分类与预测

杨惠

2017/10/23

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
- conda install numpy #安装包
- 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/'

（想要删除清华源把add改成remove就行。）

- conda config --set show_channel_urls yes #显示URL

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

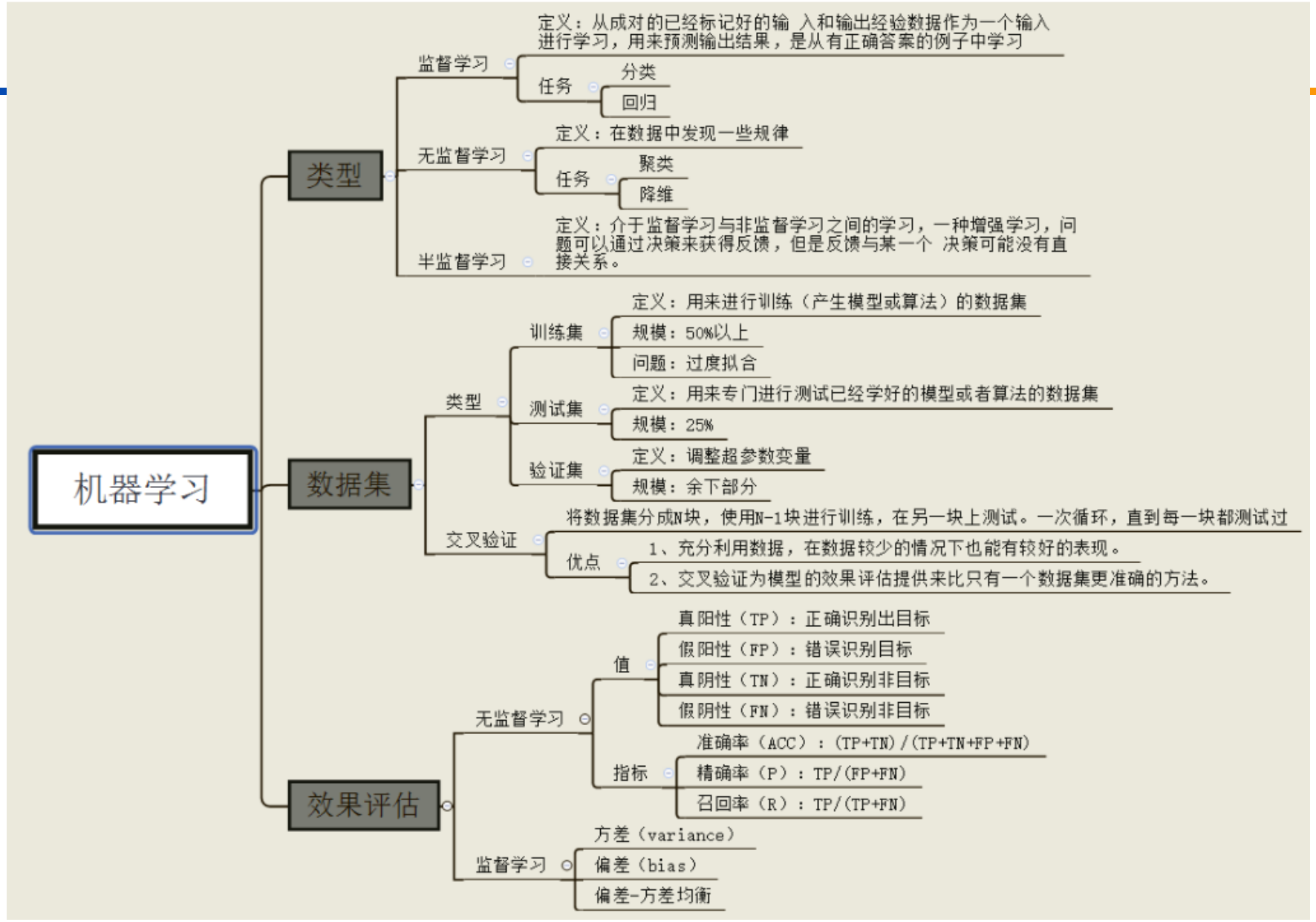
  无访问限制，在任何情况下都可以使用

  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

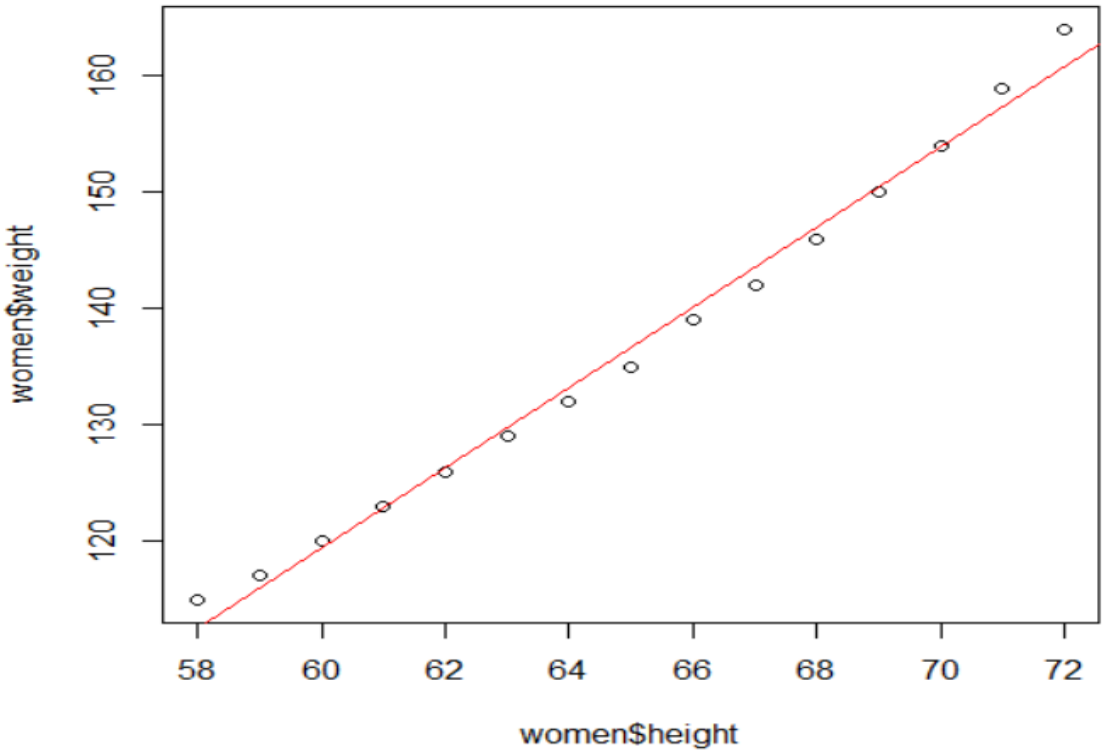
目录

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

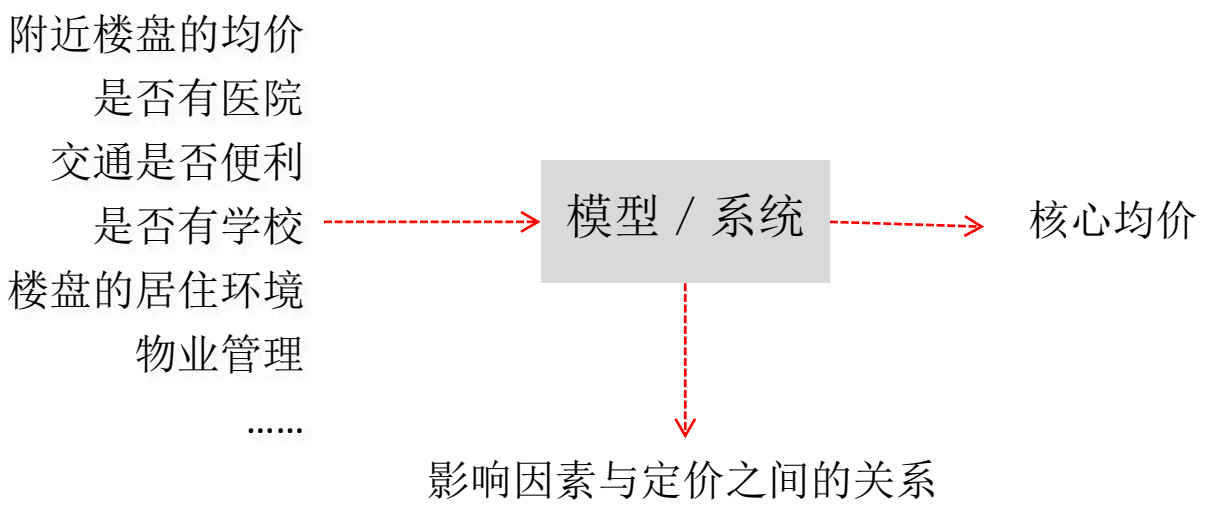
回归分析

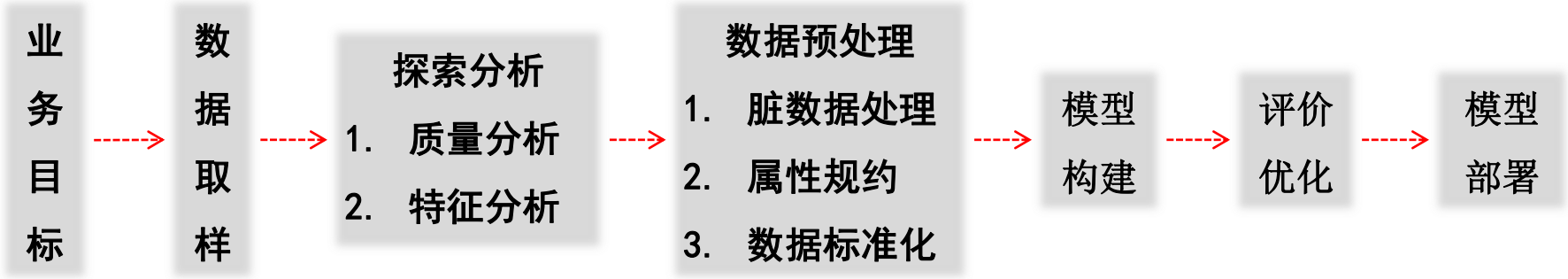
- 回归分析是一种预测性的建模技术，它研究的是因变量（目标）和自变量（预测器）之间的关系。这种技术通常用于预测分析以及发现变量之间的因果关系。例如，司机的鲁莽驾驶与道路交通事故数量之间的关系，最好的研究方法就是回归。
- 回归分析是建模和分析数据的重要工具。在这里，我们使用曲线/线来拟合这些数据点，在这种方式下，从曲线或线到数据点的距离差异最小。



