的item时发生的行为。
可以的取值为：“add”，“mul”，“min”，“max”，
分别对应取加和，乘积，最小值，最大值四种行为。默认为“add”。
- minUserBehavior：（可选）。当user对应的item list大小小于这个参数时被忽略，取值范围必须大于等于2。默认值为2。
- maxUserBehavior：（可选）。当user对应的item list大小大于这个参数时，取参数对应个item，取值范围为[2, 100000]。默认值为500。
- topN：（可选）。结果中main item对应相似度的top n个，取值范围为[1, 10000]。默认值为2000。
- itemValueDelimiter：（可选）。输出表中推荐item list中kv格式中，k和v之间的分隔符。默认为”：“。
- itemDelimiter：（可选）。输出表中推荐item list中kv格式中，kv之间的分隔符。默认为”，”。
- isAscii：（可选）。分隔符是否为ascii码。默认为False。如果为True，则itemValueDelimiter和itemDelimiter为ascii码的字符串，如itemDelimiter=”44”。

- similarityType: (可选)。相似度类型。默认为wbcosine。目前支持 wbcosine, jaccard, asymcosine的三种内置类型, 和用户自定义表达式userdefined类型。
- normExpression: (可选)。norm算子表达式。默认为空。当similarityType为userdefined类型时候生效。表达式预定义变量valueI, size: valueI为单个user的itemI对应的value值, size为同一个user对应item list的长度。
- dotExpression: (可选)。dot算子表达式。默认为空。当similarityType为userdefined类型时候生效。表达式中预定义变量valueI, valueJ, size: valueI为单个user的itemI对应的value值, valueJ为单个user的itemJ对应的value值, size为同一个user对应item list的长度。
- similarityExpression: (可选)。similarity算子表达式。默认为空。当similarityType为 userdefined类型时候生效。表达式中预定义变量sumNormI, sumNormJ, sumDotIJ: sumNormI为itemI对应norm值, sumNormJ为itemJ对应norm值, sumDotIJ为itemI和itemJ对应的 dot值。
- alpha: (可选)。默认为0.5。当similarityType为asymcosine类型时候生效。asymcosine的平滑因子的值。
- localWeight: (可选)。默认为1。当similarityType为asymcosine类型时候生效。asymcosine的权重指数。

示例:

```
inputTableName = "input_table"
#默认相似度计算
outputTableName = "%s_recommendation_result" % inputTableName
Recommendation.etrec(inputTableName, outputTableName, inputPartitions=None,
                      inputColNames=None, outputPartition=None, operator='add',
                      minUserBehavior=2, maxUserBehavior=500, topN=50)
#自定义相似度计算(夹角余弦)
outputTableName = "%s_recommendation_result" % inputTableName
Recommendation.etrec(inputTableName, outputTableName, inputPartitions=None,
                      inputColNames=None, outputPartition=None, operator='add',
                      minUserBehavior=2, maxUserBehavior=500, topN=50,
                      itemValueDelimiter = ":", itemDelimiter = ",",
                      isAscii = False,
                      similarityType = "userdefined",
                      normExpression = "valueI*valueI",
                      dotExpression = "valueI*valueJ",
                      similarityExpression = "sumDotIJ/(sqrt(sumNormI)*sqrt(sumNormJ))")
```

12.2 协同过滤-SVD

协同过滤-SVD, 是协同过滤的一种方法。

12.2.1 函数

执行`help(Recommendation.CollaborativeFiltting.SVD)`可以查看帮助信息

训练

```
def train(inputTripleTableName, inputColNames, coreRank, modelTableName, inputPartitions=None, tol=0.0001)
```

参数:

- `inputTripleTableName`: 输入多元组表名。
- `inputColNames`: 输入表列名列表, 第一个是行, 第二个是列, 第三个是值。
- `coreRank`: core的维度。
- `modelTableName`: 输出的模型表名。
- `inputPartitions`: (可选) 输入表的分区列表。
- `tol`: (可选) 容忍度。

返回:

- `SvdCfModel`类型, 表示svd模型

示例:

```
inputTableName = "movielens_ratings"
inputColNames = ["userid", "movieid", "rating"]
modelTableName = 'movielens_ratings_model'
Recommendation.CollaborativeFiltting.SVD.train(inputTableName, inputColNames, 5, modelTableName)
```

预测

```
predict(svdCfModel, predictTableName, outputTable, appendColNames=None, inputPartitions=None, outputPartition=None)
```

参数:

- `svdCfModel`: 模型。类型是`SvdCfModel`。
- `predictTableName`: 预测表。
- `outputTable`: 输出表。
- `appendColNames`: (可选) `append`的列名列表。
- `inputPartitions`: (可选) 输入表的分区列表。
- `outputPartition`: (可选) 输出表的分区。

返回:

- 无

示例:

```
modelTableName = 'movielens_ratings_model'
svdCfModel = Recommendation.CollaborativeFiltering.SVD.loadModel(modelTableName)
predictTableName = 'movielens_ratings'
outputTable = 'movielens_ratings_predict'
Recommendation.CollaborativeFiltering.SVD.predict(svdCfModel, predictTableName, outputTable)
```

推荐

```
recommend(svdCfModel, recommendUserTableName, userColName, topn, outputTableName)
```

参数:

- svdCfModel: 模型。类型是svdCfModel
- recommendUserTableName: 推荐表
- userColName: 推荐表中user的列名
- topn: 推荐的个数
- outputTableName: 输出表

返回:

- 无

示例:

```
modelTableName = 'movielens_ratings_model'
svdModel = Recommendation.CollaborativeFiltering.SVD.loadModel(modelTableName)
recommendUserTableName = 'movielens_user_table'
userColName = 'userid'
topn = 10
outputTableName = 'movielens_ratings_recommend'
Recommendation.CollaborativeFiltering.SVD.recommend(svdCfModel, recommendUserTableName, userColName, topn, outputTableName)
```

加载模型

```
loadModel(modelTableName)
```

参数:

- modelTableName: 模型表名

返回:

- SvdModel类型

是否是SVD模型

```
isModel(tableName)
```

参数:

- tableName: 表名

返回:

- 当table是SVD模型时, 返回true

12.3 协同过滤-ALS

协同过滤-ALS是协同过滤的一种方法, 包含训练, 预测和推荐几种方法。

12.3.1 函数

执行help(Recommendation.CollaborativeFiltting.ALS)可以查看帮助信息

训练

```
train(inputTripleTableName, inputColNames, coreRank, modelTableName, inputPartitions=None, maxIterNumber=5, tol=0.0001)
```

参数:

- inputTripleTableName: 输入表名
- inputColNames: 输入表列名列表, 第一个是行, 第二个是列, 第三个是值
- coreRank: core的维度
- modelTableName: 输出的模型表名
- inputPartitions: (可选) 输入表的分区列表
- maxIterNumber: (可选) 最大迭代次数
- tol: (可选) 容忍度

返回:

- AlsCfModel类型, 代表ALS模型

示例:

```
inputTableName = "movielens_ratings"
inputColNames = ["userid", "movieid", "rating"]
modelTableName = 'movielens_ratings_model'
Recommendation.CollaborativeFiltting.ALS.train(inputTableName, inputColNames, 5, modelTableName)
```

预测

```
predict(alsCfModel, predictTableName, outputTable, appendColNames=None, inputPartitions=None, outputPartition=None)
```

参数:

- alsCfModel: 模型。类型是AlsCfModel
- predictTableName: 预测表
- outputTable: 输出表
- appendColNames: (可选) append的列名列表
- inputPartitions: (可选) 输入表的分区列表
- outputPartition: (可选) 输出表的分区

返回: 无

示例:

```
modelTableName = 'movielens_ratings_model'
alsCfModel = Recommendation.CollaborativeFiltering.ALS.loadModel(modelTableName)
predictTableName = 'movielens_ratings'
outputTable = 'movielens_ratings_predict'
Recommendation.CollaborativeFiltering.ALS.predict(alsCfModel, predictTableName, outputTable)
```

推荐

```
recommend(alsCfModel, recommendUserTableName, userColName, topn, outputTableName)
```

参数:

- alsCfModel: 模型。类型是AlsCfModel
- recommendUserTableName: 推荐表
- userColName: 推荐表中user的列名
- topn: 推荐的个数
- outputTableName: 输出表

返回:

- 无

示例:

```
modelTableName = 'movielens_ratings_model'
alsCfModel = Recommendation.CollaborativeFiltering.ALS.loadModel(modelTableName)
recommendUserTableName = 'movielens_user_table'
```

```
userColName = 'userid'
topn = 10
outputTableName = 'movielens_ratings_recommend'
Recommendation.CollaborativeFiltting.ALS.recommend(alsCfModel, recommendUserTableName, userColName, topn, outputTableName)
```

加载模型

```
loadModel(modelTableName)
```

参数:

