分类与预测

杨惠

2017/10/18

Anaconda

- Anaconda是一个用于科学计算的Python发行版，支持 Linux, Mac, Windows系统，提供了包管理与环境管理的功能，可以很方便地解决多版本python并存、切换以及各种第三方包安装问题。
- Anaconda利用工具/命令conda来进行package和environment的管理，并且已经包含了Python和相关的配套工具。

Anaconda

conda 包管理系统

- 提供了和 pip 类似的管理功能并且有所增强（例如支持 env），并且，所有的包都是编译好的，不必像 pip 那样安装 numpy 和 scipy 时折腾许久。如果觉得默认包配置太累赘的话，完全可以下载 conda 最小包 miniconda 来进行安装。
- pip可以允许你在任何环境中安装python包，而conda允许你在conda环境中安装任何语言包（包括c语言或者python）。
- conda list　#查看已经安装的package
- conda update numpy　#更新包
- conda remove numpy　#卸载包

- 注：不建议交叉使用pip和conda，可能会导致库的安装路径不同，Python无法识别。

Anaconda

- 清华大学为 anaconda 提供了镜像服务，可以大大提高下载速度。
- 执行如下命令即可使用清华大学提供的anaconda镜像服务：
- conda config --add channels 'https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/'
- conda config --set show_channel_urls yes

scikit-learn

- scikit-learn（简称sklearn）是一个机器学习的Python模块，建立在SciPy基础之上。支持包括分类、回归、降维和聚类四大机器学习算法
- 主要特点：