回归问题

目标：确定新楼盘的核心均价





为什么使用回归分析

回归分析估计了两个或多个变量之间的关系。

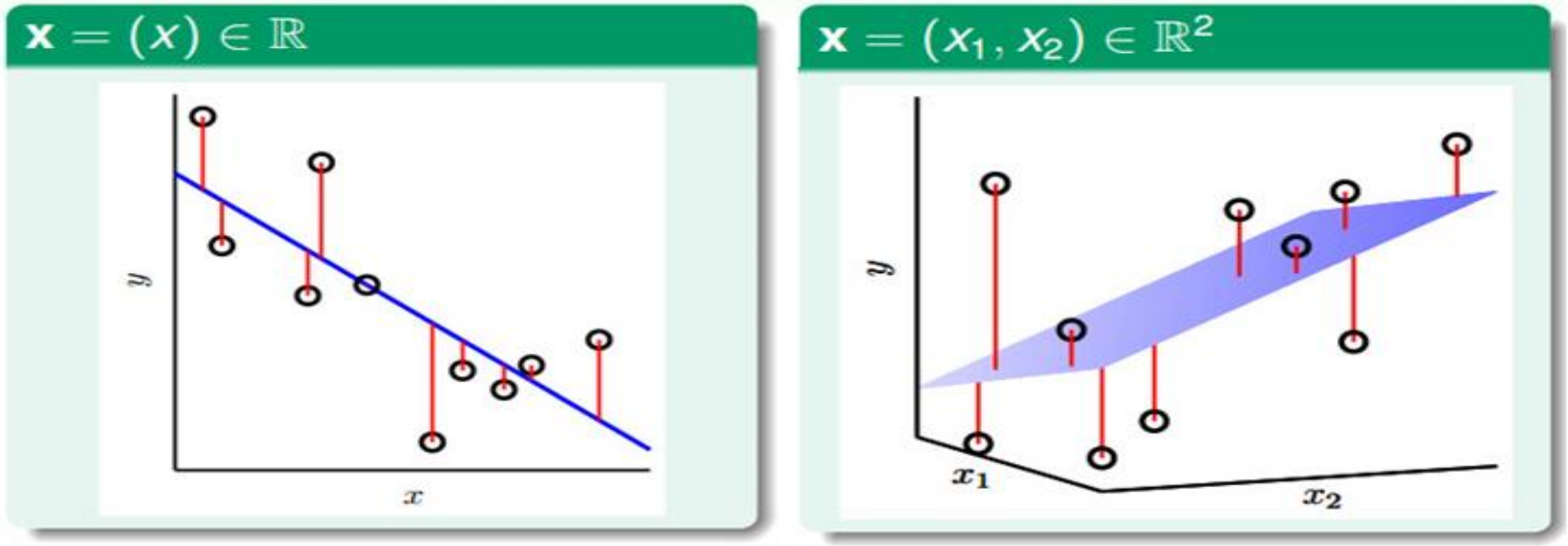
使用回归分析的好处良多。具体如下：

- 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，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
- 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包含该数据集（ load_boston ）

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

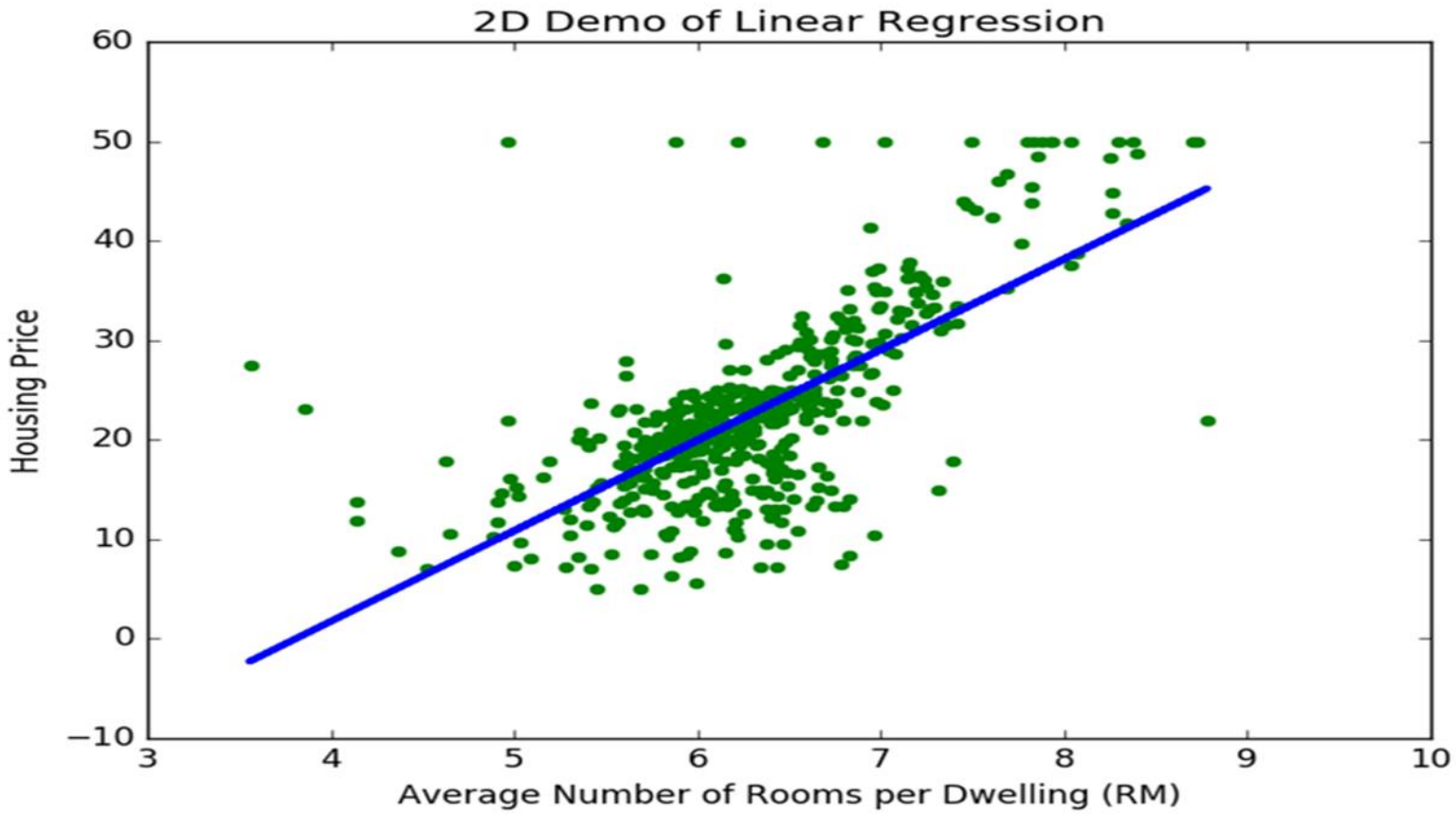
- 波士顿房价预测的部分数据

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：

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

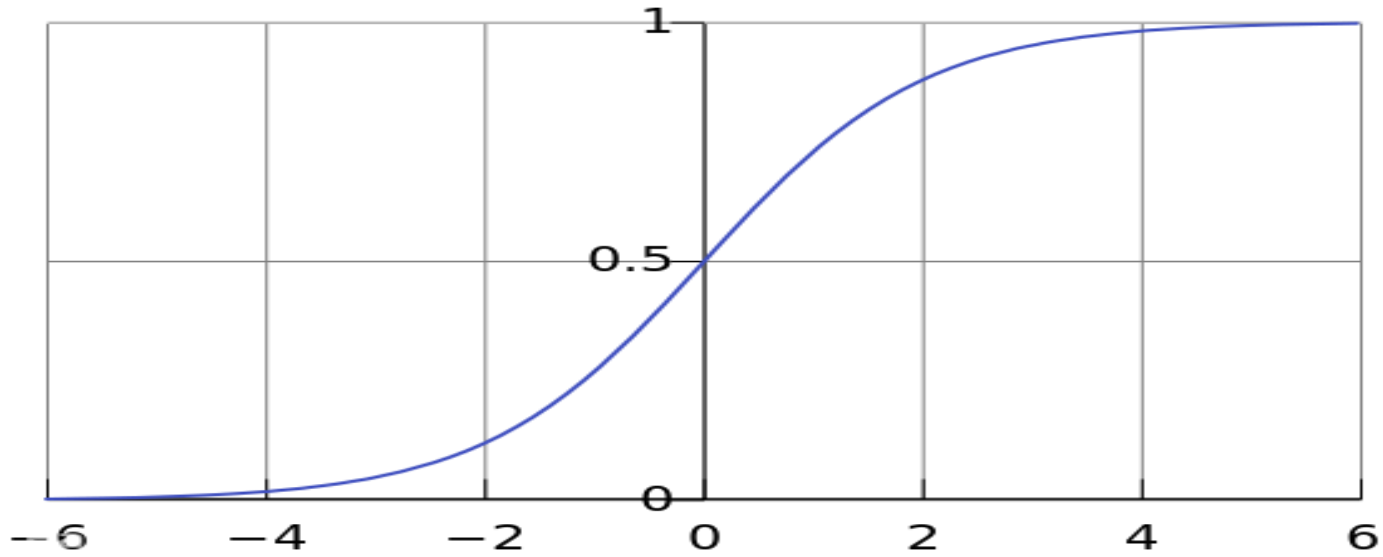
- 其中，η是待定的参数，$T(y)$是充分统计量，通常 $T(y)=y$
- 而伯努利概率分布的密度函数为：

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

- 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——离散数据编码

- 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. cross_validation.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