- modelTableName: 模型表名

返回:

- AlsCfModel

是否是ALS模型

```
isModel(tableName)
```

参数:

- tableName: 表名

返回:

- 当table是ALS模型时, 返回true

第 13 章

矩阵计算

在XLab中，矩阵计算主要支持这些运算：矩阵构造，获取矩阵信息(类型、行数、列数和非零元数目) 打印矩阵信息(矩阵类型、行数、列数和非零元数目)， 加减乘除运算，向量点积，矩阵和向量范数，矩阵的迹，矩阵转置，矩阵的svd分解。有关矩阵定义，请参考 [矩阵](#) 。

- 矩阵构造

```
a = Matrix("mat_a")
a = Matrix("table_a", partitions = ["ds=20140320", "ds=20140321", "ds=20140322"], colNames = ["col1", "col2", "col3"])
a = Matrix.rand(1000, 100)
a = Matrix.rand(1000, 100, 0.1)
```

- 获取矩阵信息(类型、行数、列数、非零元数目)

```
a.matrixType()
a.rowSize()
a.colSize()
a.nonZero()
```

- 打印矩阵信息(矩阵类型、行数、列数、非零元数目)

```
print a
```

- 将矩阵存取到表

```
a.saveTo("matrix_name")
```

- 加、减、乘、除运算

```
c = a + b
c = a - b
c = a*b
c = a*num
c = a/num
```

- 向量点积

```
num = Matrix.dot(v1, v2)
```

- 矩阵和向量范数

```
num = a.norm("1")
num = a.norm("2")
num = a.norm("Inf")
num = a.norm("fro")
```

- 向量范数

```
num = a.norm(p), p = '3', '4', '5', ...
num = a.norm("-Inf")
```

- 矩阵的迹

```
num = a.trace()
```

- 矩阵转置

```
c = a.transpose()
```

- 矩阵的svd分解

```
[u, s, v] = a.svd()
[u, s, v] = a.svd(10)
[u, s, v] = a.svd(10, 1.0e-04)
```

13.1 函数

矩阵运算支持以下函数, 具体使用方法, 可用help命令查看, 示例:

```
help(Matrix.rowSize)
help(Matrix.colSize)
help(Matrix.matrixType)
help(Matrix.transpose)
help(Matrix.rand)
help(Matrix.dot)
help(Matrix.svd)
```

```
help(Matrix.trace)
help(Matrix.norm)
help(Matrix.saveTo)
```

13.1.1 获取矩阵行数

```
def rowSize(self):
```

返回:

- 矩阵的行数

示例:

```
m = 10
n = 5
a = Matrix.rand(m, n)
## 打印结果为: 10
print a.rowSize()
```

13.1.2 获取矩阵列数

```
def colSize(self):
```

返回:

- 矩阵的列数

示例:

```
m = 10
n = 5
a = Matrix.rand(m, n)
## 打印结果为: 5
print a.colSize()
```

13.1.3 获取矩阵类型

```
def matrixType(self):
```

返回:

- 矩阵的类型; 稠密矩阵为”dense”，稀疏矩阵为”sparse”

示例1:

```
a = Matrix.rand(10, 5, 0.5)
## 打印结果为: "sparse"
print a.matrixType()
```

示例2:

```
b = Matrix.rand(10, 5)
## 打印结果为: "dense"
print a.matrixType()
```

13.1.4 获取矩阵非零元素数目

```
def nonZero(self):
```

返回:

- 矩阵的非零元素数目；稠密矩阵的非零元素数目等于行数乘列数；

示例1:

```
a = Matrix.rand(10, 5)
## 打印结果: 50
print a.nonZero()
```

示例2:

```
a = Matrix.rand(10, 5, 0.2)
## 打印结果: 10
print a.nonZero
```

13.1.5 矩阵转置

```
def transpose(self):
```

返回:

- 矩阵的转置阵

示例1:

```
a = Matrix.rand(10, 5)
b = a.transpose()
## 打印结果为: 5
print b.rowSize()
## 打印结果为: 10
print b.colSize()
```

示例2:

```
a = Matrix.rand(10, 5)
b = a.transpose()
c = b.transpose()
## a 和 c 相等
```

13.1.6 产生随机矩阵

```
def rand(m, n, ratio = -1):
```

参数:

- m: 矩阵行数
- n: 矩阵列数
- ratio: (可选) 矩阵的稀疏度, 也就是非零元素的比例; 默认情况下, 将产生稠密矩阵

返回:

- 随机产生的矩阵

示例1:

```
## 产生一个10行5列的稠密矩阵
a = Matrix.rand(10, 5)
```

示例2:

```
## 产生一个10行5列, 稀疏度为0.2的稀疏矩阵, 也就是只有20%的元素是非零的
a = Matrix.rand(10, 5, 0.2)
```

13.1.7 向量点积

```
def dot(a, b):
```

参数:

- a: 矩阵a, 列数必须是1
- b: 矩阵b, 列数必须是1

返回:

- 向量的点积值; 向量就是列数为1的矩阵;

示例:

```
## 随机产生一个10行的向量v1
v1 = Matrix.rand(10, 1)
## 随机产生另一个10行的向量v2
v2 = Matrix.rand(10, 1)
## 打印v1和v2的点积
print Matrix.dot(v1, v2)
```

13.1.8 矩阵的svd分解

```
def svd(self, k = -1, tol = 0):
```

参数:

- k: (可选) 部分奇异值分解中, 期望求解的top奇异组数目; 默认求解全部奇异组;
- tol: (可选) 部分奇异值分解中, 期望的误差精度; 默认为 $1.0e-06$

返回:

- 奇异组组成的矩阵数组

示例1:

```
a = Matrix("mat_a")
## 求矩阵a的全部奇异组
[u, s, v] = a.svd()
b = u*s*v.transpose()-a
## num 应该接近0, 因为  $u*s*v' = a$ 
num = b.norm('fro')
```

示例2:

```
## 从表名 mat_a 中获取的矩阵
a = Matrix("mat_a")
## 求矩阵a的top 10 奇异组
[u, s, v] = a.svd(10)
```

示例3:

```
## 求取 表mat_a中矩阵的 top 10 奇异组, 且期望精度为 $1.0e-04$ 
a = Matrix("mat_a")
[u, s, v] = a.svd(10, 1.0e-04)
```

13.1.9 求矩阵的迹

```
def trace(self):
```

返回:

- 矩阵的迹，也就是矩阵对角线元素之和；要求矩阵必须是方阵，也就是行数需要等于列数；

示例:

```
a = Matrix("mat_a")
## 打印矩阵a的迹
print a.trace()
```

13.1.10 向量或者矩阵的范数

```
def norm(self, normType)
```

参数:

- normType: 范数类型字符串；对于矩阵和向量，支持的范数类型有：“1”，“2”，“Inf”，“fro”；此外，对于向量，支持的范数类型还有：“-Inf”，“p”（ $p=3, 4, 5, \dots$ ）

返回:

- 矩阵或者向量(列数为1的矩阵)的指定类型的范数

示例1:

```
a = Matrix.rand(1000, 100)
[u, s, v] = a.svd(1)
## 打印结果近似为0，因为矩阵的2范数就是矩阵的最大奇异值
print s.trace() - a.norm('2')
```

示例2:

```
a = Matrix.rand(1000, 1)
# 打印结果近似为0，因为向量的Frobineous范数和2范数是相等的;
print a.norm('fro') - a.norm('2')
```

13.1.11 存取矩阵到表

```
def saveTo(self, tableName):
```

参数:

- tableName: 矩阵存储的目标表

示例:

```
a = Matrix.rand(1000, 1000)
## 将随机生成的一个1000x1000的矩阵, 存取到表 "mat_a"
a.saveTo("mat_a")
```

第 14 章

自然语言处理算法

目前XLib中支持的自然语言处理相关的模块有：阿里分词（AliWS），文本归一化（全半角，大小写，繁简，特殊字符转换），文本特征选择算法（信息增益，卡方检验，IDF）、TFIDF算法、过滤噪音、分隔字符串等。其中AliWS和文本归一化两个模块的核心代码是由一淘搜索研发事业部提供。

14.1 阿里分词（AliWS）

阿里分词（AliWS）由一淘搜索研发事业部研发，是服务于整个阿里巴巴集团的词法分析系统。

2014年8月，千决团队与XLab团队合作，易江燕和梓美、水德，将AliWS集成进XLib。

在XLib中，AliWS的版本为 v1.3.0.7，目前支持淘宝中文分词，分词类型为语义粒度。

AliWS的详细介绍，见链接：[AliWS](#)

14.1.1 函数

AliWS目前只有一个函数segment，具体使用方法，可用help命令查看，示例：

```
help(NLP.AliWS.segment)
```

分词

```
def segment(inputTableName, selectedColNames, outputTableName,
           inputPartitions = None, mParameters = None, outputPartition = None,
           appendColNames = None, outputColNames = None)
```