  操作简单，高效的数据挖掘和数据分析

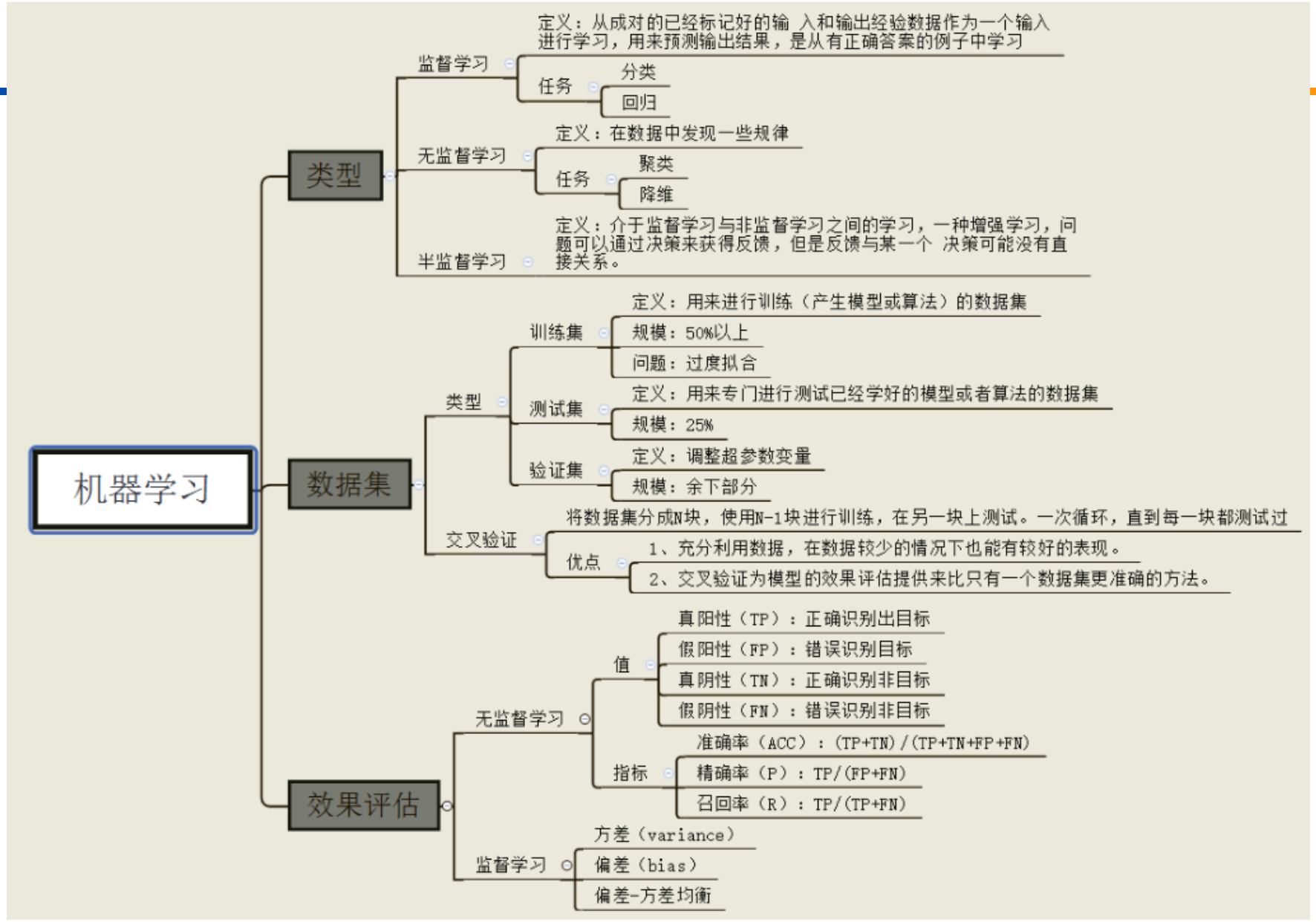
  无访问限制，在任何情况下都可以使用

  建立在numpy，SciPy，matplotlib基础上

  内置了大量数据集，节省了获取和整理数据集的时间
- 加载：import sklearn

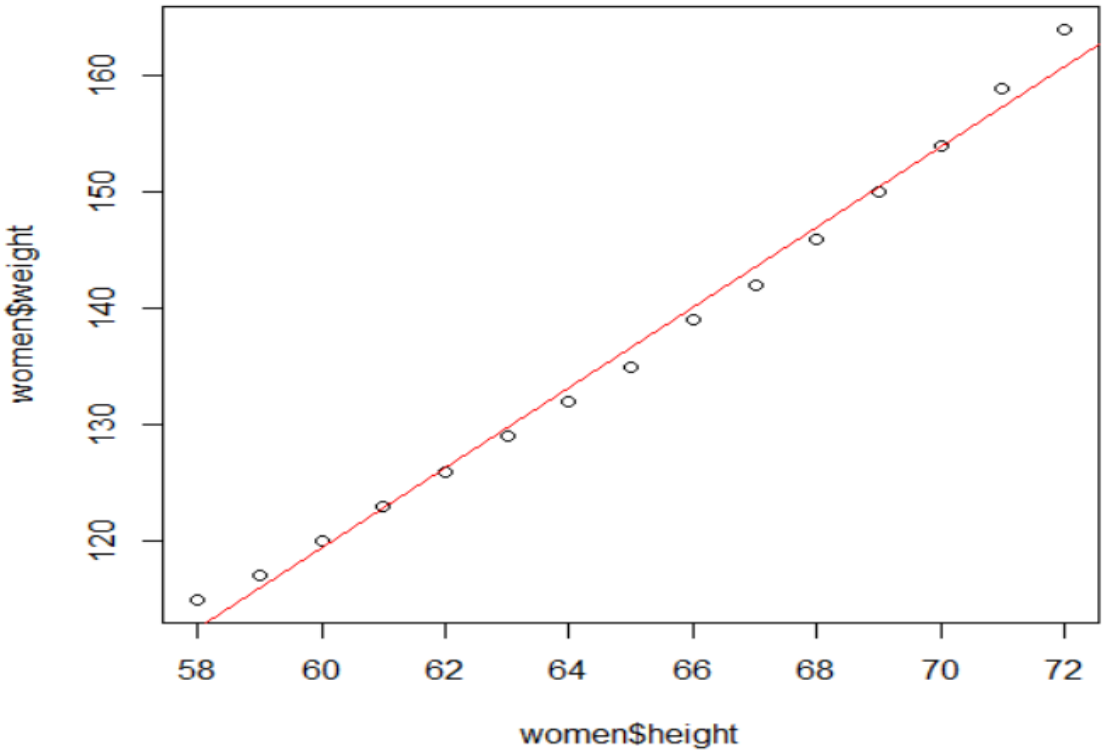
目录

机器学习


机器学习
类型
监督学习
定义：从成对的已经标记好的输 入和输出经验数据作为一个输入进行学习，用来预测输出结果，是从有正确答案的例子中学习
任务
分类
回归
无监督学习
定义：在数据中发现一些规律
任务
聚类
降维
半监督学习
定义：介于监督学习与非监督学习之间的学习，一种增强学习，问题可以通过决策来获得反馈，但是反馈与某一个 决策可能没有直接关系。
数据集
类型
训练集
定义：用来进行训练（产生模型或算法）的数据集
规模：50%以上
问题：过度拟合
测试集
定义：用来专门进行测试已经学好的模型或者算法的数据集
规模：25%
验证集
定义：调整超参数变量
规模：余下部分
交叉验证
将数据集分成N块，使用N-1块进行训练，在另一块上测试。一次循环，直到每一块都测试过
优点
1、充分利用数据，在数据较少的情况下也能有较好的表现。
2、交叉验证为模型的效果评估提供来比只有一个数据集更准确的方法。
效果评估
无监督学习
值
真阳性（TP）：正确识别出目标
假阳性（FP）：错误识别目标
真阴性（TN）：正确识别非目标
假阴性（FN）：错误识别非目标
指标
准确率（ACC）：(TP+TN)/(TP+TN+FP+FN)
精确率（P）：TP/(FP+FN)
召回率（R）：TP/(TP+FN)
监督学习
方差（variance）
偏差（bias）
偏差-方差均衡

回归分析

- 回归分析是一种预测性的建模技术，它研究的是因变量（目标）和自变量（预测器）之间的关系。这种技术通常用于预测分析以及发现变量之间的因果关系。例如，司机的鲁莽驾驶与道路交通事故数量之间的关系，最好的研究方法就是回归。
- 回归分析是建模和分析数据的重要工具。在这里，我们使用曲线/线来拟合这些数据点，在这种方式下，从曲线或线到数据点的距离差异最小。



为什么使用回归分析

如上所述，回归分析估计了两个或多个变量之间的关系。下面，让我们举一个简单的例子来理解它：

- 比如说，在当前的经济条件下，你要估计一家公司的销售额增长情况。现在，你有公司最新的数据，这些数据显示出销售额增长大约是经济增长的2.5倍。那么使用回归分析，我们就可以根据当前和过去的信息来预测未来公司的销售情况。

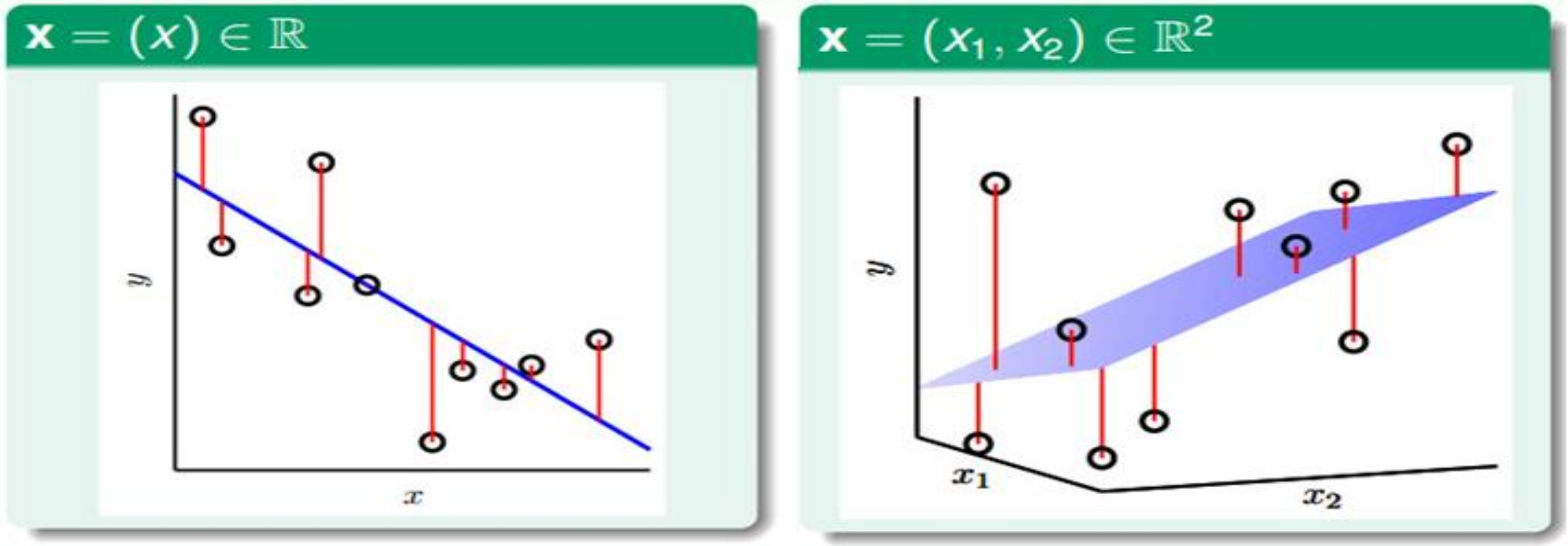
使用回归分析的好处良多。具体如下：

- 1. 它表明自变量和因变量之间的显著关系；
- 2. 它表明多个自变量对一个因变量的影响强度。

回归分析也允许我们去比较那些衡量不同尺度的变量之间的相互影响，如价格变动与促销活动数量之间联系。这些有利于帮助市场研究人员，数据分析人员以及数据科学家排除并估计出一组最佳的变量，用来构建预测模型。

线性回归

- 线性回归是最简单的回归模型。它的目的是：在自变量（输入数据）仅1维的情况下，找出一条最能够代表所有观测样本的直线（估计的回归方程）；在自变量（输入数据）高于1维的情况下，找到一个超平面使得数据点距离这个平面的误差（residuals）最小。而前者的情况在高中数学课本就已经学过，它的解法是普通最小二乘法（Ordinary Least Squares， OLS），线性回归示意图如下：