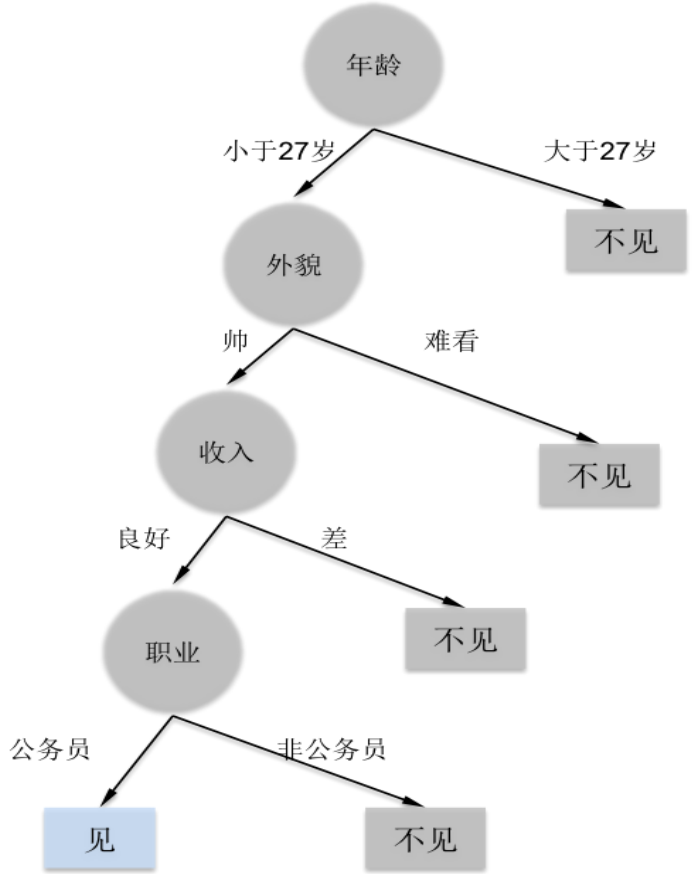
缺点：

- 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

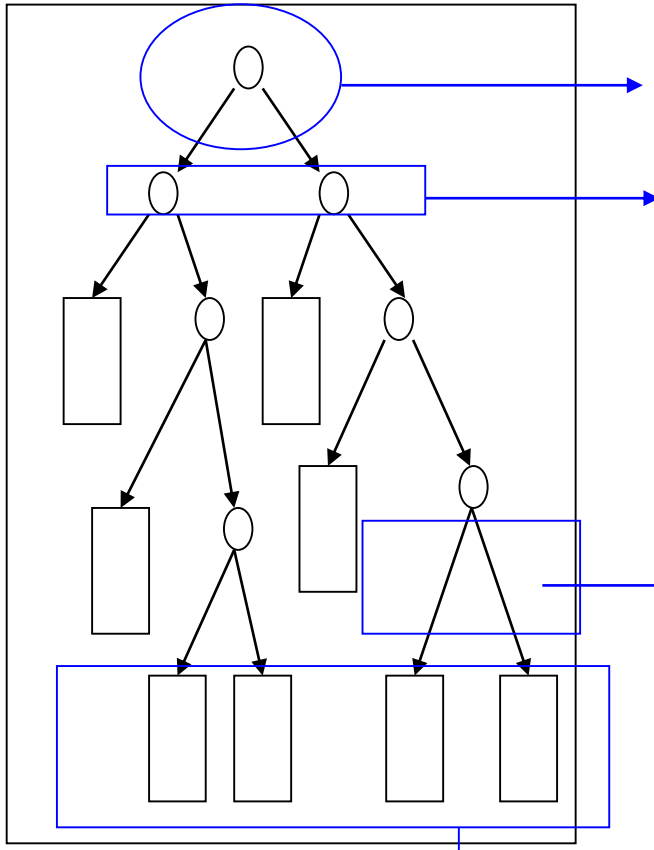
套用俗语，决策树分类的思想类似于找对象。现想象一个女孩的母亲要给这个女孩介绍男朋友，于是有了下面的对话：

- 右图完整表达了这个女孩决定是否见一个约会对象的策略。
- 圆圈节点：决策依据；
- 方框节点：决策结果；
- 箭头：决策路径。



决策树

结构



根部节点(root node)

中间节点(non-leaf node)(代表测试的条件)

分支(branches)(代表测试的结果)

叶节点(leaf node)(代表分类后所获得的分类标记)

决策树

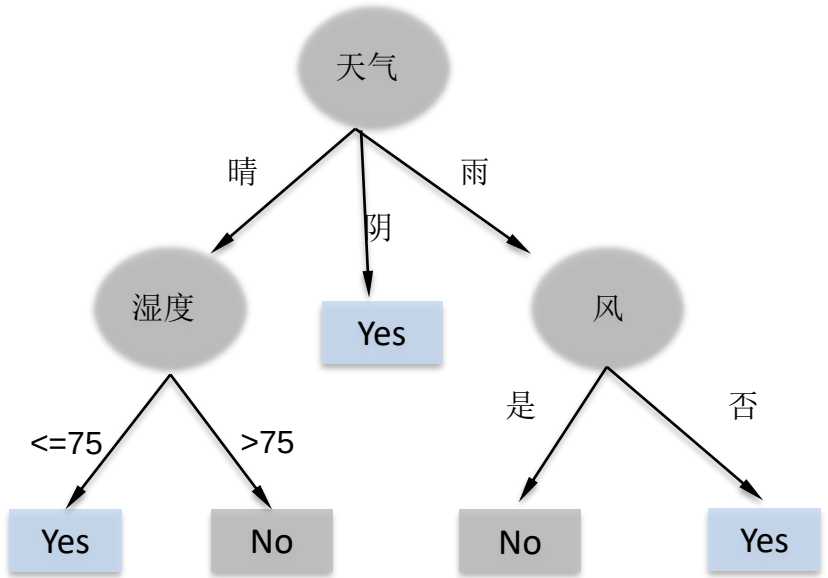
天气情况对是否打高尔夫球的影响

日期	天气	温度(华氏度)	湿度	起风	打球?
1	晴	85	85	F	No
2	晴	80	90	T	No
3	阴	83	78	F	Yes
4	雨	70	96	F	Yes
5	雨	68	80	F	Yes
6	雨	65	70	T	No
7	阴	64	65	T	Yes
8	晴	72	95	F	No
9	晴	69	70	F	Yes
10	雨	75	80	F	Yes
11	晴	75	70	T	Yes
12	阴	72	90	T	Yes
13	阴	81	75	F	Yes
14	雨	71	80	T	No
15	阴	85	90	F	?
16	雨	80	79	F	?
17	晴	78	70	T	?

决策树

天气情况对是否打高尔夫球的影响

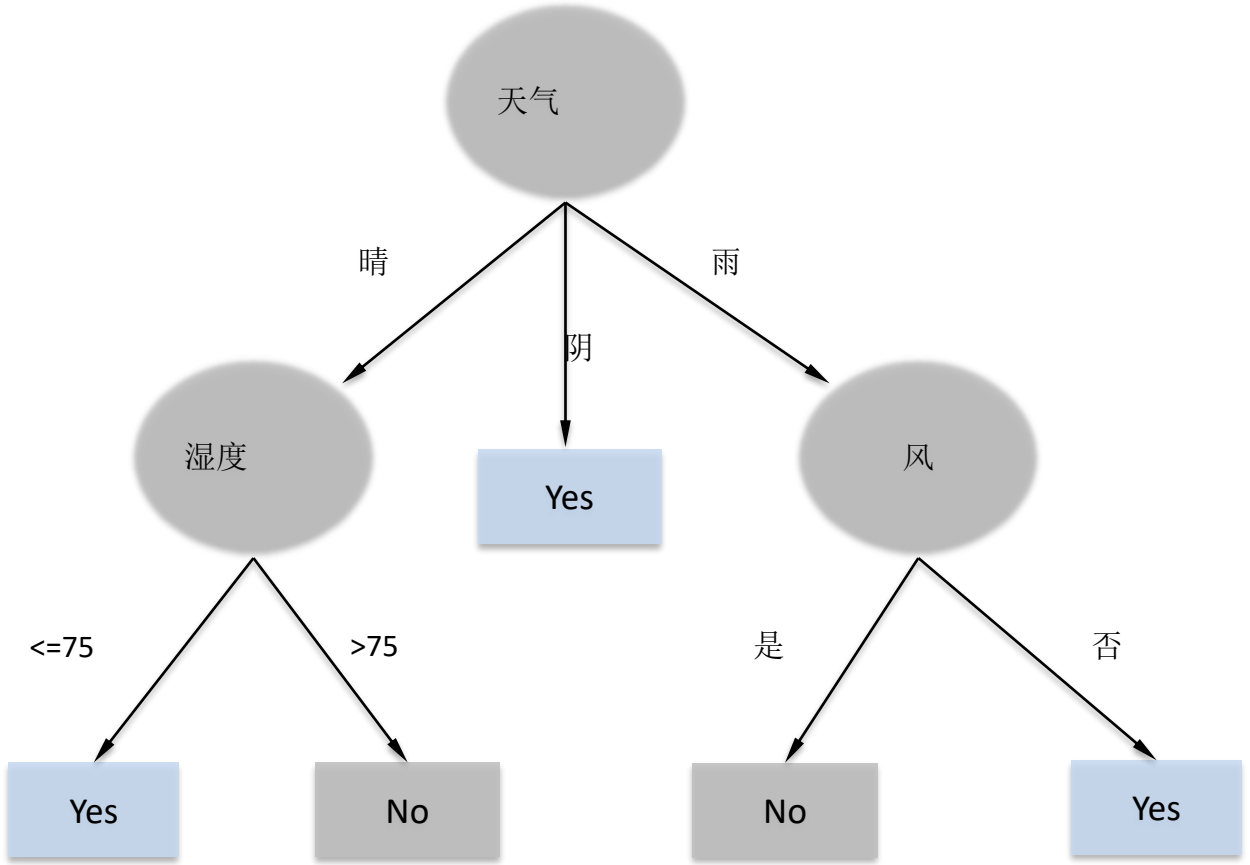
日期	天气	温度(华氏度)	湿度	起风	打球?
1	Sunny	85	85	F	No
2	Sunny	80	90	T	No
8	Sunny	72	95	F	No