参数:

- inputTableName 输入表
- selectedColNames 待分词列, 仅支持String类型
- outputTableName 输出表
- inputPartitions (可选) 输入表分区信息
- mParameters (可选) 分词配置参数, 字符类型map
- outputPartition (可选) 输出表分区信息
- appendColNames (可选) 保留列名
- outputColNames (可选) 分词结果列, 默认为selectedColNames的列名加后缀”_seg”

Note:

mParameters的key包括下面几种:

- Tokenizer 分词器类型, 默认为TAOBAO_CHN, 淘宝中文分词, 目前仅支持此类型
- SegmentAlgorithm 分词算法 (CRF或者UNIGRAM), 默认为CRF
- EnableDfa 简单实体识别, 布尔类型, 默认为true
- EnablePerson 人名识别, 布尔类型, 默认为false
- EnableOrganization 机构名识别, 布尔类型, 默认为false
- EnableCorrect 是否使用纠错词典, 布尔类型, 默认为true
- EnablePos 是否词性标注, 布尔类型, 默认为false
- EnableTelephoneRetrieval 检索单元配置-电话号码识别, 布尔类型, 默认为true
- EnableTimeRetrieval 检索单元配置-时间号码识别, 布尔类型, 默认为true
- EnableDateRetrieval 检索单元配置-日期号码识别, 布尔类型, 默认为true
- EnableNumberLetterRetrieval 检索单元配置-数字字母识别, 布尔类型, 默认为true
- EnableChnNumMerge 中文数字检索单元合并: 中文数字合并为一个检索单元, 布尔类型, 默认为false, 例: false: [第|十|五] true: [第|十五]
- EnableNumMerge 普通数字检索单元合并: 数字合并为一个检索单元, 布尔类型, 默认为true, 例: false: [34|.|5] true: [34.5] 例外: 对于词典中存在的数字如3.4按照词典中的检索单元格式切分

- NumberSplitLength 检索单元数字串切分长度, 默认为0, 0 : 不对数字串进行长度切分; >0: 数字串按指定长度进行截断作为检索单元, 并标记为SEG_TOKEN_RETRIEVE_BASIC_SUB
- EnableChnTimeMerge 中文时间合并: 将中文时间合并为一个语义单元, 布尔类型, 默认为false, 例: false: [五分][二十秒] true: [五分二十秒]
- EnableChnDateMerge 中文日期合并: 将中文日期合并为一个语义单元, 布尔类型, 默认为false, 例: false: [三月][一日] true: [三月一日]
- EnableCrossSemanticRetrieval 跨语义检索单元, 布尔类型, 默认为true
- EnableNormalizedOutput 输出结果总是以小写, 半角形式, 布尔类型, 默认为false
- EnablePersonNameRetrieval 中国/日本人名检索单元切分形式, 取值[0, 1, 2]默认为1, 0: 没有检索单元; 1: 长于一个字的姓或名作为扩充检索单元给出 2: 按照姓和名拆分作为基本检索单元(不建议使用方式2, 容易引发不一致)
- EnableSemanticTagOutput 是否输出语义标签, 布尔类型, 默认为false
- SegSeparator 分词分隔符, 默认为空格” “
- PosSeparator 词性分隔符, 默认为” / ”
- SemSeparator 语义标签分隔符, 默认为” | ”
- SegTokenType 输出粒度(可选), 1 - 语义单元, 默认为1, 目前仅支持语义粒度

示例1:

```
Table.drop("cn_corpus_sogou_seg_out")
NLP.AliWS.segment("cn_corpus_sogou", ["content"], "cn_corpus_sogou_seg_out")
```

示例2:

```
inputTableName = "test_aliws_input_big"
outputTableName = "test_aliws_input_big_out"
Table.drop(outputTableName)
mParameters = dict()
mParameters['EnablePerson'] = true
mParameters['EnablePos'] = false
print mParameters
NLP.AliWS.segment(inputTableName, ["str1", "str2"], outputTableName, inputPartitions=['pt=000000', 'pt=000001'], mParameters= mParameters)
```

14.2 文本归一化 (Normalize)

文本归一化 (Normalize) 由一淘搜索研发事业部研发, 处理的内容包括: 全半角, 大小写, 繁简, 特殊字符转换。

2014年8月, 千决团队与XLab团队合作, 易江燕和梓美、水德, 将Normalize集成进XLib。

在XLib中, Normalize支持的类型有五种:

- L2U (英文小写转大写)
- U2L (英文大写转小写)
- S2D (全角转半角)
- T2S (中文繁体转简体)
- NORMAL_DIGIT_CHAR(特殊数字字符归一化, 如 I :1, ①:1; (1):1; 1.:1; (‐):‐ ; (a):a)

14.2.1 函数

函数normalize, 具体使用方法, 可用help命令查看, 示例:

```
help(NLP.normalize)
```

归一化

```
def normalize(inputTableName, selectedColNames, outputTableName, \
              inputPartitions = None, normalizeTypes = None, outputPartition = None,
              appendColNames = None, outputColNames = None):
```

参数:

- inputTableName 输入表
- selectedColNames 待normalize列, 仅支持String类型
- outputTableName 输出表
- inputPartitions (可选)输入表分区信息
- normalizeTypes (可选)normalize类型, string 数组, 取值为: L2U U2L S2D T2S NORMAL_DIGIT_CHAR, 默认为L2U S2D T2S NORMAL_DIGIT_CHAR这四种类型都处理
- outputPartition (可选)输出表分区信息
- appendColNames (可选)保留列名
- outputColNames (可选)normalize完的列名, 默认为selectedColNames的列名加后缀”_norm”

示例1:

```
Table.drop("cn_corpus_sogou_normalize_outed")
NLP.normalize("cn_corpus_sogou", ["content"], "cn_corpus_sogou_normalize_outed")
```

示例2:

```
inputTableName = "cn_corpus_sina"
selectedColNames = ["title", "content"]
outputTableName = "cn_corpus_sina_normalized"
```

```

inputPartitions = ['pt=000000', 'pt=000001']
normalizeTypes = ['L2U', 'S2D', 'T2S']
outputPartition = "pt=20140813"
appendColNames = ["docid", "category"]
outputColNames = ["title_nm", "content_nm"]

Table.drop(outputTableName)
NLP.normalize(inputTableName, selectedColNames, outputTableName, \
    inputPartitions = inputPartitions, normalizeTypes = normalizeTypes, outputPartition = outputPartition, \
    appendColNames = appendColNames, outputColNames = outputColNames)

```

14.3 过滤噪音 (FilterNoise)

FilterNoise是指将输入数据的某一列中包含噪音列过滤掉。

注：不保序。

14.3.1 函数

函数filterNoise，具体使用方法，可用help命令查看，示例：

```
help(NLP.filterNoise)
```

过滤噪音

```
def filterNoise(inputTableName, selectedColumnName, noiseTableName, noiseColumnName, outputTableName, \
    inputPartitions = None, noisePartitions = None, outputPartition = None, appendColNames = None):
```

参数：

- inputTableName 输入表
- selectedColumnName 待过滤列
- noiseTableName 噪音表名
- noiseColumnName 噪音列名
- outputTableName 输出表
- inputPartitions (可选) 输入表分区信息
- noisePartitions (可选) 噪音表分区信息
- outputPartition (可选) 输出表分区信息
- appendColNames (可选) 保留列名

示例1:

```
Table.drop("test_word_filtered")
NLP.filterNoise("test_word", "word", "stop_word_cn", "word", "test_word_filtered")
```

示例2:

```
Table.drop("test_word_filtered")
NLP.filterNoise("test_word", "word", "stop_word_cn", "word", "test_word_filtered", inputPartitions = ['pt=00004', 'pt=00003'],
```

14.4 分割单词串

在XLib中，分割单词串是指将分完词的字符串分割为一个个单词，这里有两种形式：三元组和多行，分隔符可指定。

- 分割单词串为三元组(splitWordsToTriple)

分割单词串为三元组是指将doc中的单词串分割word并统计word在doc中的count，比如：

输入doc 和title:

```
doc      title
0      亚马逊 手机 苹果 苹果 财富
```

输出三元组:

```
doc    word      count
0      亚马逊    1
0      手机      1
0      苹果      2
0      财富      1
```

- 分割单词串为多行(splitWordsToMultiRows)

分割单词串为多行是指将doc中单词串分割为多行，并保证word在原单词串中的序，比如：

输入doc 和title:

```
doc          title
0      亚马逊 手机 苹果 苹果 财富
```

输出两列多行:

```
doc    word
0      亚马逊
0      手机
0      苹果
0      苹果
0      财富
```

14.4.1 函数

分割单词串包括两个函数, `splitWordsToTriple`和`splitWordsToMultiRows`。具体使用方法, 可用`help`命令查看, 示例:

```
help(NLP.splitWordsToTriple)
help(NLP.splitWordsToMultiRows)
```