线性回归

- 线性回归模型如下式子所示：

$$f\left(x\right)=w_1x_1+w_2x_2+\cdots+w_nx_n+b\times1$$

- 其中权重ω_i和常数项b是待定的。这意味着将输入的自变量按一定的比例加权求和，得到预测值输出。
- 通过前面的学习我们知道，Python有强大的第三方扩展模块sklearn（读作scikit-learn），实现了绝大部分的数据挖掘基础算法，包括线性回归。
- 我们将通过举例说明，如何使用sklearn快速实现线性回归模型。这个例子是经典的波士顿房价预测问题
- 在sklearn的安装文件中，已包含此问题对应的数据集。

回归实现

线性回归

- sklearn.linear_model中的LinearRegression可实现线性回归（最小二乘法）
- LinearRegression 的构造方法：
- sklearn.linear_model.LinearRegression(

  fit_intercept=True #默认值为 True，表示 计算随机变量，False 表示不计算随机变量

  ，normalize=False #默认值为 False， 表示在回归前是否对回归因子 X 进行归一化

  ，True 表示是，copy_X=True)

回归实现

LinearRegression 的常用方法有：

- decision_function(X) #返回 X 的预测值 y
- fit(X,y[,n_jobs]) #拟合模型
- get_params([deep]) #获取 LinearRegression 构造方法的参数信息
- predict(X) #求预测值 #同 decision_function
- score(X,y[,sample_weight]) #确定性系数（ R^2 ），计算公式为： $1-\left(\frac{(y_{true}-y_{pre})^2}{(y_{true}-y_{truemean})^2}\right)$

  取值在[0，1]之间。越接近1，表明方程中的变量对y的解释能力越强。通常将R^2乘以100%表示回归方程解释y变化的百分比。
- set_params(**params) #设置 LinearRegression 构造方法的参数值

回归练习

使用Python实现下面输入与输出的线性回归

- 输入：[[0, 0], [1, 1], [2, 2]]——两个输入
- 输出：[0, 1, 2]
- 预测：[3, 3]

- 特别地，
- 输入：[1, 2, 3, 4] ——一个输入
- 输出：[1, 4, 9, 12]
- 预测：[[5]]

波士顿房价数据集（Boston House Price Dataset）

数据说明

- 波士顿房价数据集（Boston House Price Dataset）包含对房价的预测，以千美元计，给定的条件是房屋及其相邻房屋的详细信息。
- 该数据集是一个回归问题。每个类的观察值数量是均等的，共有 506 个观察，13 个输入变量和1个输出变量。
- sklearn库的datasets的load_boston包含该数据集

波士顿房价数据集（Boston House Price Dataset）

变量名说明：

- CRIM：城镇人均犯罪率。
- ZN：住宅用地超过 25000 sq.ft. 的比例。
- INDUS：城镇非零售商用土地的比例。
- CHAS：查理斯河空变量（如果边界是河流，则为1；否则为0）。
- NOX：一氧化氮浓度。
- RM：住宅平均房间数。
- AGE：1940 年之前建成的自用房屋比例。
- DIS：到波士顿五个中心区域的加权距离。
- RAD：辐射性公路的接近指数。
- TAX：每 10000 美元的全值财产税率。
- PTRATIO：城镇师生比例。
- B：1000（Bk-0.63）^ 2，其中 Bk 指代城镇中黑人的比例。
- LSTAT：人口中地位低下者的比例。
- MEDV：自住房的平均房价，以千美元计。

实现线性回归

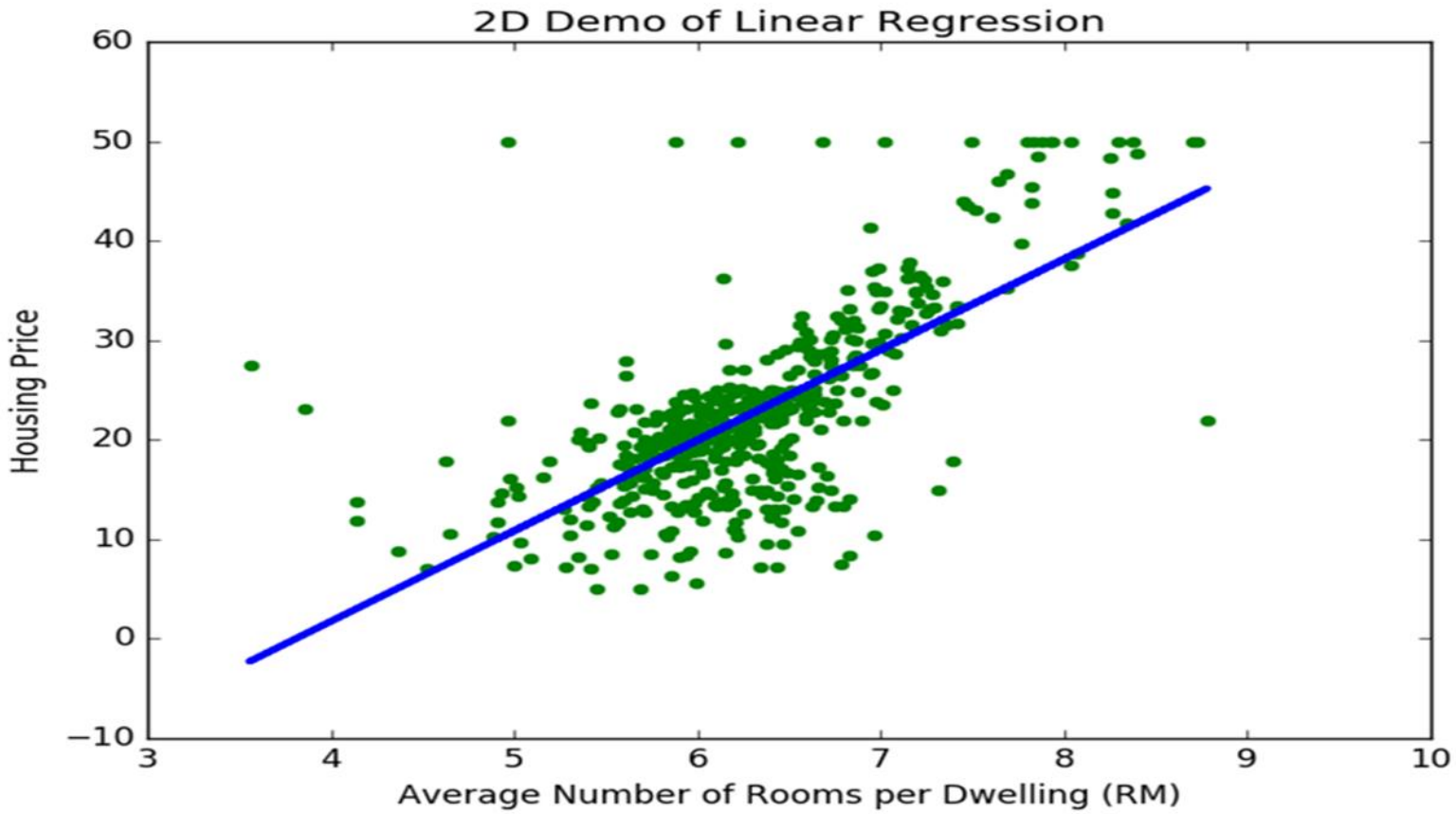
- 波士顿房价预测的部分数据

城镇人均犯罪率	……	住宅平均房间数	……	与五大就业中心的距离	……
0.00632	……	6.575	……	4.09	……
0.02731	……	6.421	……	4.9671	……
0.02729	……	7.185	……	4.9671	……
0.03237	……	6.998	……	6.0622	……
0.06905	……	7.147	……	6.0622	……