决策树

决策树关键词

- 属性选择的先后顺序
- 熵值
- 信息增益
- 信息增益率



决策树

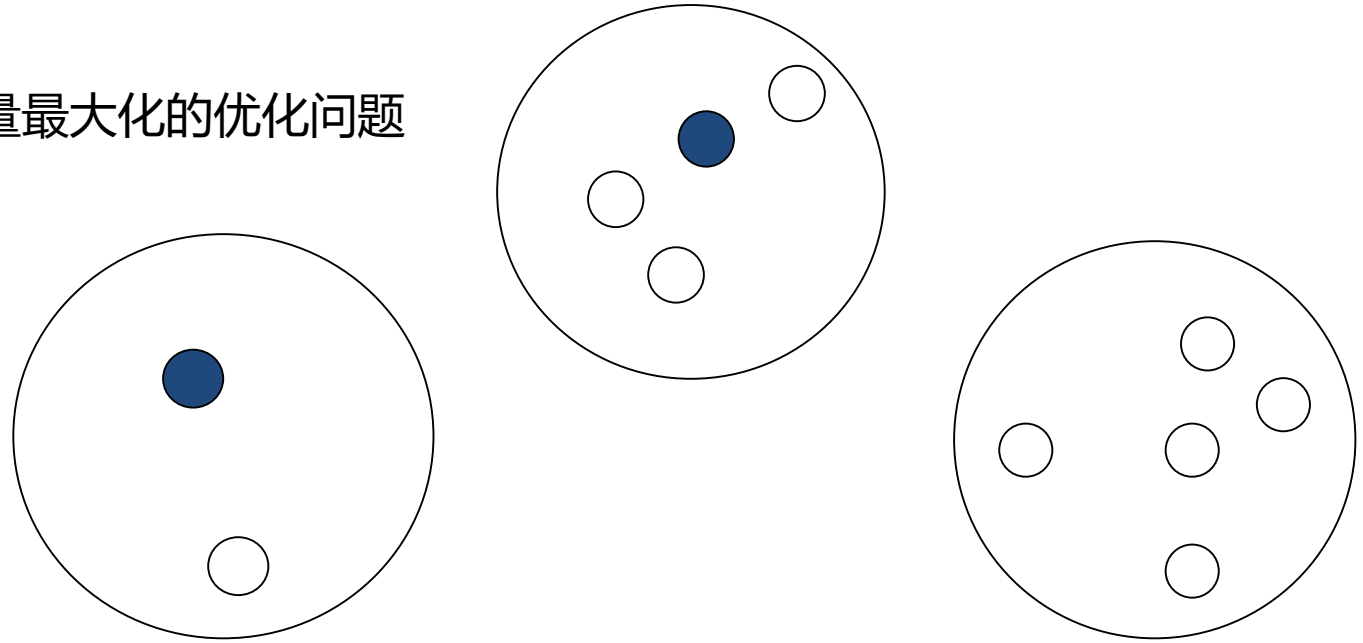
问题：对于给定样本集，如何判断应该在哪个属性上进行拆分

- 每次拆分都存在多种可能，哪个才是较好的选择呢？
- 理想情况：在拆分过程中，当叶节点只拥有单一类别时，将不必继续拆分。
- 目标是寻找较小的树，希望递归过程尽早停止
- 较小的树意味着什么？
- 当前最好的拆分属性产生的拆分中目标类的分布应该尽可能地单一（单纯），多数类占优。
- 如果能测量每一个节点的纯度，就可以选择能产生最纯子节点的那个属性进行拆分；
- 决策树算法通常按照纯度的增加来选择拆分属性。

决策树

纯度的概念

- 纯度度量
  - 当样本中没有两项属于同一类：0
  - 当样本中所有项都属于同一类：1
- 最佳拆分可以转化为选择拆分属性使纯度度量最大化的优化问题



决策树

纯度的度量

- 拆分增加了纯度，但如何将这种增加量化呢，或者如何与其他拆分进行比较呢？
- 用于评价拆分分类目标变量的纯度度量包括
  - 基尼(Gini，总体发散性) CART
  - 熵(entropy，信息量)
  - 信息增益(Gain)
  - 信息增益率 ID3，C4.5，C5.0
- 改变拆分准则（splitting criteria）导致树的外观互不相同

决策树

熵(entropy)

- 信息论中的熵：是信息的度量单位，是一种 对属性“不确定性的度量”。
- 属性的不确定性越大，把它搞清楚所需要的信息量也就越大，熵也就越大。
- Shannon公式： $I(A) = -\log_2 p(A)$
- 其中，I(A)度量事件A发生所提供的信息量，称之为事件A的自信息，P(A)为事件A发生的概率。

决策树

熵(entropy)

- 如果一个属性有N个可能的取值，出现的概率分别为
- 那么这个属性的信息熵为：

$$H = -\sum_{i=1}^{N} p_i \log_2 p_i$$

- 所以，“打球？”的熵为：-(9/14)log2(9/14)-(5/14)log2(5/14)

决策树

打球与否？

日期	天气	温度(华氏度)	湿度	起风	打球?
1	晴	85	85	F	No
2	晴	80	90	T	No
3	阴	83	78	F	Yes
4	雨	70	96	F	Yes
5	雨	68	80	F	Yes
6	雨	65	70	T	No
7	阴	64	65	T	Yes
8	晴	72	95	F	No
9	晴	69	70	F	Yes
10	雨	75	80	F	Yes
11	晴	75	70	T	Yes
12	阴	72	90	T	Yes
13	阴	81	75	F	Yes
14	雨	71	80	T	No
15	阴	85	90	F	?
16	雨	80	79	F	?
17	晴	78	70	T	?

决策树

熵(entropy)

- 根据公式计算总的信息熵，其中数据中总记录数为14，打球9条，不打球5条，表示为I(9,5):

$$I(9,5) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.940$$

计算每个属性的信息熵。

- 天气属性

- 晴：打球记录2条，不打球记录为3条，表示为I(2,3)

$$I(2,3) = -(2/5)\log_2(2/5) - (3/5)\log_2(3/5) = 0.97$$

- 阴：打球记录4条，不打球记录0条，表示为I(4,0)

$$I(4,0) = -(4/4)\log_2(4/4) - (0/4)\log_2(0/4) = 0$$

- 雨：打球记录3条，不打球记录2条，表示为I(3,2)

$$I(3,2) = -(3/5)\log_2(3/5) - (2/5)\log_2(2/5) = 0.97$$

决策树

熵(entropy)

- 天气属性的信息熵（分支数据集熵的加权）

$$(5/14)I(2,3)+(4/14)I(4,0)+(5/14)I(3,2)=0.694$$

- “起风”的信息熵

$$(8/14)I(6,2)+(6/14)I(3,3)=0.892$$

决策树

不同属性拆分的信息增益

- Gain（天气）= I(9,5) - E(天气) = 0.940 - 0.694 = 0.246
- Gain（起风）= I(9,5) - E(起风) = 0.940 - 0.892 = 0.048
- “天气”属性的信息增益值最大，以其三个属性值“晴”、“阴”和“雨”作为该根结点的三个分支

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。

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。

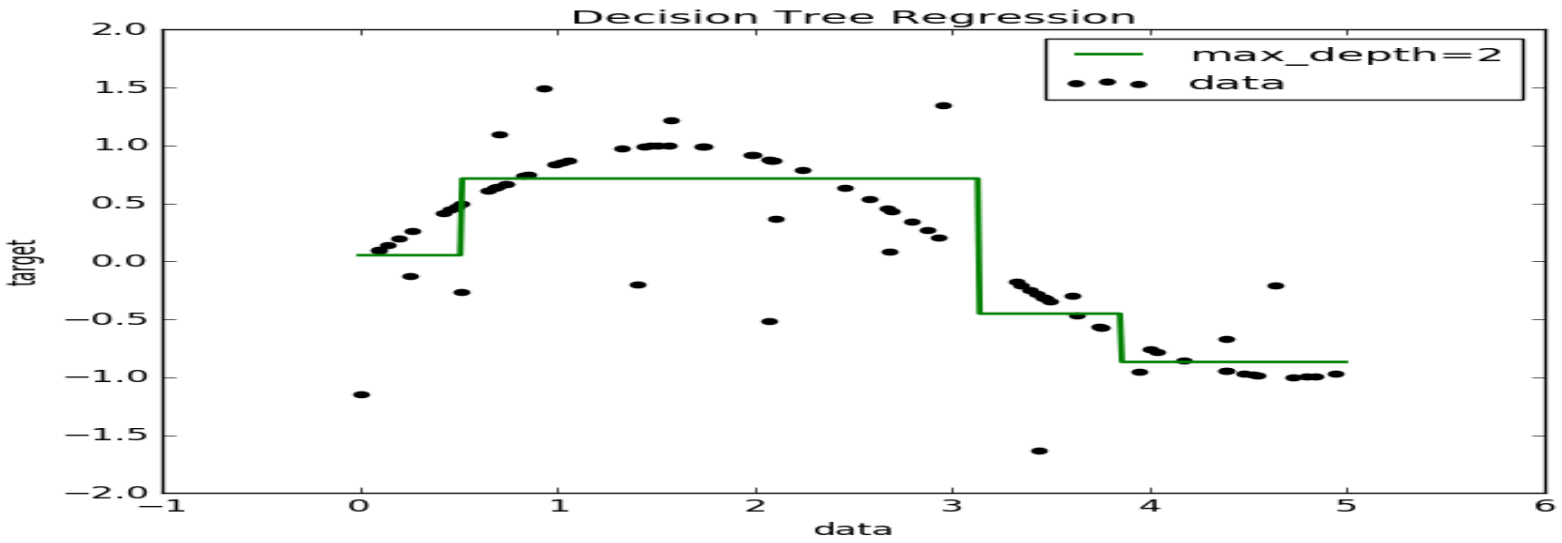
C5.0算法

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

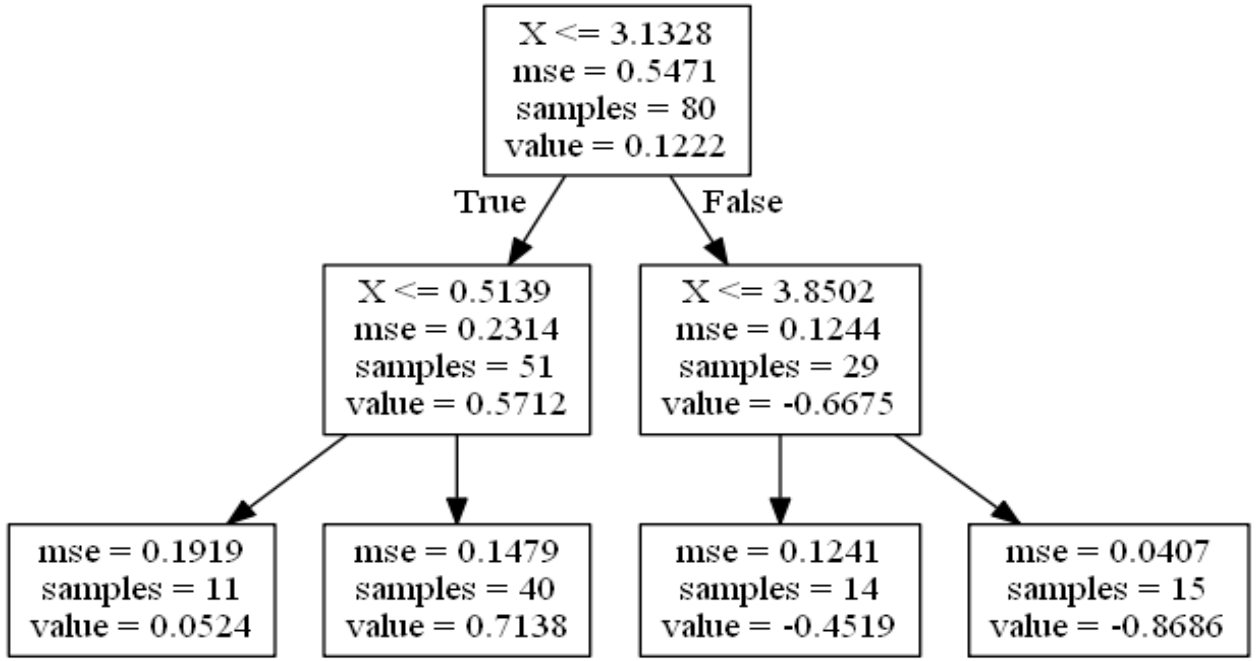
- 分类回归树（Classification And Regression Tree，CART）算法最早由Breiman等人提出，现已在统计领域和数据挖掘技术中普遍使用，Python中的scikit-learn模块的Tree子模块主要使用CART算法来实现决策树。