分割单词串为三元组(`splitWordsToTriple`)

```
def splitWordsToTriple(inputTableName, docColName, wordColNames, outputTableName, \
    separator = None, inputPartitions = None, outputPartition = None):
```

参数:

- `inputTableName` 输入表
- `docColName` doc列名
- `wordColNames` words字符串列名, 支持String数组
- `outputTableName` 输出表
- `separator` (可选)words分隔符, char
- `inputPartitions` (可选)输入表分区信息
- `outputPartition` (可选)输出表分区信息

示例1:

```
Table.drop("cn_corpus_sogou_seg_splitwords_out")
NLP.splitWordsToTriple("cn_corpus_sogou_seg", "docid", ["content_seg"], "cn_corpus_sogou_seg_splitwords_out")
```

示例2:

```
Table.drop("cn_corpus_sogou_seg_splitwords_out")
NLP.splitWordsToTriple("cn_corpus_sogou_seg", "docid", ["content_seg"], "cn_corpus_sogou_seg_splitwords_out", separator = "\\"
```

分割单词串为多行(`splitWordsToMultiRows`)

```
def splitWordsToMultiRows(inputTableName, docColName, wordColName, outputTableName, \
    separator = None, inputPartitions = None, outputPartition = None):
```

参数:

- `inputTableName` 输入表
- `docColName` doc列名

- wordColName words字符串列名
- outputTableName 输出表
- separator (可选)words分隔符, char
- inputPartitions (可选)输入表分区信息
- outputPartition (可选)输出表分区信息

示例1:

```
Table.drop("cn_corpus_sogou_seg_splitwordsmultirows_out")
NLP.splitWordsToMultiRows("cn_corpus_sogou_seg", "docid", "content_seg", "cn_corpus_sogou_seg_splitwordsmultirows_out")
```

示例2:

```
Table.drop("cn_corpus_sogou_seg_splitwordsmultirows_out")
NLP.splitWordsToMultiRows("cn_corpus_sogou_seg", "docid", "content_seg", "cn_corpus_sogou_seg_splitwordsmultirows_out", sepa
```

14.5 文本特征选择 (FeatureSelect)

这里的特征选择是指文本的特征选择，目前支持卡方检验和信息增益两种方法。

14.5.1 函数

包括两个函数chiSquare和infoGain，具体使用方法，可用help命令查看，示例：

```
help(NLP.FeatureSelect.chiSquare)
help(NLP.FeatureSelect.infoGain)
```

卡方检验

```
def chiSquare(inputTableName, docColName, categoryColName, wordColName, outputTableName, \
    inputPartitions = None, outputPartition = None)
```

参数：

- inputTableName 输入表
- docColName doc列名
- categoryColName 分类列名
- wordColName word列名，仅支持String类型
- outputTableName 输出表
- inputPartitions (可选)输入表分区信息

- outputPartition (可选)输出表分区信息

示例1:

```
NLP.FeatureSelect.chiSquare("cn_corpus_tan_tfidf_join", "docid", "category", "word", "cn_corpus_tan_tfidf_join_chisquare")
```

示例2:

```
Table.drop("cn_corpus_tan_tfidf_join_chisquare")
```

```
NLP.FeatureSelect.chiSquare("cn_corpus_tan_tfidf_join", "docid", "category", "word", "cn_corpus_tan_tfidf_join_chisquare", inpu
```

信息增益

```
def infoGain(inputTableName, docColName, categoryColName, wordColName, outputTableName, \
            inputPartitions = None, outputPartition = None):
```

参数:

- inputTableName 输入表
- docColName doc列名
- categoryColName 分类列名
- wordColName word列名, 仅支持String类型
- outputTableName 输出表
- inputPartitions (可选)输入表分区信息
- outputPartition (可选)输出表分区信息

示例1:

```
NLP.FeatureSelect.infoGain("cn_corpus_tan_tfidf_join", "docid", "category", "word", "cn_corpus_tan_tfidf_join_infogain")
```

示例2:

```
Table.drop("cn_corpus_tan_tfidf_join_infogain")
```

```
NLP.FeatureSelect.infoGain("cn_corpus_tan_tfidf_join", "docid", "category", "word", "cn_corpus_tan_tfidf_join_infogain", inpu
```

14.6 词频统计 (wordcount)

词频统计是指统计某一列单词的词频。

14.6.1 函数

wordCount, 具体使用方法, 可用help命令查看, 示例:

```
help(NLP.wordCount)
```

词频统计

```
def wordCount(inputTableName, wordColName, outputTableName, \
    countColName = None, inputPartitions = None, outputPartition = None):
```

参数:

- inputTableName 输入表
- docColName doc列名
- categoryColName 分类列名
- wordColName word列名, 仅支持String类型
- outputTableName 输出表
- inputPartitions (可选)输入表分区信息
- outputPartition (可选)输出表分区信息

示例1:

```
Table.drop("cn_corpus_sogou_seg_splitwords_wordcount_out")
```

```
NLP.wordCount("cn_corpus_sogou_seg_splitwords_out", "word", "cn_corpus_sogou_seg_splitwords_wordcount_out", countColName = "
```

示例2:

```
Table.drop("cn_corpus_sogou_seg_splitwords_wordcount_out")
```

```
NLP.wordCount("cn_corpus_sogou_seg_splitwords_out", "word", "cn_corpus_sogou_seg_splitwords_wordcount_out", countColName = "
```

14.7 权重计算 (Weight)

权重计算目前只包括TF-IDF。TF-IDF (term frequency - inverse document frequency) 是一种用于资讯检索与文本挖掘的常用加权技术。TF-IDF是一种统计方法, 用以评估一词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加, 但同时会随着它在语料库中出现的频率成反比下降。TF-IDF加权的各种形式常被搜索引擎应用, 作为文件与用户查询之间相关程度的度量或评级。除了TF-IDF以外, 互联网上的搜寻引擎还会使用基于连结分析的评级方法, 以确定文件在搜寻结果中出现的顺序。

详细介绍见链接: [TF-IDF](#)

14.7.1 函数

`tfidf`, 具体使用方法, 可用`help`命令查看, 示例:

```
help(NLP.wordCount)
```

TF-IDF

```
def tfidf(inputTableName, docColName, wordColName, countColName, outputTableName, \
           inputPartitions = None, outputPartition = None, outputColNames = None):
```

参数:

- inputTableName 输入表
- docColName doc列名
- wordColName word列名, 仅支持String类型
- countColName count列名, 仅支持数值类型
- outputTableName 输出表
- inputPartitions (可选)输入表分区信息
- outputPartition (可选)输出表分区信息
- outputColNames (可选)输出列名

示例1:

```
NLP.Weight.tfidf("cn_corpus_tan_splitwords", "docid", "word", "count", "cn_corpus_tan_splitwords_tfidf")
```

14.8 Word2Vec

Word2Vec是Google在2013年开源的一个将词表转为向量的算法, 其利用神经网络, 可以通过训练, 将词映射到K维度空间向量, 甚至对于表示词的向量进行操作还能和语义相对应, 由于其简单和高效引起了很多人的关注。

Google Word2Vec的工具包相关链接: <https://code.google.com/p/word2vec/>

当前版本由易江燕、石昊和舍神合作开发和优化, 云帝、明澄参与讨论和验证, 针对海量数据在MPI上的高效实现, 后续还会增加更多的研究成果。

14.8.1 函数

train, 具体使用方法, 可用help命令查看, 示例:

```
help(NLP.Word2Vec.train)
```

训练

```
def train(inputTableName, wordColName, outputTableName, \
    inputPartitions = None, inVocabularyTableName = None, inVocabularyPartitions = None, \
    layerSize = 100, cbow = 0, window = 5, minCount = 5, \
    hs = 1, negative = 0, sample = 0, alpha = 0.025, \
    iterTrain = 1, randomWindow = 1, \
    outVocabularyTableName = None, outVocabularyPartition = None, \
    outputPartition = None):
```

参数:

- inputTableName 输入表
- wordColName 单词列名, 单词列中每行为一个单词, 语料中换行符用</s>表示
- outputTableName 输出表
- inputPartitions (可选)输入表分区信息
- inVocabularyTableName (可选)输入词表
- inVocabularyPartitions (可选)输入词表分区
- layerSize (可选)单词的特征维度, 默认为100, 区间为(0, 1000]
- cbow (可选)语言模型, 值为1: 表示cbow模型, 值为0: skip-gram模型, 默认为0
- window (可选)单词窗口大小, 默认为5
- minCount (可选)截断的最小词频, 默认为5
- hs (可选)是否采用HIERARCHICAL SOFTMAX, 值为1: 表示采用, 值为0: 不采用, 默认为1
- negative (可选)NEGATIVE SAMPLING, 值为0不可用, 建议值5-10, 默认为0
- sample (可选)向下采样阈值, 值为<=0: 不采用, 建议值为 $1e-3-1e-5$, 默认为0
- alpha (可选)开始学习速率, 默认0.025
- iterTrain (可选)训练的迭代次数, 默认为1, 取值区间 $>=1$
- randomWindow (可选)window是否随机, 值为1, 表示随机, 值为0, 表示不随机, 默认为1(1-5)
- outVocabularyTableName (可选)输出词表
- outVocabularyPartition (可选)输出词表分区
- outputPartition (可选)输出表分区信息