- 直觉告诉我们：上表的第6列（住宅平均房间数）与最终房价一般是成正比的，具有某种线性关系。我们利用线性回归来验证想法。
- 同时，作为一个二维的例子，可以在平面上绘出图形，进一步观察图形。

线性回归例子

我们利用线性回归来验证想法。同时，作为一个二维的例子，可以在平面上绘出图形，进一步观察图形，如下见图 ：



逻辑回归

- 分类和回归二者不存在不可逾越的鸿沟。就波士顿房价预测作为例子：如果将房价按高低分为“高级”、“中级”和“普通”三个档次，那么这个预测问题也属于分类问题。
- 准确地说，逻辑回归（Logistic Regression）是对数几率回归，属于广义线性模型（GLM），它的因变量一般只有0或1.
- 需要明确一件事情：线性回归并没有对数据的分布进行任何假设，而逻辑回归隐含了一个基本假设 ：每个样本均独立服从于伯努利分布（0-1分布）。
- 伯努利分布属于指数分布族，这个大家庭还包括：高斯（正态）分布、多项式分布、泊松分布、伽马分布、Dirichlet分布等。

逻辑回归

- 事实上，假设数据服从某类指数分布，我们可以有线性模型拓展出一类广义模型，即通过非线性的的**关联函数**（Link Function）将线性函数映射到其他空间上，从而大大扩大了线性模型本身可解决的问题范围。根据数理统计的基础知识，指数分布的概率密度函数可抽象的表示为：

$$p(y;\eta)=b(y)\cdot e^{\eta^T T(y)-\alpha(\eta)}$$

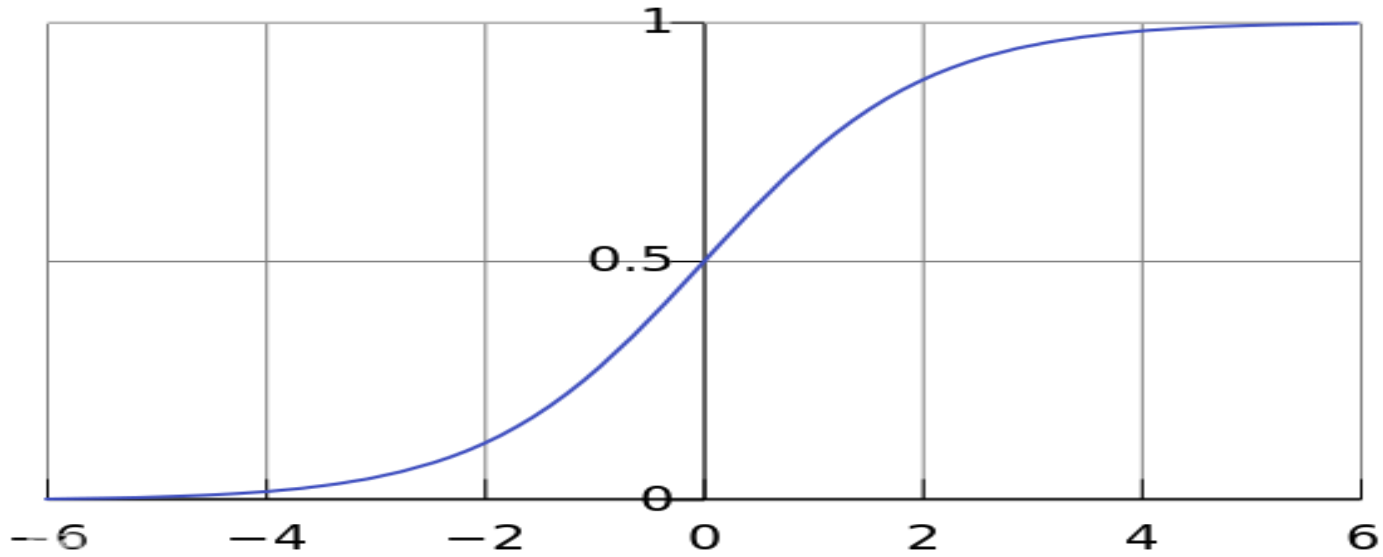
- 其中，η是待定的参数，$T(y)$是充分统计量，通常 $T(y)=y$
- 而伯努利概率分布的密度函数为：

$$\begin{aligned} p(y;h) &= h^y(1-h)^{1-y} \\ &= e^{y\ln h+(1-y)\ln(1-h)} \\ &= e^{\ln\left(\frac{h}{1-h}\right)y+\ln(1-h)} \end{aligned}$$

- 其中，h是有假设衍生的关联函数，y可能的取值为0或1，值得注意的是，当y=1时，p（y，h）=h。也就是说h表征样本属于正类（类别 ‘1’ ）的概率。

逻辑回归

- 对照上述式子，令 $\eta=\ln\left(\frac{h}{1-h}\right)$，可得：$h=\frac{1}{1+e^{-\eta}}$
- 这便是大名鼎鼎的Logistic函数，亦称Sigmoid函数。因为它的函数形如字母“S”，如下图：



- 观察图像可知，当指数分布的自然参数η在变化时，h的取值范围恰好为（0,1）。由于h的表征样本属于正类（类别‘1’）的概率，通常将h大于某个阈值（如0.5）的样本预测为‘属于正类（1）’，否则预测结果为‘属于负类（0）’.

逻辑回归

- 由基本假设$\eta = w^T x$。给定x，目标函数为：

$$h_w(x) = E[T(y) \mid x] = E[y \mid x] = p(y = 1 \mid x; w) = h = \frac{1}{1 + e^{-w^T x}}$$

- 上式表明，我们的最终目标是根据数据，确定合适的一组权重ω。在对原始输入进行加权组合之后，通过关联函数作非线性变换，得到的结果表示样本x属于正类的概率。因此，关联函数亦被称为 ‘激活函数’，如同神经元接受到足够的刺激，才会变得兴奋。
- 通常，确定权重ω采用的方法是：最大似然估计（ Maximum Likelihood Estimation ）

数据说明

通过分析不同的因素对研究生录取的影响来预测一个人是否会被录取。

- 数据的格式如下：

	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3

- admit ：表示是否被录取(目标变量)
- gre：标准入学考试成绩，预测变量
- gpa：学业平均绩点，预测变量
- rank：母校排名(预测变量)

——LogisticRegression.csv文件

Logistics Regression 做分类问题

数据预处理——独热编码（one-hot encoding）

- 离散特征的编码分为两种情况：
  1、离散特征的取值之间没有大小的意义，比如color：[red,blue],那么就使用one-hot编码
  2、离散特征的取值有大小的意义，比如size:[X,XL,XXL],那么就使用数值的映射{X:1,XL:2,XXL:3}