- 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
iris = load_iris()
clf = DecisionTreeClassifier(random_state=0)
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) #计算交叉验证指标
```

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
0	6	3	male	

- 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。

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)  # 缺失值处理（均值插补）
print(data.head(6))   # 查看数据
```

算法实现

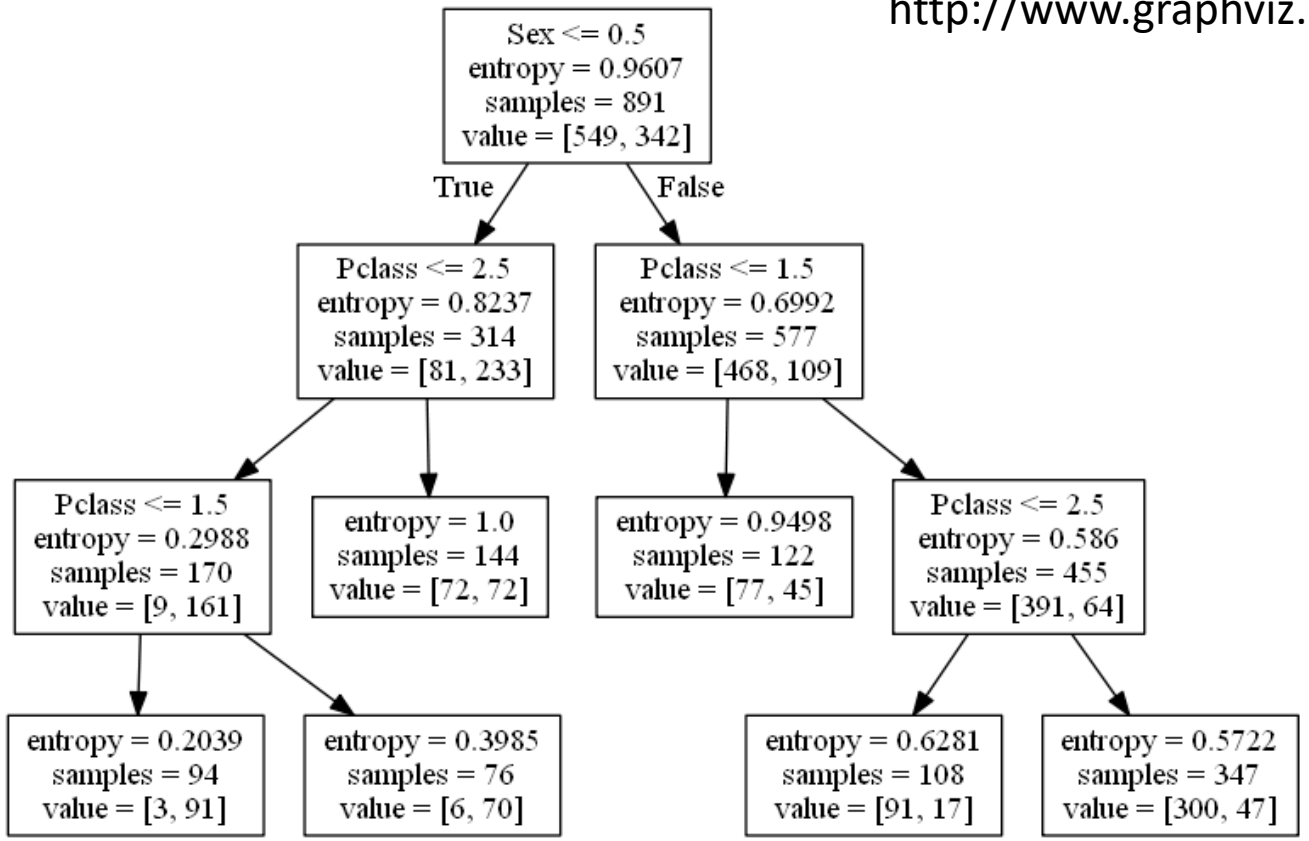
```
X = data.iloc[:, 1:3]     # 为便于展示，未考虑年龄（最后一列）
y = data.iloc[:, 0]
dtc = DTC(criterion='entropy')     # 初始化决策树对象，基于信息熵
dtc.fit(X, y)     # 训练模型
print('输出准确率：', dtc.score(X,y))
# 可视化决策树，导出结果是一个dot文件，需要安装Graphviz才能转换为.pdf或.png格式
with open('../tmp/tree.dot', 'w') as f:
    f = export_graphviz(dtc, feature_names=X.columns,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

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。

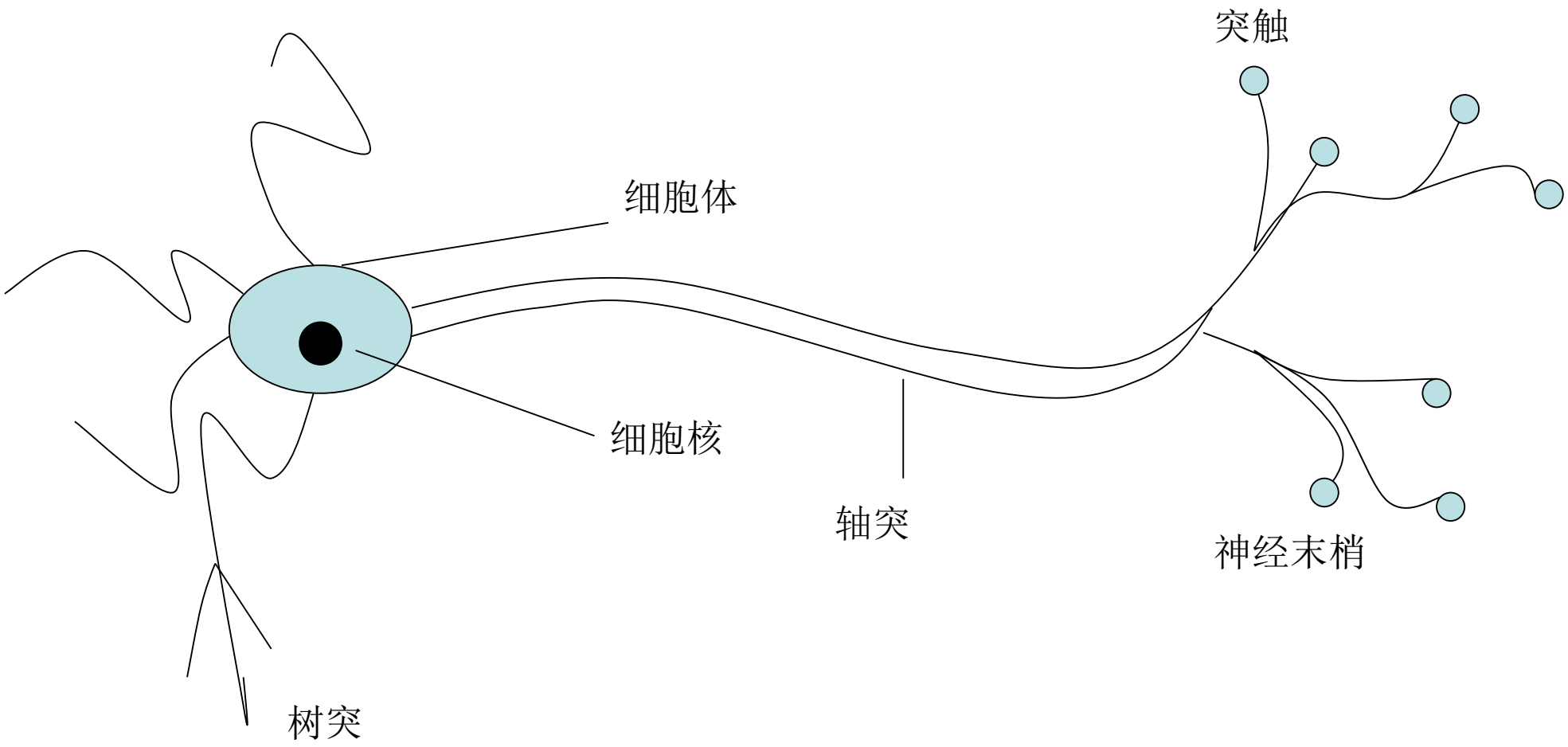
生成的效果图如下：

http://www.graphviz.org/Download_windows.php



目录

神经网络

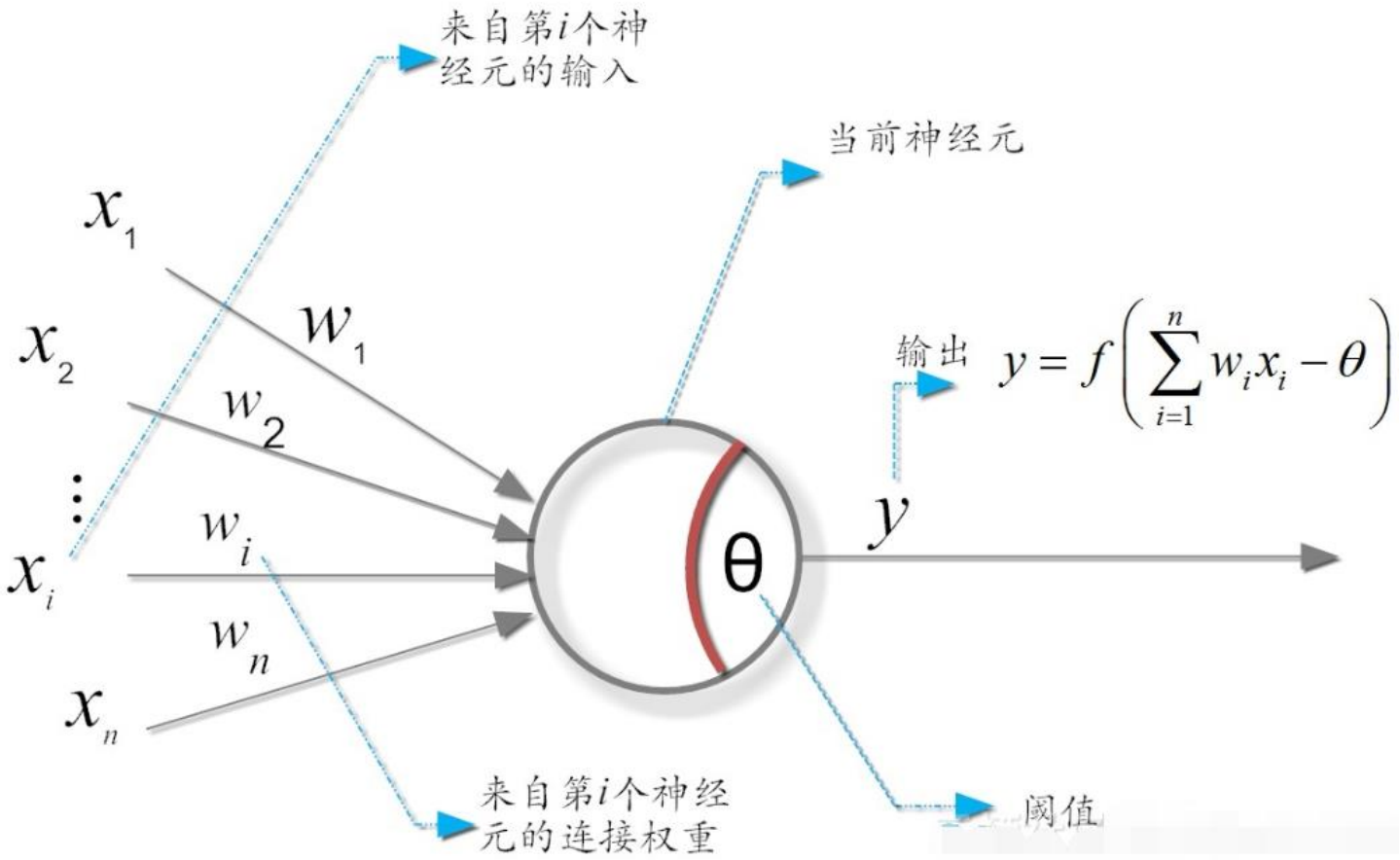


神经网络

- Def：“神经网络是有具有适应性的简单单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界物体所做出的交互反应。”[Kohonen, 1988]
- 神经网络 (neural networks) 最重要的用途是分类。
  - 垃圾邮件识别
  - 疾病判断
  - 猫狗图片分类
- 这种能自动对输入的东西进行分类的机器，就叫做分类器（Classifier）。

神经网络

M-P神经元模型



其中，$f(\sum_{i=1}^{n} \omega_i x_i - \theta)$称为激活函数(activation function)。

神经网络

常用激活函数：

- 阶跃函数

$$sgn(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