注解: 重要: 输入表为一列string类型的表, 每行写入一个单词, 换行符用</s>表示, 为了保证语义的上下文, 请使用XLab的分割单词串功能进行数据生成。

示例1:

```
inputTableName = "cn_corpus_sogou_seg_splitwordsmultirows"
wordColName = "word"
outputTableName = inputTableName + "_word2vec"
outputVocabName = inputTableName + "_vocab"

Table.drop(outputTableName)
Table.drop(outputVocabName)
NLP.Word2Vec.train(inputTableName, wordColName, outputTableName,
    inputPartitions = None,
    inVocabularyTableName = None, inVocabularyPartitions = None,
    layerSize = 100, cbow = 0, window = 5, minCount = 5,
    hs = 1, negative = 0, sample = 0, alpha = 0.025,
    iterTrain = 1, randomWindow = 1,
    outVocabularyTableName = outputVocabName,
    outVocabularyPartition = None,
    outputPartition = None)
```


第 15 章

有向图相关计算

有向图的相关定义，请参考 http://zh.wikipedia.org/wiki/%E5%9B%BE_%28%E6%95%B0%E5%AD%A6%29。

在XLab中，有向图底层是由稀疏矩阵来存储的：稀疏矩阵中的一个非零元(`row, col, value`)，对应图中的一条由编号`row`指向编号`col`，权重为`val`的边。支持的运算有：有向图的构造、获取有向图的信息、打印有向图的信息、计算PageRank。

- 有向图的构造

有向图有两种构造方式：

1. 通过三元组表来构造：需要给定边的起点列名、终点的列名，权重的列名（可选）
2. 通过稀疏矩阵与节点的索引表来构造

```
# 通过三元组来构造
G = DirectedGraph("small_graph_triples", sourceNodeIdColName="from_node", \
                   targetNodeIdColName="to_node")
G = DirectedGraph("small_graph_triples", sourceNodeIdColName="from_node", \
                   targetNodeIdColName="to_node", edgeValueColName="edge_weight")

# 通过稀疏矩阵来构造
G = DirectedGraph("graph_adjacent_matrix", nodeMappingTableName=None)
G = DirectedGraph("graph_adjacent_matrix", nodeMappingTableName="graph_node_mapping")
```

- 获取有向图的信息(节点数目，边的数目)

```
G.nodeSize()
G.edgeSize()
```

- 打印有向图的信息

```
print G
```

- 将有向图保存到表

```
G.saveTo("graph_adjacnet_matrix", "graph_node_mapping")
```

- 将有向图保存为三元组的形式（边的集合）

```
G.saveToEdgeList("graph_edge_list", sourceNodeIdColName="from_node", \
                  targetNodeIdColName="to_node")
G.saveToEdgeList("graph_edge_list", sourceNodeIdColName="from_node", \
                  targetNodeIdColName="to_node", edgeValueColName="edge_value")
```

15.1 函数

有向图支持以下函数，具体使用方法，可以用help命令查看，示例：

```
help(DirectedGraph)
help(DirectedGraph.nodeSize)
help(DirectedGraph.edgeSize)
help(DirectedGraph.saveTo)
help(DirectedGraph.saveToEdgeList)
help(DirectedGraph.LinkAnalysis.pageRank)
```

15.1.1 获取有向图的节点数目

```
def nodeSize(self):
```

返回：

- 有向图的节点数目

示例：

```
G = DirectedGraph("graph_adjacent_matrix", nodeMappingTableName="graph_node_mapping")
print G.nodeSize()
```

15.1.2 获取有向图的边的数目

```
def edgeSize(self):
```

返回:

- 有向图的边数目

示例:

```
G = DirectedGraph("graph_adjacent_matrix", nodeMappingTableName="graph_node_mapping")
print G.edgeSize()
```

15.1.3 保存为邻接矩阵形式

```
def saveTo(self, adjacentMatrixName, nodeMappingTableName):
```

参数:

- adjacentMatrixName: 邻接矩阵的表名
- nodeMappingTableName: 节点索引的表名

示例1:

```
G = DirectedGraph("triples", sourceNodeIdColName="source", \
                  targetNodeIdColName="destination", edgeValueColName="value")
G.saveTo("adjacent_mat", "node_mapping_tab")
```

示例2:

```
G = DirectedGraph("graph_adj_mat", nodeMappingTableName="graph_node_mapping")
G.saveTo("adjacent_adj_mat_new", "graph_node_mapping_new")
```

15.1.4 保存为三元组（边的集合）的形式

```
def saveToEdgeList(self, edgeListTableName, \
                  sourceNodeIdColName = "source_node", \
                  targetNodeIdColName = "target_node", \
                  edgeValueColName = "edge_value"):
```

参数:

- edgeListTableName: 三元的表名
- sourceNodeIdColName: (可选, 默认为"source_node")边的起点的列名
- targetNodeIdColName: (可选, 默认为"target_node")边的终点的列名

- edgeValueColName: (可选, 默认为” edge_value ”)边的权重的列名

示例1:

```
# 加载图
G = DirectedGraph("graph_adj_mat", nodeMappingTableName="graph_node_mapping")
# 保存 (不保存边的权值)
G.saveTo("graph_edge_list", sourceNodeIdColName="source", \
          targetNodeIdColName="target", edgeValueColName=None)
```

示例2:

```
# 加载图
G = DirectedGraph("graph_adj_mat", nodeMappingTableName="graph_node_mapping")
# 保存
G.saveTo("graph_edge_list", sourceNodeIdColName="source", \
          targetNodeIdColName="target", edgeValueColName="weight")
```

15.2 链接分析算法

15.2.1 PageRank算法

PageRank算法是Google创始人拉里·佩奇和谢尔盖·布林于1997年构建早期的搜索系统原型时提出的链接分析算法。它是基于随机冲浪模型的最好典范, 主要思想是模拟一个悠闲的上网者, 上网者首先随机选择一个网页打开, 然后在这个网页上呆了几分钟后, 跳转到该网页所指向的链接, 这样无所事事、漫无目的地在网页上跳来跳去, PageRank就是估计这个悠闲的上网者分布在各个网页上的概率。若对算法原理感兴趣的话可以参考(这个有一篇文章, 详细介绍了PageRank计算过程): <<http://www.ams.org/samplings/feature-column/fcarc-pagerank>>

PageRank对应的函数为 `DirectedGraph.LinkAnalysis.pagerank`。

具体使用方法, 可用`help`命令查看, 示例:

```
help(DirectedGraph.LinkAnalysis.pagerank)
```

接口:

```
class LinkAnalysis:
    @staticmethod
    def pagerank(graph, prTableName, alpha=0.85, epsilon=1e-6, maxIter=100):
```

参数:

- graph: 有向图
- prTableName: 输出表
- alpha: (可选) 阻尼系数(damping factor), 默认0.85

- epsilon: (可选) 收敛误差, 默认值为 $1e-6$
- maxIter: (可选) 最大迭代次数, 默认值为100

示例:

```
G = DirectedGraph("directed_graph_small_triples", sourceNodeIdColName="source", \
                  targetNodeIdColName="destination", edgeValueColName="value")
DirectedGraph.LinkAnalysis.pageRank(G, "pageRank_out_table", alpha=0.85)
```


第 16 章

张量模型

16.1 GigaTensor

16.1.1 函数

GigaTensor 包含了train, predict, isModel, loadModel四个函数，这几个函数的使用方法可以使用help命令查看：

```
help(Tensor.GigaTensor.train)
help(Tensor.GigaTensor.predict)
help(Tensor.GigaTensor.isModel)
help(Tensor.GigaTensor.loadModel)
```

训练

```
def train(inputTupleTable, modeColNames, valColName, coreRank,
          modelTableName, maxIterNumber=5, epsilon=1e-4,
          initUTable=None):
```

Args:

- inputTupleTable: 训练数据
- modeColNames: 训练数据每个维度的列名列表
- valColName: 观测值列名

- coreRank: core的维度
- modelTableName: 模型表名
- maxIterNumber: (可选)最大迭代次数
- epsilon: (可选)精度
- initUTable: (可选)初始化模型表名, 可以是GigaTensorModel, 也可以是DinTuckerModel

返回:

- GigaTensorModel, 代表GigaTensor的模型
- 给定GigaTensorModel model, 给定mode的列名(或者id)和输出表, 可以将每个mode的U矩阵存到表中,
- model.getMode(modeName, modeTableName)
- 给定输出表, 可以将lambda在面上的投影存到表中,
- model.getLambda(lambdaTableName)

示例:

```
modeColNames = ['mod1', 'mod2', 'mod3']
valColName = 'val'
gigaModel = Tensor.GigaTensor.train('trainTupleTable', modeColNames,
    valColName, 3, 'modelTableName')
print gigaModel
```

预测

```
def predict(model, inputTupleTable, outTableName, inputPartitions=None, outPartition=None, appendColNames=None)
```

参数:

- model: GigaTensorModel, 可以由train或者LoadModel获得
- inputTupleTable: 预测数据
- modelTableName: 模型表名
- outTableName: 输出表
- inputPartitions: (可选)输入表的分区列表
- outPartition: (可选)输出表的分区
- appendColNames: (可选)append的列名列表

返回:

- 无

示例:

```
model = Tensor.GigaTensor.LoadModel('modelTableName')
modeColNames = ['mod1', 'mod2', 'mod3']
valColName = 'val'
Tensor.GigaTensor.predict(model, 'modelTableName', 'outTableName', appendColNames=['id'])
```

加载模型

```
loadModel(modelTableName)
```

参数:

- modelTableName: 模型表名

返回:

- GigaTensorModel

是否是GigaTensor模型

```
isModel(tableName)
```

参数:

- tableName: 表名

返回:

- 当table是GigaTensor模型时, 返回true

16.2 DinTucker

16.2.1 函数

DinTucker 包含了train, predict, isModel, loadModel四个函数, 这几个函数的使用方法可以使用help命令查看:

```
help(Tensor.DinTucker.train)
help(Tensor.DinTucker.predict)
help(Tensor.DinTucker.isModel)
help(Tensor.DinTucker.loadModel)
```

训练

```
def train(inputTupleTable, modeColNames, valColName, coreRank, modelTableName, iterNum=1, emIterNum=15, learningRate=0.002,
```

参数:

- inputTupleTable: 训练数据
- modeColNames: 训练数据每个维度的列名列表
- valColName: 观测值列名
- coreRank: core的维度
- modelTableName: 模型表
- iterNum: (可选) job迭代次数, 默认1
- emIterNum: (可选)EM迭代次数, 默认15
- learningRate: (可选)学习速率, 默认0.002
- modelType: (可选)value列的类型 ‘binary’ 或者 ‘continuous’ , 默认‘binary’
- kernelPara: (可选)kernel的配置参数, 默认为空 本参数使用 key-value 对来表示不同的 kernel, 支持参数有: { ‘name’ : ‘linear’ } 表示为线性kernel { ‘name’ : ‘poly’ , ‘degree’ : 2} 表示多项式核函数 { ‘name’ : ‘rbf’ , ‘gamma’ : 0.01} 表示高斯核函数 { ‘name’ : ‘taper_matern’ , ‘taper’ : ‘spherical’ , ‘alpha’ : 1.0, ‘phi’ : 1.0, ‘theta’ : 1.0} 表示样条核函数 { ‘name’ : ‘taper_matern’ , ‘taper’ : ‘wendland1’ , ‘alpha’ : 1.0, ‘phi’ : 1.0, ‘theta’ : 1.0} { ‘name’ : ‘taper_matern’ , ‘taper’ : ‘wendland2’ , ‘alpha’ : 1.0, ‘phi’ : 1.0, ‘theta’ : 1.0} 每种核函数的具体形式, 可以参考: http://en.wikipedia.org/wiki/Gaussian_process

返回:

- DinTuckerModel, 代表DinTucker的模型
- 给定DinTuckerModel model, 给定mode的列名(或者id)和输出表, 可以将每个mode的U矩阵存到表中,
- model.getMode(modeName, modeTableName)

示例:

```
modeColNames = [‘mod1’, ‘mod2’, ‘mod3’]
valColName = ‘val’
coreRank = 3
model = Tensor.DinTucker.train(“tupleTable”, modeColNames, valColName, coreRank, ‘dintuckermodel’, iterNum=1)
print model
```

预测

```
predict(model, inputTupleTable, outTableName, emIterNum=20, nBagging=5, inputPartitions=None, outPartition=None, appendColNames=None)
```

函数:

- model: DinTuckerModel
- inputTupleTable: 训练数据
- outTableName: 预测输出表
- emIterNum: (可选)em迭代次数, 默认20
- nBagging: (可选)bagging个数, 默认5
- inputPartitions: (可选)输入表的分区列表, 默认None
- outPartition: (可选)输出分区, 默认None
- appendColNames: (可选)append的列名列表, 默认None

返回:

- 无

示例:

```
model = Tensor.DinTucker.loadModel('modelTableName')
Tensor.DinTucker.predict(model, 'predictTupleTable', 'predictOutTable', nBagging=5)
```

加载模型

```
loadModel(modelTableName)
```

参数:

- modelTableName: 模型表名

返回:

- DinTuckerModel

是否是DinTuckerTensor模型

```
isModel(tableName)
```

参数:

- tableName: 表名

返回:

- 当table是DinTuckerTensor模型时, 返回true

第 17 章

常见问题

17.1 XLab/XLib使用说明

Q: 如何在ODPS中的xlib下运行python脚本?

- A: 请查看ODPS文档 XLab/XLib->简介->脚本运行。

Q: Xlab. jar的源码在哪里?

- A: 目前暂不支持开源。

Q: 在XLab里面一次运行多个脚本程序可以吗?

- A: 目前不支持。

Q: Xlib怎么删除表格的一列?

- A: 不支持删除一列, 可以用sql按照要求重新create一张表。

Q: 是否支持逐个insert 操作?

- A: 目前不支持。

Q: XLab中是否支持UDF?

- A: 目前不支持。

Q: xlib是否提供SDK?

- A: 目前不提供。

Q: MapReduce中如何调用xlab中的模型? 看哪个文档?

- A: 目前xlib/xlab还没有开放sdk, 因此不支持这样调用。Xlib支持在ODPS客户端和XLab中调用。可以写脚本等待MR运行完后执行ODPS客户端的xlib算法。

Q: 请问有没有办法可以设置xlab任务的mapper个数?

- A: 不能设置, 由xlab任务决定mapper数。

Q: Xlab脚本语言中是否可以像MR一样设置mapper和reducer个数?

- A: 目前不支持。

Q: 在XLab中打开UTF-8(无BOM)文件, 总是乱码, 请问怎么解决?

- A: 目前请您直接开拷贝文件内容到Xlab, 下个版本支持。

Q: Xlib里的python脚本, 不能导入自己写的模块, 不管模块放在当前目录还是放在/python27/lib这是为什么?

- A: 您好, XLab暂不支持外部模块的import功能, 如果想执行您自己写python文件可以在Xlab里面用execfile函数。execfile这个函数是python的builtin函数, python的文档里可以找到说明。
- A: 示例: execfile('test.py'), 注test.py放在XLab目录下。

Q: 一个行语句在XLab客户端中太长了, 我用“”换行, 在XLab中可以运行, 但是odps -f filename 就不行。

- A: odps -f filename, filename里面的命令式逐条执行的, 所以续行符不可用。如果要执行xlab里保存的文件(包含续行符), 可以使用xlib的执行命令 execfile('test1.py')命令。

Q: ODPS客户端里面xlib的指令开头不能多加空格

- A: 依照python语法的缩进规范。

Q: Xlib模型输入表为空报错?

- A: 必须填写输入表。

Q: 误删了系统中的一个表 statistics_xlib_sys_unz9o7k56b1t5mks, 怎么办?

- A: 是Xlib临时表, 重跑就可以了。

17.2 表的操作

Q: XLab/XLib的表与ODPS表的关系?

- A: XLab/XLib的表即ODPS表。

Q: 在XLab中如何查看所有表?

- A: 请查看ODPS XLab/XLib->表的操作->显示指定工程下的所有表名
- A: help(Table.listTableNames)

Q: 如何获得Table元数据信息, 例如列的信息

- A: 请查看ODPS XLab/XLib->表的操作
- A: Help命令: help(Table)

Q: XLAB中如何对表格进行排序?

- A: 对于XLab来说, 排序可以使用文档中ODPS XLab/XLib->数据处理->排序
- A: 也可以使用文档中ODPS XLab/XLib->基本统计->多列排序和分位处理。
- A: 对于XLab的排序函数处理过的数据, 在XLab中显示的是前1W行。

17.3 数据处理

Q: 如何保序的合并两张表的两列/如何把结果表和原始表的ID合并起来, 不乱序?

- A: 使用数据处理的DataProc.appendColumns

Q: 怎么给一个表加一个递增的ID列, 来表示行号?