- 独热编码即 One-Hot 编码，又称一位有效编码，其方法是使用N位状态寄存器来对N个状态进行编码，每个状态都有它独立的寄存器位，并且在任意时候，其中只有一位有效。

Logistics Regression 做分类问题

离散数据编码

M：1
L：2
XL：3

	color	size	prize	class label
0	green	M	10.1	0
1	red	L	13.5	1
2	blue	XL	15.3	0
3	red	M	10.1	0

	color	size	prize	class label
0	green	M	10.1	0
1	red	L	13.5	1
2	blue	XL	15.3	0
3	red	M	10.1	0

	size	prize	class label	color_red	color_green	color_blue
0	M	10.1	0	0	1	0
1	L	13.5	1	1	0	0
2	XL	15.3	0	0	0	1
3	M	10.1	0	1	0	0

Logistics Regression 做分类问题

数据预处理——独热编码（one-hot encoding）

- 使用pandas的get_dummies函数进行one-hot编码
- get_dummies(data, prefix=None, prefix_sep='_', dummy_na=False, columns=None, sparse=False, drop_first=False)
- data：数据（数组或者数据框）
- prefix：默认为None，如果是列名，则表示字符串传递长度等于在DataFrame上调用get_dummies时的列数的列表
- columns：要编码的DataFrame中的列名。如果*列*是None，那么所有与列 *对象*或*类别* D型细胞将被转换。

数据集划分

sklearn.model_selection.train_test_split随机划分训练集和测试集

- train_test_split是交叉验证中常用的函数，功能是从样本中随机的按比例选取train data和testdata，形式为：
- X_train,X_test, y_train, y_test =cross_validation.train_test_split(train_data,train_target,test_size=0.4, random_state=0)

数据集划分

train_test_split参数解释：

- train_data：所要划分的样本特征集
- train_target：所要划分的样本结果
- test_size：样本占比，如果是整数的话就是样本的数量
- random_state：是随机数的种子。
- 随机数种子：其实就是该组随机数的编号，在需要重复试验的时候，保证得到一组一样的随机数。比如你每次都填1，其他参数一样的情况下你得到的随机数组是一样的。但填0或不填，每次都会不一样。
- 随机数的产生取决于种子，随机数和种子之间的关系遵从以下两个规则：
- 种子不同，产生不同的随机数；种子相同，即使实例不同也产生相同的随机数。

算法实现

工程中求解逻辑回归更倾向于选择一些迭代改进的算法，它们会直接对解空间进行部分搜索，找到合适的结果便停止寻优。在入门时建议首先掌握scikit-learn中的逻辑回归实现算法。

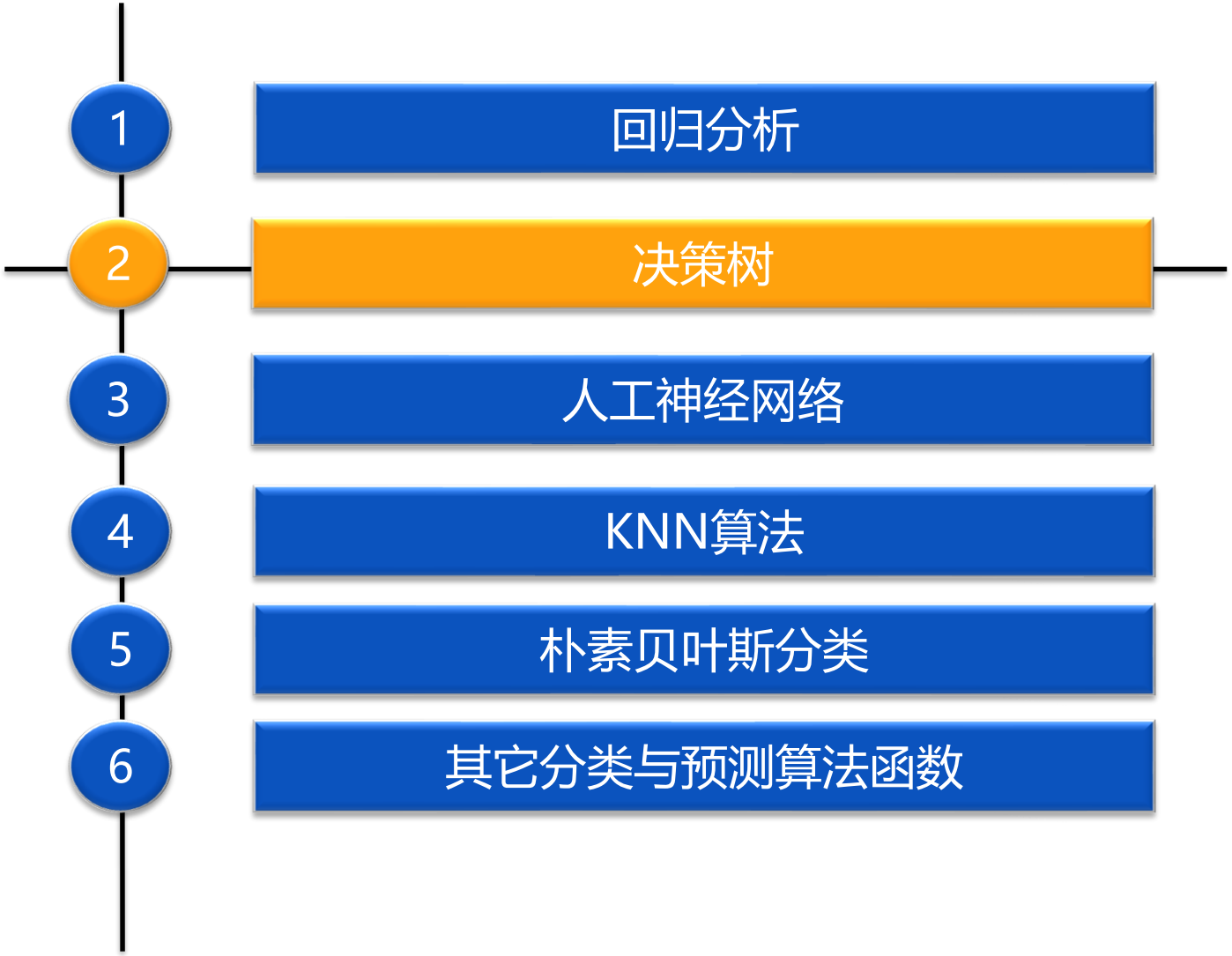
算法实现代码如下：

- import pandas as pd
- from sklearn.linear_model import LogisticRegression, RandomizedLogisticRegression
- from sklearn.cross_validation import train_test_split
- # 导入数据并观察
- data = pd.read_csv('../data/LogisticRegression.csv', encoding='utf-8')
- #将类别型变量进行独热编码one-hot encoding
- data_dum = pd.get_dummies(data, columns=['rank'])
- print data_dum.tail(5) # 查看数据框的最后五行