- Sigmoid函数

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

若$f = sigmoid(x)$，则

$$f'(x) = f(x)(1 - f(x))$$

神经网络

感知器

- 感知器(Perceptron)由两层神经元组成，输入层接收外界输入信号后传递给输出层，输出层是M-P神经元。

$$y=f(\omega_1 x_1+\omega_2 x_2-\theta)$$

输出层

输入层

神经网络

感知器

- 给定训练数据集，权重$\omega_i(i=1,2,\ldots,n)$以及阈值θ可通过学习得到。
- 阈值θ可看作一个固定输入为-1.0的“哑节点”（dummy node）所对应的连接权重ω_{n+1}，这样权重与阈值的学习就可统一为权重的学习。
- 感知机的学习规则：

  对训练样本(x,y)，若当前感知机的输出为$\hat{y}$，则感知机权重调整：

$$\omega_i \leftarrow \omega_i + \Delta\,\omega_i$$

$$\Delta\,\omega_i = \eta(y-\hat{y})x_i$$

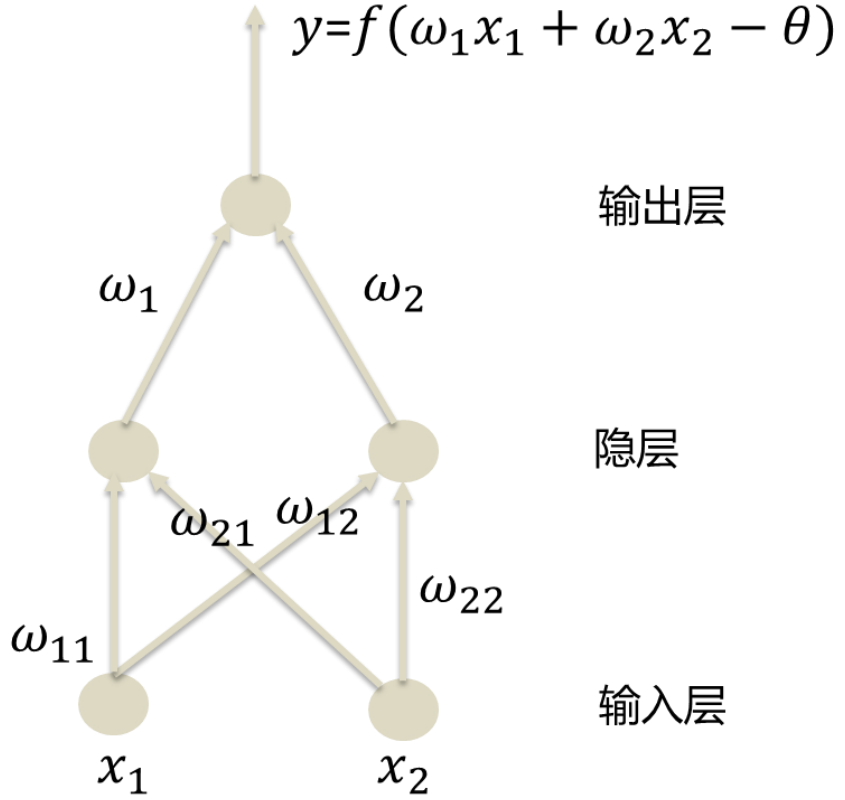
  其中，$\eta \in (0,1)$称为学习率（learning rate）。

  直到$y=\hat{y}$，感知机停止权重调整。

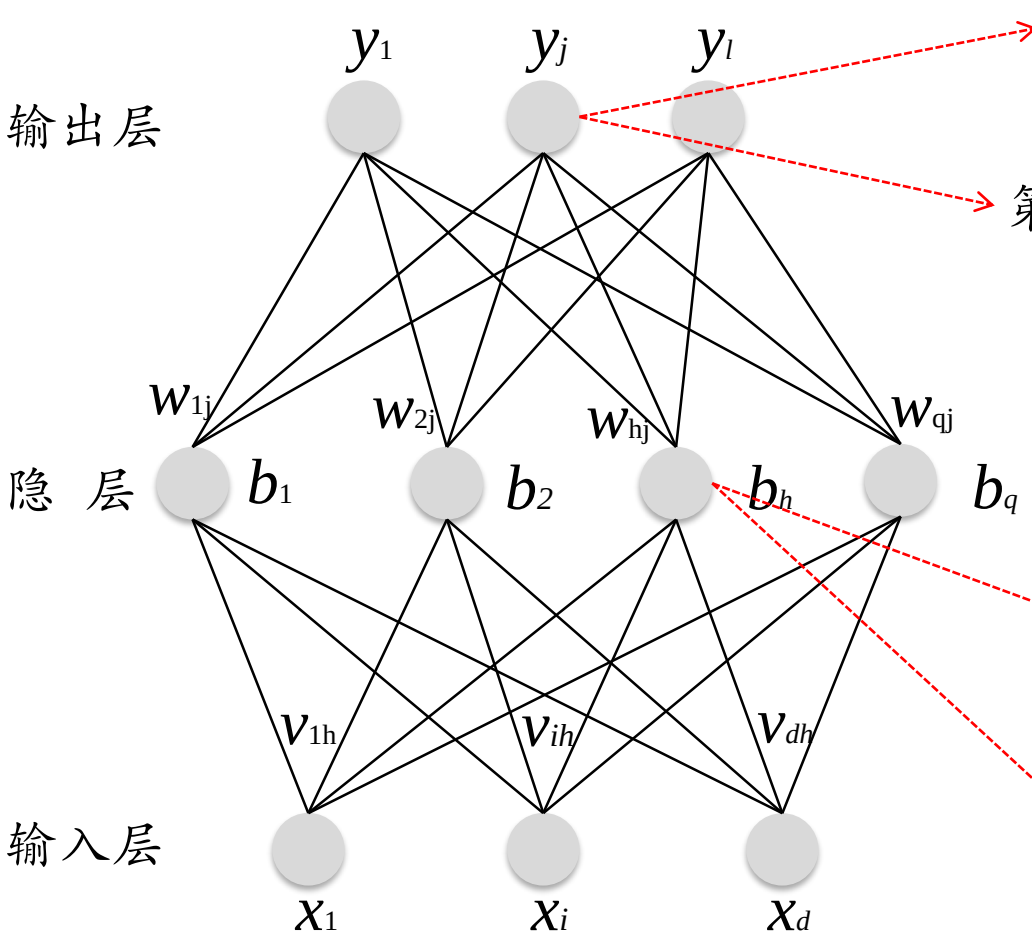
神经网络

多层网络

- 要解决非线性问题，需要考虑使用多层功能神经元。输入层与输出层之间的一层神经元，被称为隐层(hidden layer)。隐层与输出层神经元都是拥有激活函数的功能神经元。



神经网络



第j个输出层神经元的输出：

$$\hat{y}_j = f(b_j - q_j)$$

第j个输出层神经元的输入：

$$b_j = \sum_{h=1}^{g} w_{hj} b_h$$

第h个隐层神经元的输出：

$$b_h = f(a_h - g_h)$$

第h个隐层神经元的输入：

$$a_h = \sum_{i=1}^{d} v_{ih} x_i$$

神经网络

BP算法（误差逆传播算法）

- 假设隐层与输入层神经元的激活函数均为sigmoid函数
- 对训练例(x_k, y_k)，假记神经网络的输出为$\hat{y}_k = (\hat{y}_1^k, \hat{y}_2^k, \ldots, \hat{y}_l^k)$，即

$$\hat{y}_j^k = f(\beta_j - \theta j)$$

则网络在(x_k, y_k)上的均方误差为

$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$$

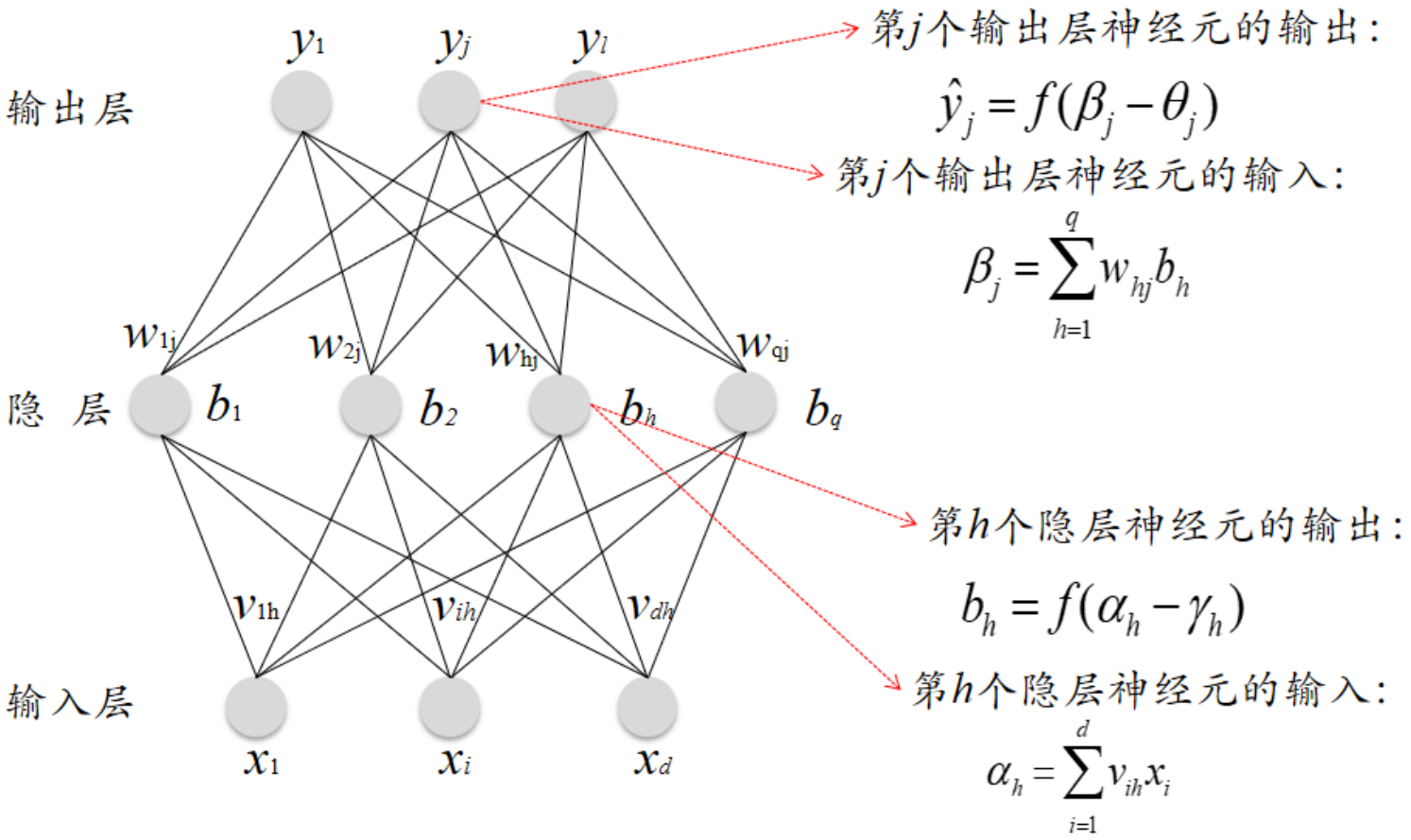
神经网络

BP算法（误差逆传播算法）推导

- $f(x) = sigmoid(x) = \frac{1}{1+e^{-x}}$
- $\hat{y} = f(\beta_j - \theta_j)$
- $E = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j - y_j)^2$