- A: 请参看ODPS文档XLab/XLib 数据处理->追加ID列(appendId) 函数实现, 用help命令查看使用帮助
help(DataProc.appendId)

Q: 如何构建一个所有entry全是0的矩阵?

- A: 请查看XLab/XLib文档->数据处理->数据生成。
- A: 生成稠密矩阵, 用help命令查看使用帮助, 生成稠密矩阵: help(DataGenerator.randomDense);
生成稀疏矩阵: help(DataGenerator.randomSparse)

Q: 随机采样是不是不考虑正负样本的比例的

- A: 是的, 完全随机, 目前不支持用户指定比例。

Q: XLab里随机采样doc里写的是默认不放回的吗?

- A: 单次采样不放回, 每次采样是各自独立的。权重值越大的被选中概率越大。

Q: 需要对每条正样本重复50次(重采样)请问有没有这样的现成的方法?

- A: 现在 odps 不支持你所说的重采样, 你可以试试加权采样: 在 xlab 中运行
help(DataProc.Sample.weightedSample)。

Q: Xlab中, 用fillmissingvalues函数填充Null值时, 怎么选择表中的所有列?

- A: 对于脚本: 您使用help(Table), 可以看到getColNames这个方法可以获取所有的列名, 通过for循环
可以将所有列放到configs里。对于fillmissingvalues, 目前还不支持configs[(all columns)]这种
语法。对于界面: 菜单->数据处理->缺失值填充 ->缺失值选项, 中间那一列按钮中有”>>”, 可以
直接选择所有列。

17.4 随机森林

Q: 随机森林MR job fail the data is empty, sort fail!

- A: 出现这个问题有两种可能：一是默认算法类型为混合，会出现上述问题，选择Cart算法即可解决此问题。二是在每棵树采样个数占总数据的比例很小的时候，会出现抽不到数据的情况，报出这种错误，请重新合理采样即可解决。

Q: 随机森林训练函数中的weightColName那一列的取值范围是多少？

- A: 大于0的都可以。

Q: 随机森林算法中，可以看到各个特征的重要性么？如何看到呢？

- A: 在模型显示的树上可以看到每一个结点的信息，但是不能看到重要性，如果要看重要性，两种途径：函数help(DataProc. InfoValue) 和界面，输入表不要包含string类型，XLab菜单->模型->属性选择。

Q: 随机森林的预测为什么总是到了49%就不动了？

- A: 随机森林算法具体的运行时间和输入的参数有关，设置输树的深度，减小每棵树的最大记录数，减少随机属性计算数目等均可以减少算法的运行时间。你这job目前是运行正常的，另外算计森林选择混合模式是有三种算法组成， $0 \sim 1/3$ 是id3， $1/3 \sim 2/3$ 是cart， $2/3 \sim 3/3$ 是c45，如果想缩短时间，可以看看49%完的是什么算法，然后只选择这一种算也是可以的。

Q: 想问下Xlib里面随机森林分类预测算法参数设置的问题，其中的minNumObj和minNumPer是如何作用的，如果根据当前的分裂属性，叶节点数据小于了这两个阈值，是直接进行剪枝操作，还是换一个随机属性重新进行分裂？

- A: minNumObj指的是叶结点的最小个数；minNumPer指的是叶结点数据个数占父节点的最小比例；叶节点数据小于了这两个阈值的一个，是直接进行剪枝操作，也就是说不会产生这个叶子节点。

Q: 在随机森林训练的过程中，有一个这个信息，请问是什么意思？每棵树最大记录数 是指每次建立一棵树的随机抽取的样本数，还是每棵树最多子负节点数？

- A: 每棵树最大记录数 是指采样的个数，在界面上限制范围是[1000, 1000000]，在脚本上限制是(0, 1000000]。

Q: 随机森林的如果采用验证表，这个验证集会不会对随机森林建树过程中进行参数调整啊？如果不影响，那么验证集和测试集又有什么区别？

- A: 不会。验证集会在树上显示其分类的过程；测试集是为了验证结果，根据测试集计算混淆矩阵和ROC曲线。

Q: 随机森林的algorithmTypes的设置方法没看懂。文档里的[2, 4]都是指位置？因为混合的树有些情况下树过多了会报错，之前工单说cart算法可以解决问题。用函数调用的话要怎么设置？

- A: 在一个拥有5棵树的森林中，[2, 4]表示0, 1为id3算法，2, 3为cart算法，4为c4.5算法。如果输入为None，则算法在森林中均分。如果都选cart可以写为[0, 5]。

Q: XLAB里面决策树的图在哪里可以看到？

- A: 打开模型表, 选择随机森林->查看。

Q: 随机森林预测为什么使用的资源多?

- A: 在随机森林预测中, 数据膨胀的倍数等于随机森林中树的棵树。最终的instance数目=树的颗数 * 单棵树处理所有预测数据需要的instance数目。

Q: 在随机森林预测中, 怎么输出概率?

- A: 随机森林预测现在只支持二分类情况下的概率输出。在随机森林预测界面中, 选中二分类, 同时填入要输出的分类的值, 在预测输出表中就可以看到第二列为输入分类值的概率。

Q: 随机森林训练中为什么会出现小于最小节点个数的节点?

- A: 在随机森林训练的时候, 当一个节点分裂时, 有小于二个不可分节点时, 这个节点是会分裂的。

Q: 随机森林中为什么会出现特征熵的增益率不是最大, 但是被选中的情况?

- A: 在随机森林C4.5算法中, 为了防止过拟合, 在选择特征的时候, 选择的是熵的增益大于平均增益中增益率最大的特征。

17.5 逻辑回归

Q: 逻辑回归中主分类是什么?

- A: 主分类就是要输出概率的分类。

Q: 逻辑回归参数goodValue是什么意思?

- A: GoodValue就是逻辑回归中的正例, 比如label列有1, 0两个值, 一般来说就把1作为正例。

Q: 逻辑回归回归正则项输出, 11正则项系数比较大, 是不是weight为0的个数会增多?

- A: 是的, 权重为0的不输出。

Q: 逻辑回归回归的时候, 能同时指定11系数和12系数吗?

- A: 不能, 只能二选一。

Q: 逻辑回归的参数(概率大于0.5预测值=1的那个0.5)可以调吗?

- A: 不可以, 但是可以通过其他方法进行操作:打开逻辑回归预测结果表, 选择数据处理->变量转换, 根据probability列(预测为正例的概率)生成新的结果列。
- A: 比如:case when probability>0.1 then 1 else 0 end 即可把0.1作为阈值, 重新生成结果列。

Q: 逻辑回归train函数出现 failed to match partition depth.

- A: 二分类逻辑回归, 需要填上goodValue的值, 不能是None。

17.6 线性回归

Q: 线性回归报错 “ matrix is singular”

- A: 矩阵必须是可逆的。

17.7 GBRT

Q: Xlab中GBRT算法训练的时候自变量稀疏矩阵如何得到?我之前是将训练表中特征用工具->普通表转为稀疏矩阵表转了之后再用GBRT训练算法进行训练,但是最终训练出来的模型都是空的,我想请问一下这个是不是那个稀疏矩阵的问题?

- A: 请确认算法输入稀疏矩阵是否为空。
- A: 算法是否运行成功, 如果运行成功, 模型一定不是空的, 至少会有一个节点。
- A: 请确认查看的模型表明是否是对的, 是否查看错了表明。

Q: 请问gbrt的predict输出的y_var是什么物理意义? 概率还是什么?

- A: 预测表里面的y_var是gbrt的回归值。类似于线性回归 ($y=ax+b$) , 用户输入值x, 根据model表中的线性系数a, b得到线性回归的结果y。在gbrt里面的y_var含义类似线性回归的y值, 区别是model表不是线性系数, 而是树了。

Q: 请问XLAB上面操作GBRT有没有更加信息的一些资料, 比如我的输入的预测表是怎么样的形式? 我看他是要稀疏矩阵的, 还是按预测表普通表来生成稀疏矩阵之后进行预测吗, 如果这样的话userid, brandid信息就带不上去了。还有最后生成的y_val是什么意思呢? 是他回归树的最终值吗? 关于gbrt这块你们在帮助文档上比lr, rf等模型少了太多信息, 能否更加信息一点呢, 谢谢!

- A: 第一个问题: 稀疏矩阵预测完可以使用DataProc.appendColumns将需要的列和预测结果表append到一起;
- A: 第二个问题: y_var是回归最终值;
- A: 第三个问题: 文档会改进, 增加更多关于输入和输出的说明。

Q: 在GBRT训练的时候出现Unknow Exception错误, 那个是训练表的数据, 1e条记录, 有300列, 我用它来做LR, SVM, RF都是没问题的, 这是怎么回事?