算法实现

- # 切分训练集和测试集
- X_train, X_test, y_train, y_test = train_test_split(data_dum.ix[:, 1:], data_dum.ix[:, 0], test_size=.1, random_state=520)
- lr = LogisticRegression() # 建立LR模型
- lr.fit(X_train, y_train) # 用处理好的数据训练模型
- print ('逻辑回归的准确率为：{0:.2f}%'.format(lr.score(X_test, y_test) *100))

目录



决策树概述

- 决策树方法在分类、预测、规则提取等领域有着广泛应用。
- 决策树是一树状结构，它的每一个叶节点对应着一个分类，非叶节点对应着在某个属性上的划分，根据样本在该属性上的不同取值将其划分成若干个子集。
- 对于非纯的叶节点，多数类的标号给出到达这个节点的样本所属的类。构造决策树的核心问题是在每一步如何选择适当的属性对样本做拆分。
- 对一个分类问题，从已知类标记的训练样本中学习并构造出决策树是一个自上而下，分而治之的过程。

决策树概述

优点：

- 易于理解
- 只需要很少的准备数据
- 复杂度是数据点数的对数
- 能够同时处理数值和分类数据
- 能够处理多输出问题
- 采用白盒模型
- 使用统计测试可以验证模型
- 即使假设有点错误也可以表现很好

缺点：

- 可以创建复杂树但不能很好的推广
- 不稳定
- 是NP 问题
- 有很难学习的概念
- 如果一些类占主导地位创建的树就有偏差

决策树算法分类

常用的决策树算法见下表：

决策树算法	算法描述
ID3算法	其核心是在决策树的各级节点上，使用信息增益作为属性的选择标准，来帮助确定每个节点所应采用的合适属性。
C4.5算法	C4.5决策树生成算法相对于ID3算法的重要改进是使用信息增益率来选择节点属性。C4.5算法既能够处理离散的描述属性，也可以处理连续的描述属性。
C5.0算法	C5.0是C4.5算法的修订版，适用于处理大数据集，采用Boosting方式提高模型准确率，根据能够带来的最大信息增益的字段拆分样本。
CART算法	CART决策树是一种十分有效的非参数分类和回归方法，通过构建树、修剪树、评估树来构建一个二叉树。当终结点是连续变量时，该树为回归树；当终结点是分类变量，该树为分类树。

ID3算法

- ID3算法基于信息熵来选择最佳测试属性。
- 它选择当前样本集中具有最大信息增益值的属性作为测试属性；样本集的划分则依据测试属性的取值进行，测试属性有多少不同取值就将样本集划分为多少子样本集，同时决策树上相当于该样本集的节点长出新的叶子节点。
- ID3算法根据信息论理论，采用划分后样本集的不确定性作为衡量划分好坏的标准，用信息增益值度量不确定性：信息增益值越大，不确定性越小。
- 因此，ID3算法在每个非叶子节点选择信息增益最大的属性作为测试属性，这样可以得到当前情况下最纯的拆分，从而得到较小的决策树。

ID3基本原理

- 设S是s个数据样本的集合。假定类别属性具有m个不同的值 $C_i(i=1,2,\cdots,m)$ 设s_i是C_i中的样本数。对给定一个样本，总信息熵为：$I(s_1,s_2,\cdots,s_m)=-\sum_{i=1}^{m}P_i\log_2(P_i)$
- 其中，p_i是任意样本属于C_i的概率，一般可以用 $\frac{s_i}{s}$ 估计。
- 若一个属性A具有k个不同的值 $\{a_1,a_2,\cdots,a_k\}$，利用属性A将集合S划分为j个子集 $\{S_1,S_2,\cdots,S_j\}$，其中s_i包含了集合S中属性A取值为a_j的样本。若选择属性A为测试属性，则这些子集就是从集合S的节点生长出来的新的叶节点。设s_{ij}是子集S_j中类别为C_i的样本数，则根据属性A划分样本的信息熵为：

$$E(A)=\sum_{j=1}^{k}\frac{s_{1j}+s_{2j}+\cdots+s_{mj}}{s}I(s_{1j},s_{2j},\cdots,s_{mj})$$

- 其中，$I(s_{1j},s_{2j},\cdots,s_{mj})=-\sum_{i=1}^{m}P_i\log_2(P_i)$ $P_{ij}=\frac{s_{ij}}{s_{1j}+s_{2j}+\cdots+s_{mj}}$ 是子集S_j中类别为C_i的样本的概率。

ID3基本原理

- 最后，用属性 A划分样本集 S后所得的信息增益（Gain）为：

$$Gain(A) = I(s_1, s_2, \cdots, s_m) - E(A)$$

- 显然 E(A)越小，Gain(A) 的值越大，说明选择测试属性 A对于分类提供的信息越大，选择 A之后分类的不确定程度的越小。
- 属性A 的 k个不同的值对应样本集S 的k 个子集或分支，通过递归调用上述过程（不包括已选择的属性），生成其他属性作为节点的子节点和分支来生成整棵决策树。
- ID3决策树算法作为一个典型的决策树学习算法，其核心是在决策树的各级节点上都用信息增益作为判断标准进行属性的选择，使得在每个非叶子节点上进行测试时，都能获得最大的类别分类增益，使分类后数据集的熵最小。这样的处理方法使得树的平均深度最小，从而有效地提高分类效率。

ID3算法实现

ID3算法的详细实现步骤如下：

a
- 对当前样本集合，计算所有属性的信息增益；

b
- 选择信息增益最大的属性作为测试属性，把测试属性取值相同的样本划为同一个子样本集；

c
- 若子样本集的类别属性只含有单个属性，则分支为叶子节点，判断其属性值并标上相应的符号之后返回调用处；否则对子样本集递归调用本算法。

ID3算法实现

我们通过举例说明：使用scikit-learn建立基于信息熵的决策树模型。

- 这个例子是经典的Kaggle101问题——泰坦尼克生还预测，部分数据如下：

Survived	PassengerId	Pclass	Sex	Age
0	1	3	male	22
1	2	1	female	38
1	3	3	female	26
1	4	1	female	35
0	5	3	male	35

- 为了说明的方便，数据集有许多属性被删除了。通过观察可知：列Survived是指是否存活，是类别标签，属于预测目标；列Sex的取值是非数值型的。我们在进行数据预处理时应该合理应用Pandas的功能，让数据能够被模型接受。