- $\omega_i \longleftarrow \omega_i + \Delta\omega_i$

梯度下降策略：

$$\Delta\omega_i = -\eta \frac{\partial E_k}{\partial w_{hj}}$$



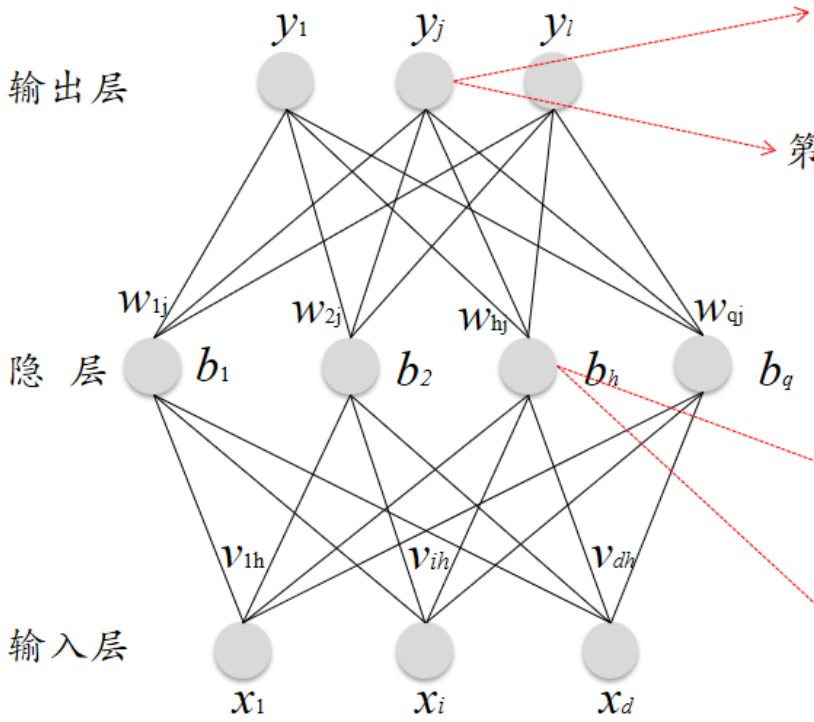
神经网络

BP算法（误差逆传播算法）推导

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \beta_j}\frac{\partial \beta_j}{\partial w_{hj}}$$

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h$$

$$\frac{\partial f}{\partial x} = f(x)(1-f(x))$$



第j个输出层神经元的输出：

$$\hat{y}_j = f(\beta_j - \theta_j)$$

第j个输出层神经元的输入：

$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h$$

第h个隐层神经元的输出：

$$b_h = f(\alpha_h - \gamma_h)$$

第h个隐层神经元的输入：

$$\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$$

$$g_j = -\frac{\partial E_k}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \beta_j} = -(\hat{y}_j - y_j)f'(\beta_j - \theta_j) = \hat{y}_j(1-\hat{y}_j)(y_j - \hat{y}_j)$$

$$\Rightarrow \Delta w_{hj} = -\eta\frac{\partial E_k}{\partial w_{hj}} = \eta g_j b_h$$

神经网络

BP算法（误差逆传播算法）推导

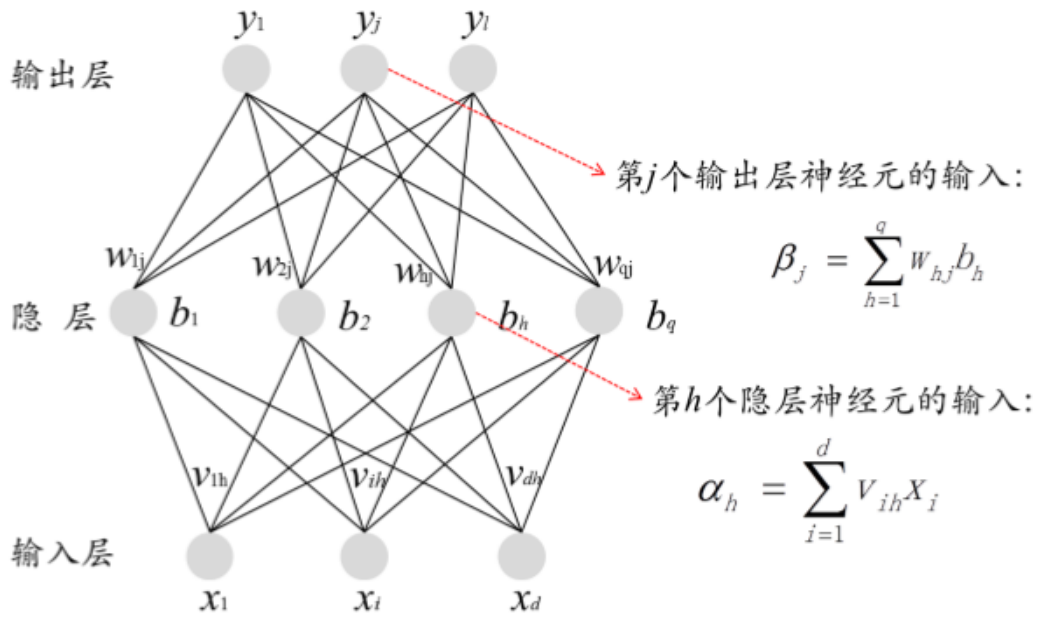
同理可得：

$$\Delta w_{hj} = \eta \hat{y}_j (1-\hat{y}_j)(y_j - \hat{y}_j) b_h$$

$$\nabla \theta_j = -\eta \hat{y}_j (1-\hat{y}_j)(y_j - \hat{y}_j)$$

$$\nabla v_{ih} = \eta b_h (1-b_h) \sum_{j=1}^{l} w_{hj} g_j x_i$$

$$\nabla \gamma_h = -\eta b_h (1-b_h) \sum_{j=1}^{l} w_{hj} g_j$$



隐层神经元的梯度项:

$$e_h = -\frac{\partial E_k}{\partial b_h}\frac{\partial b_h}{\partial \alpha_h} = -\sum_{j=1}^{l}\frac{\partial E_k}{\partial \beta_j}\frac{\partial \beta_j}{\partial b_h} f'(\alpha_h - \gamma_h) = b_h(1-b_h)\sum_{j=1}^{l}\omega_{hj} g_j$$