- A: 每个算法的实现是不一样的, gbdt的规模依赖于数据的行数, 行数越多, 后台占用资源越多。产生这个问题的原因是, gbdt在申请资源的时候没有申请足够的资源, 当资源使用超出申请的资源总量时, 算法会被停止。如果降低数据的行数, 能降低资源的使用, 算法才能跑过。下个版本会对这个问题优化。

Q: GBRT是否支持缺失值?

- A: 不支持。

Q: GBRT是否支持分类?

- A: 不支持。现在的每一棵树都是一棵回归树。

Q: GBRT损失函数是什么？

- A: 现在的GBRT只支持最小二乘损失函数。

17.8 关联规则

Q: 关联规则频繁项集rule表输出为空

- A: 关联规则rule默认生成为lift大于1的规则, 请检查频繁项级对应各个规则的lift度。

Q: 表里有记录 $700+w$ 条, 其中有 $340w+$ 个用户, $2w+$ 的品牌, 但是输出的频繁项集结果却只有一个, 而且支持度还很高?

- A: $340w+$ 个用户 如果最小支持度为2%支持度 计算大概支持数 $7W$ 以上才会被留下, 可以适当降低支持度再试试。

Q: 关联规则输出表 transaction_count 列是什么意思?

- A: transaction_count 是支持度计数, 表示支持该规则的事务数, 比如对应规则: $18158 \Rightarrow 16692 \ \& \ 21330$ 。transaction_count=1063, 这个1063是” $18158 \Rightarrow 16692 \ \& \ 21330$ ”出现了1063次

Q: 关联规则调用中序列必须是一个数字类型是吧?

- A: long和datetime都可以。

Q: 关联规则参数 @param maxTransactionWindowLen 是什么意思?

- A: 序列模式下最大的时序窗口长度(sequence), 如果sequence为时间类型则需要转换为对应时间差的毫秒级偏移量, 如果为非正整数则表示无限制。例子: 规则 $a \Rightarrow b \Rightarrow c$ 时间对应为0, 1, 2, 那么时序窗口长度就是 $2-0=2$, 如果这个值大于maxTransactionWindowLen, 则该规则被过滤掉。

Q: 关联规则任务启动之后, 可以关闭xlab吗?

- A: 不可以, 里面是分了几个job进行计算的, 关闭了流程就走不下去了。

17.9 矩阵计算

Q: 原来是用matlab写的程序, 绝大多数都是对矩阵循环。odps这一套里头有没有类似遍历表格这种的循环的比较直接的操作?

- A: Xlab中, 矩阵的数据是在云端的, 目前不支持对矩阵的进行读写操作。

Q: xlib里生成矩阵的函数Matrix怎么使用?

- A: 请参考ODPS文档XLab/XLib->矩阵计算, 也可以使用help命令, help(Matrix)。

Q: xlab做完svd之后返回的数据是存内存里么?

- A: 存在云端的表里。

Q: 请问有将矩阵分块拆开的函数吗?

- A: 目前不支持这样的函数, 需要自己拆分矩阵。

17.10 格式转换

Q: xlabs里面的getTripleMaps如果只指定行, 就出错。

- A: 目前必须同时指定行与, 五月底的下个版本会支持。

Q: 调用三元表转索引三元表的函数时, 报错。

- A: tripleWithMapToIndexedTriple 必须指定row map和 col map, 缺一不可。五月底的下个版本会支持的。

Q: 如何转稀疏矩阵?

- A:
- 可以使用普通表转矩阵

```
DataConvert.tableToSparseMatrix("appendinputleft", "appendinputleft_out", selectedColIndex=[0, 1, 2])
```

- 如果要转的稀疏矩阵的列数超过了普通表可以表示的最大列数的情况下使用Key-value转稀疏矩阵

```
DataConvert.KVToSparseMatrix("kv_table_name", "map_table_name", "out_matrix_name",
    kvMatColsName = ['col1'], colDelimiter = ",", valDelimiter = ":", isAscii = False)
```

17.11 基本统计

Q: 全表统计中的相关系数是用什么方法计算的?

- A: 皮尔逊相关系数。

17.12 SVM

Q: 请问ODPS提供线性SVM是基于核方法的吗?

- A: 是线性核并且目前只支持这一种方法。

17.13 聚类

Q: 请问ODPS XLab支持对稀疏矩阵进行聚类操作吗? 3万列能支持吗?

- A: 目前不支持, 只支持普通表的聚类, 普通表最大1024列, 支持不了3万列。

Q: 在XLab中使用kmeans聚类如何保留原表中的一些列?

- A: 可以通过 DataProc.appendColumns 将 index 表和原表进行列拼合。通过 help(DataProc.appendColumns) 可以查看具体用法。

Q: Kmeans三个输出表分别是什么意思?

- A: idxTableName: 存放每个点的聚类编号。一列, 列名为cluster_index, long型, 行数等于输入表总行数。每行的值表示输入表对应行表示的点的聚类编号
- A: centerTableName: 存储聚类质心位置。列数等于输入表(有选中则等于选中列总数) 行数等于聚类数, 每行表示一个聚类的最终的聚类质心位置。
- A: clusterCountTableName: 存储每个聚类中的点的总数。一列, 列名为cluster_count, long型, 行数等于聚类数, 每行表示centerTableName中对应行所表示的聚类质心的类中所包含的聚类点的总数。

17.14 分类评估

Q: xlab模型中的评估, 计算出的tp是错误的, 这是怎么回事? 小样本测试的时候又是正确的。

- A: 亲, 请确认下您计算的tp过程是否正确, 有可能是sql join条件不唯一数据扩充导致的。

17.15 排序

Q: ExceptionBase: sortrank only support sorting double cols!

- A: sort_rank -c指定的sort目前只支持double类型。如果只是需要求解分位数可以调用percentile函数。

Q: sort_rank行数上限是多少?

- A:
- 指定group字段时, 输入表规模上限为10亿行规模下能支持5个选择列(-c参数选择的列), 11亿行规模下能支持4个选择列, 15亿行规模下能支持3个选择列。
- 不指定group字段时, 10亿行规模下支持25个选择列, 20亿行规模下能支持12个选择列。
- XLab脚本调用是上面规模的两倍。

第 18 章

附录

18.1 附录 A 表达式支持的操作符、函数以及相应的优先级

- 所有的计算里面，如果出现类型不一致，不会抛出异常，而是结果返回NULL；
- xlib表达式部分，计算层对boolean, bigint, double的表示都是统一用double表示；
- 和odps-sql不一样的地方：xlib 的表达式部分，不会对字符串类型和数值类型(Boolean, long, double)进行相互的隐式类型转换；
- 所有计算过程中，odps-sql抛异常的地方，xlib这边是结果返回NULL。

18.1.1 表达式运算优先级

```
1: (), lengthb, length, lower, upper, trim, reverse, instr, left, right, repeat, substr, replace, insert, concat, conv, cosh, sinh, tanh, exp, rand, trunc, ln, char_matchcount, chr, is_encoding, keyvalue, md5, regexp_extract, regexp_instr, regexp_replace, regexp_substr, regexp_count, split_part, to_char, tolower, toupper, cast, dateadd, datediff, datepart, datetrunc, from_unixtime, getdate, to_date, unix_timestamp, weekday, weekofyear, lastday, decode, get_idcard_age, get_idcard_birthday, get_idcard_sex, unique_id, uuid, abs, log, log10, sqrt, sin, cos, tan, cot, asin, acos, atan, acot, ceil, floor, round, isdate, pow, case when
```

```
2: ^ (表示指数)
```

```
3: +(正), -(负)
```

```

4: *, /, %

5: +, -, (加、减)

6: <<, >> (左移位, 右移位)

7: <, <=, >, >=

8: ==(=), !=(<>), is null 和 is not null, in 或者 @

9: & (按位and)

10: | (按位or)

11: !(not)

12: &&(and)

13: ||(or)

```

18.1.2 表达式字符串的注意事项

- 字符串都以单引号或者双引号来引用;
- 当用单引号表示的字符串中需要出现单引号时, 需要用 “\” 进行转义, 如: length('\ hello \\') = 7;
- 当用双引号表示的字符串中需要出现双引号时, 需要用 “\” 进行转义, 如: length(\" hello \\\" \") = 7;
- 双引号出现在单引号表示的字符串中, 是不需要转义的, 如: length(" hello" ') = 7;
- 单引号出现在双引号表示的字符串中, 是不需要转义的, 如: length(" hello' ") = 7;
- 反斜杠自身也需要转义, 如: length("\\\\" hello\\\\" ") = 9。

18.1.3 表达式支持的字符串操作和函数

- 字符串比较’>’, ’>=’ , ’<’ , ’<=’ , ’=’ , ’==’ ;
- ’@’ (in) (支持中文)。

18.1.4 正则表达式和sql正则表达式区别

- xlib 的 rlike与odps 的sql中的rlike相同。

-
- search