ID3算法实现

具体实现代码如下：

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier as DTC, export_graphviz
data = pd.read_csv('../data/titanic_data.csv', encoding='utf-8')
data.drop(['PassengerId'], axis=1, inplace=True)  # 舍弃ID列，不适合作为特征
# 数据是类别标签，将其转换为数，用1表示男，0表示女。
data.loc[data['Sex'] == 'male', 'Sex'] = 1
data.loc[data['Sex'] == 'female', 'Sex'] = 0
data.fillna(int(data.Age.mean()), inplace=True)
print(data.head(5))   # 查看数据
```

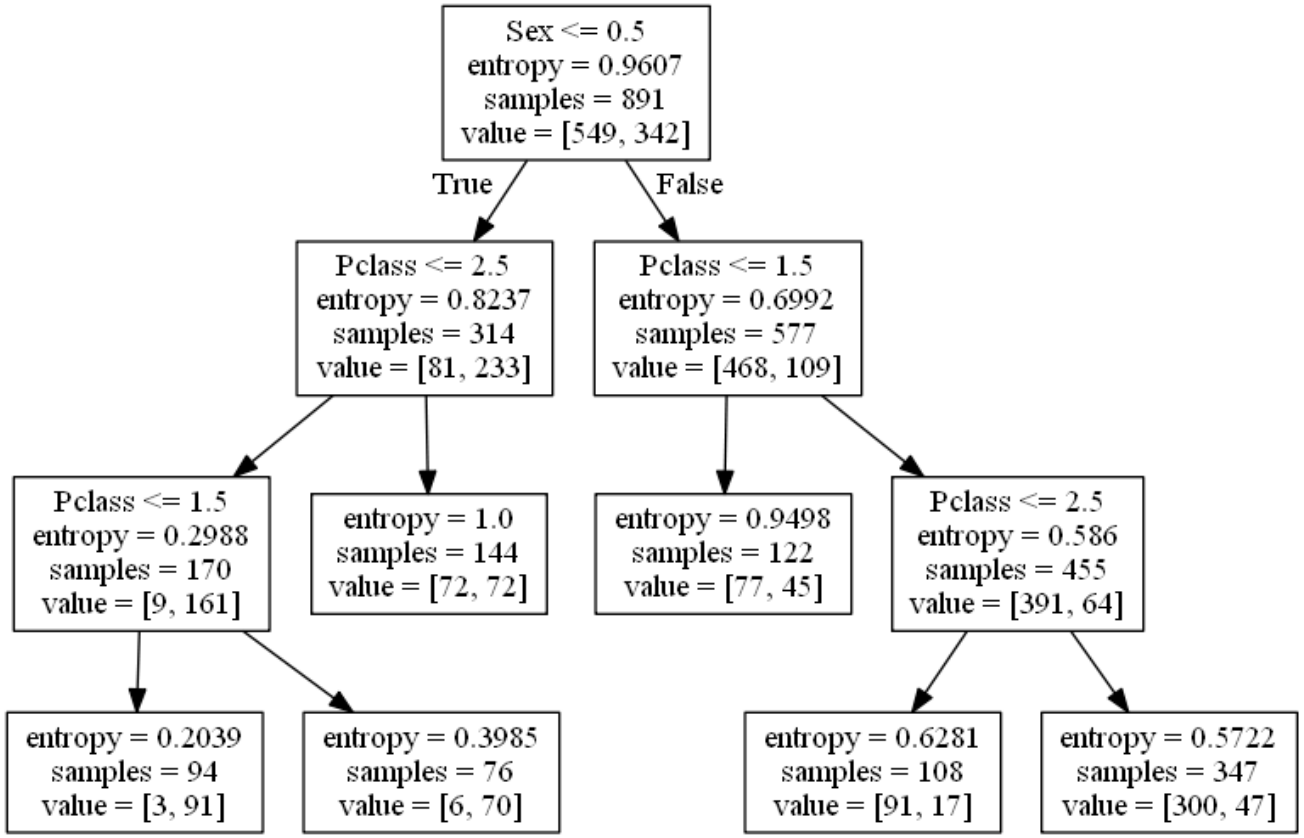
ID3算法实现

- ```
  X = data.iloc[:, 1:3]    # 为便于展示，未考虑年龄（最后一列）
  y = data.iloc[:, 0]
  dtc = DTC(criterion='entropy')    # 初始化决策树对象，基于信息熵
  dtc.fit(X, y)    # 训练模型
  print '输出准确率：', dtc.score(X,y)
  # 可视化决策树，导出结果是一个dot文件，需要安装Graphviz才能转换为.pdf或.png格式
  with open('../tmp/tree.dot', 'w') as f:
      f = export_graphviz(dtc, feature_names=X.columns, out_file=f)
  ```
  ```

ID3算法实现

运行代码后，将会输出一个tree.dot的文本文件。为了进一步将它转换为可视化格式，需要安装Graphviz（跨平台的、基于命令行的绘图工具），再在命令行中以如下方式编译。

生成的效果图如下：



其他树算法

- ID3算法是决策树系列中的经典算法之一，它包含了决策树作为机器学习算法的主要思想。但ID3算法在实际应用中有许多不足，所以在此之后提出了大量的改进策略，如C4.5算法，C5.0算法和CART算法。
- 由于ID3决策树算法采用信息增益作为选择测试属性的标准，会偏向于选择取值较多的，即所谓高度分支属性，而这类属性并不一定是最优的属性。
- 同时，ID3算法只能处理离散属性，对于连续型的属性，在分类前需要对其进行离散化。为了解决倾向于选择高度分支属性的问题，人们采用信息增益率作为选择测试属性的标准，这样便得到C4.5决策树算法。

C4.5算法

C4.5是基于ID3算法进行改进后的一种重要算法，它是一种监督学习算法，其目标是通过学习，找到一个从属性值到类别的映射关系，并且这个映射能用于对新的类别未知的实体进行分类。

C4.5算法的优点

- 产生的分类规则易于理解，准确率较高；相比于ID3算法，能处理非离散数据或不完整数据
- 改进了ID3算法的缺点：使用信息增益选择属性时偏向于选择高度分支属性

C4.5算法的缺点

- 在构造树的过程中，需要对数据集进行多次的顺序扫描和排序，因而导致算法的低效
- 当训练集大小超过内存上限时程序无法运行，故适合能够驻留于内存的数据集

C5.0算法

- C5.0算法是C4.5算法的修订版，适用于处理大数据集，采用Boosting方式提高模型准确率，又称为Boosting Trees，在软件上计算速度比较快，占用的内存资源较少。
- C5.0作为经典的决策树模型算法之一，可生成多分支的决策树，C5.0算法根据能够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，这一过程重复进行直到样本子集不能再被拆分为止。最后，重新检查最低层次的拆分节点，那些对模型值没有显著贡献的样本子集被剔除或者修剪。

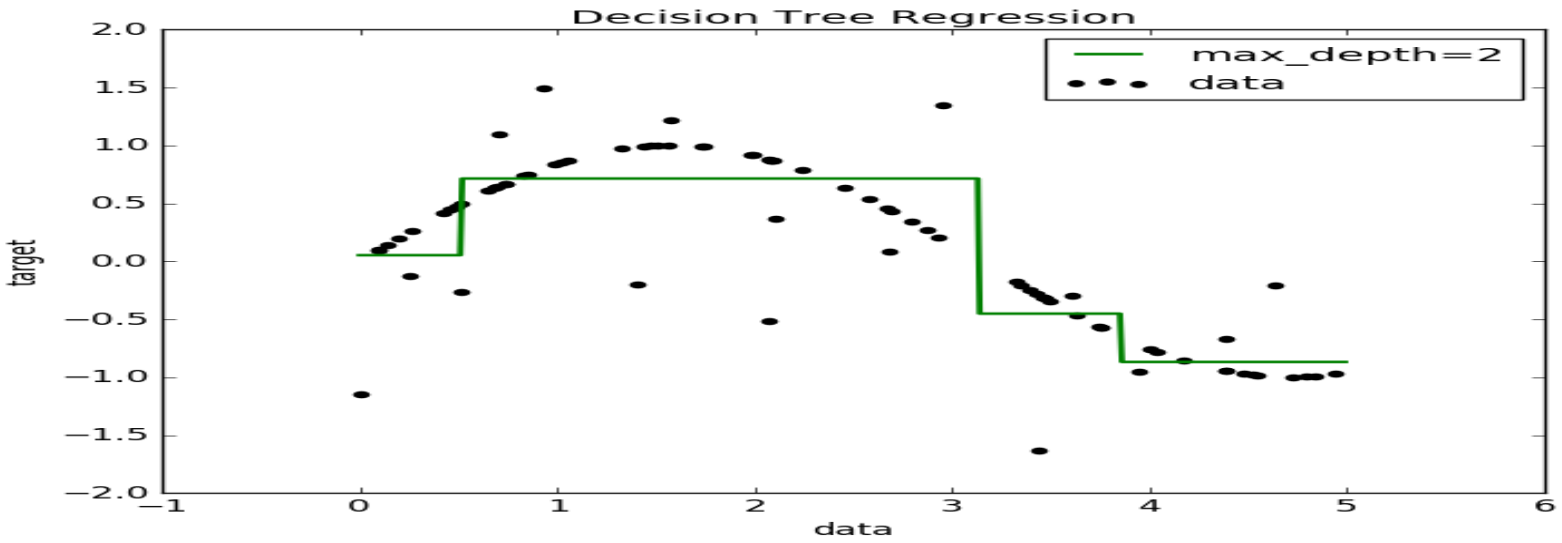
C5.0算法

C5.0较其他决策树算法的优势

C5.0模型在面对数据遗漏和输入字段很多的问题时非常稳健；	C5.0模型易于理解，模型输出的规则有非常直观的解释；	C5.0模型也提供了强大技术支持以提高分类的精度。

CART算法

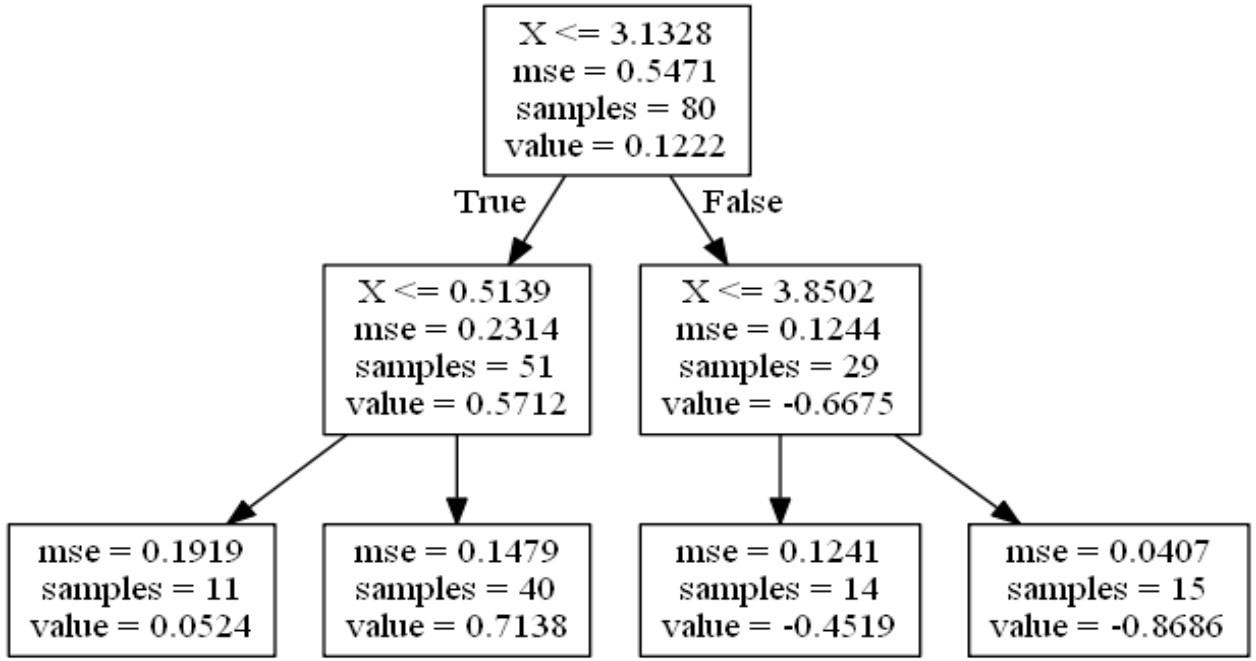
- 分类回归树（Classification And Regression Tree，CART）算法最早由Breiman等人提出，现已在统计领域和数据挖掘技术中普遍使用，Python中的scikit-learn模块的Tree子模块主要使用CART算法来实现决策树。



- 上图 中的数据由正弦函数 随机生成。可以明显观察到：由CART算法生成的回归线对应一个阶梯函数。

CART算法

我们将CART模型内部的分类规则可视化，如下图所示：



- 由于决策树的特性，1维自变量仅能体现为阶梯函数（不可能出现斜线或曲线）。但考虑极限情况，阶梯函数可以逼近一条曲线。这里可以认为：在仅允许用阶梯函数作回归的条件下，算法达到了均方误差最小的要求。

决策树

分类——实现类是DecisionTreeClassifier，能够执行数据集的多类分类

- 输入参数为两个数组X[n_samples,n_features]和 y[n_samples],X 为训练数据，y 为训练数据的标记数据
- DecisionTreeClassifier 构造方法为：
- sklearn.tree.DecisionTreeClassifier(criterion='gini'， splitter='best' ，max_depth=None
  , min_samples_split=2
  , min_samples_leaf=1
  , max_features=None
  , random_state=None
  , min_density=None
  , compute_importances=None
  , max_leaf_nodes=None)

决策树

回归——实现类是DecisionTreeRegressor，输入为X，y 同上，y 为浮点数

- DecisionTreeRegressor 构造方法为：
- sklearn.tree.DecisionTreeRegressor(criterion='mse'，splitter='best'，max_depth=None
  , min_samples_split=2
  , min_samples_leaf=1
  , max_features=None
  , random_state=None
  , min_density=None
  , compute_importances=None
  , max_leaf_nodes=None)

决策树

多输出问题——实现类有：DecisionTreeClassifier 和DecisionTreeRegressor

- 构造方法同上。
- DecisionTreeClassifier 示例：

```
from sklearn.datasets import load_iris
from sklearn.cross_validation import cross_val_score
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0)
iris = load_iris()
cross_val_score(clf, iris.data, iris.target, cv=10)
```

决策树

多输出问题——实现类有：DecisionTreeClassifier 和DecisionTreeRegressor

- 构造方法同上。
- DecisionTreeRegressor 示例：

```
from sklearn.datasets import load_boston
from sklearn.cross_validation import cross_val_score
from sklearn.tree import DecisionTreeRegressor
boston = load_boston()
regressor = DecisionTreeRegressor(random_state=0)
cross_val_score(regressor, boston.data, boston.target, cv=10)
```

Thank you!

泰迪科技：www.tipdm.com
热线电话：40068-40020