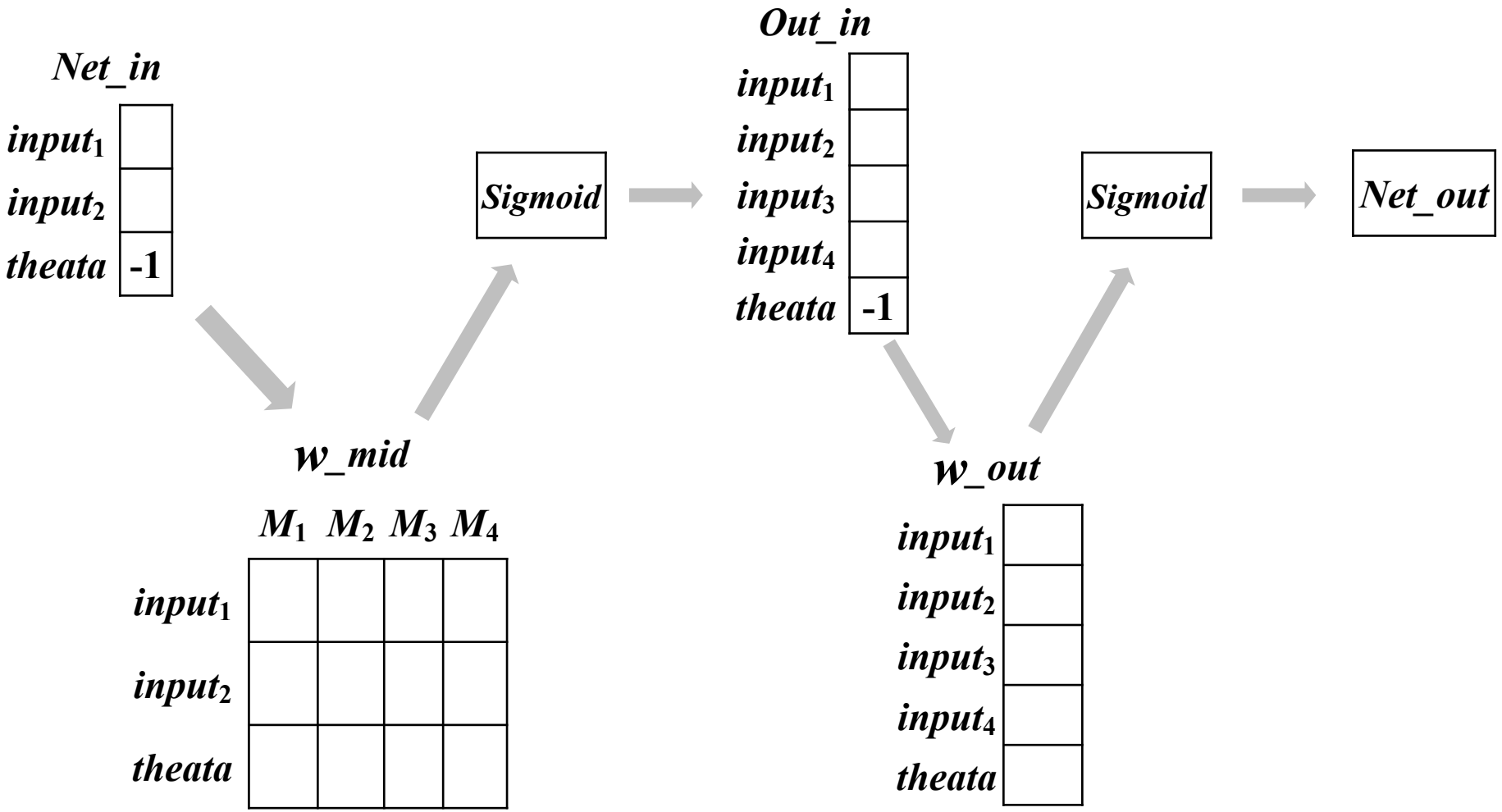
神经网络

BP算法步骤

- **输入**：训练集$D = \{(x_k, y_k)\}_{k=1}^{m}$；
  学习率η
- **过程**：
  在(0,1)范围内随机初始化网络中所有连接权和阈值
  repeat
  for all $(x_k, y_k) \in D$ do
  根据网络输入和当前参数计算当前样本的输出$\hat{y}_k$
  计算输出层神经元的梯度项g_j
  计算隐层神经元的梯度项e_h
  更新连接权ω_{hj}, v_{ih}与阈值θ_j, γ_h
  end for
  Until 达到停止条件
- **输出**：连接权与阈值确定的多层前馈神经网络

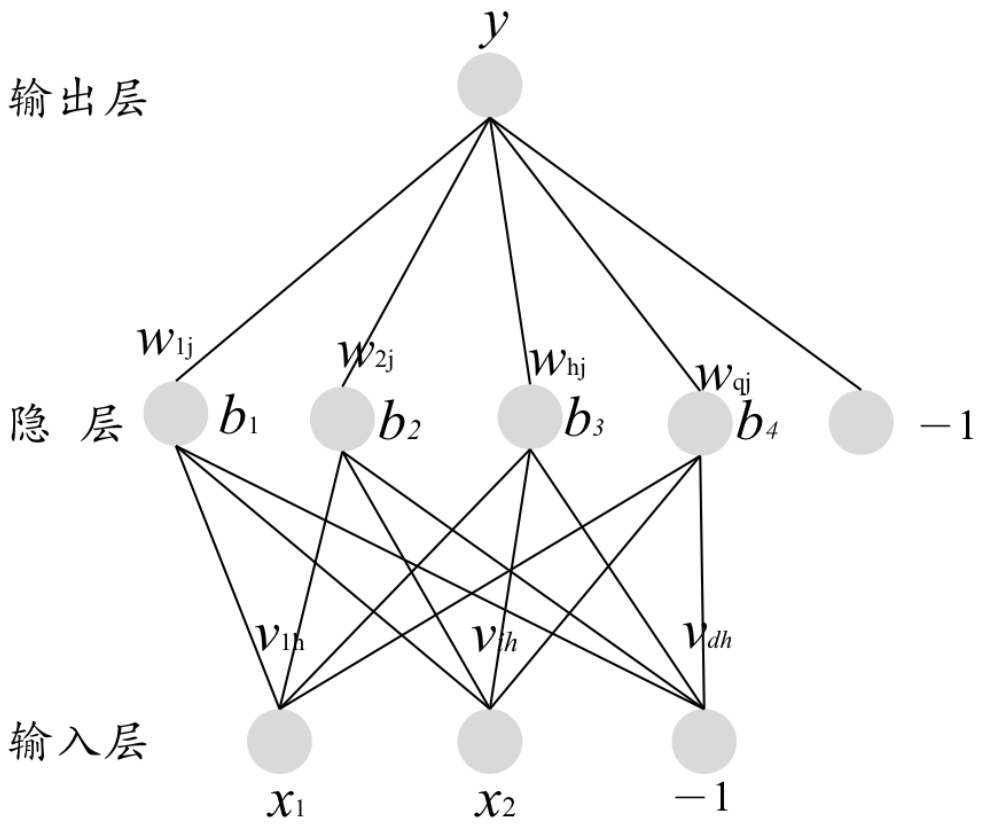
神经网络

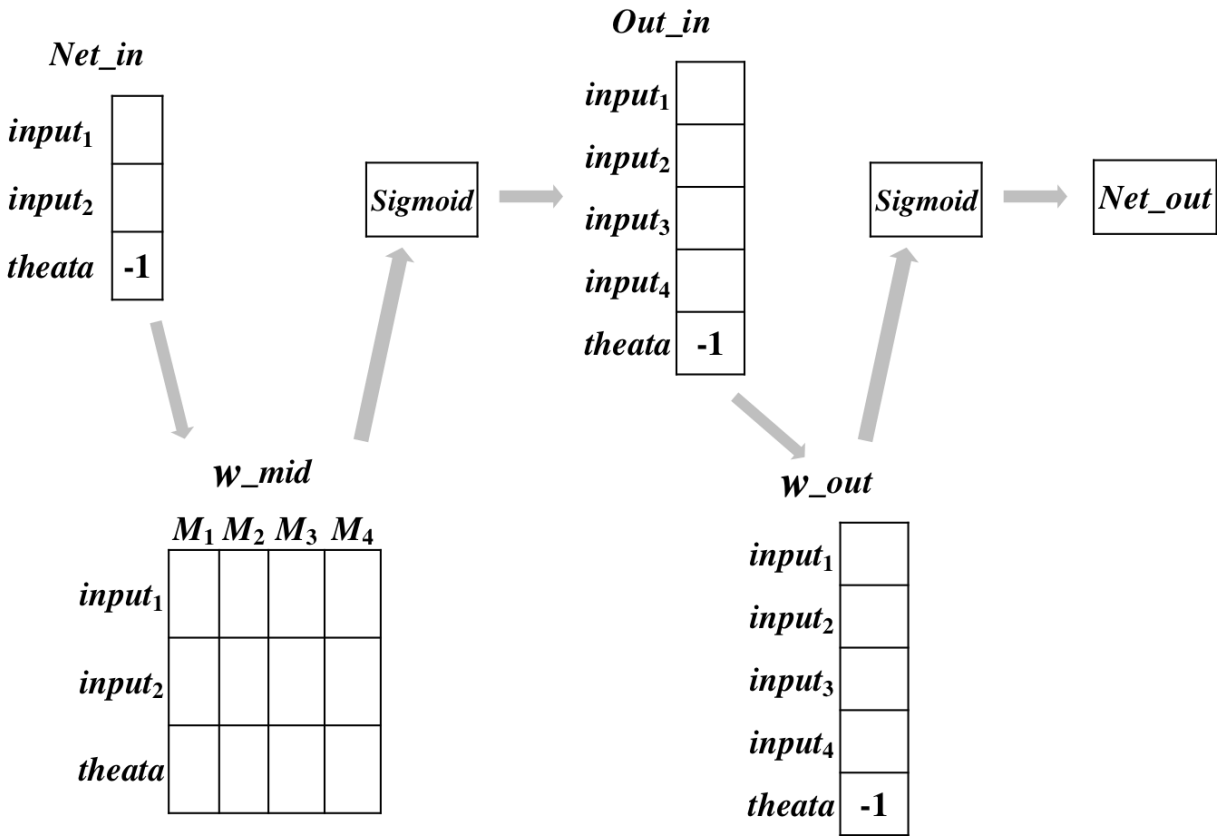
网络训练过程


Net_in
input1
input2
theata
-1
Sigmoid
Out_in
input1
input2
input3
input4
theata
-1
Sigmoid
Net_out
w_mid
M1 M2 M3 M4
input1
input2
theata
w_out
input1
input2
input3
input4
theata

神经网络

网络训练过程


y
输出层
w_{1j}
w_{2j}
w_{hj}
w_{qj}
隐 层
b_1
b_2
b_3
b_4
-1
v_{1h}
v_{ih}
v_{dh}
输入层
x_1
x_2
-1


Net_in
input1
input2
theata
-1
Sigmoid
Out_in
input1
input2
input3
input4
theata
-1
Sigmoid
Net_out
w_mid
M1 M2 M3 M4
input1
input2
theata
w_out
input1
input2
input3
input4
theata
-1

神经网络

Python实现

- from sklearn.neural_network import MLPClassifier
- hidden_layer_sizes 参数：输入为元组，表示在神经网络中每层的神经元数量，其中元组中的第n个元素表示MLP模型第n层的神经元数量
- max_iter参数：最大迭代次数

- fit训练模型
- predict预测

神经网络

Python实现

- coefs_是权重矩阵的列表，其中索引i处的权重值表示第i层到第i+1层之间的权重
- intercepts_是偏差向量的列表，其中索引i处的向量表示添加到第i+1层之间的偏差值

- 另，sklearn中自带混淆矩阵及分类报告
- from sklearn.metrics import classification_report,confusion_matrix

神经网络

优点

- 有良好的自组织学习功能。
- 类神经网络可以建构非线性的模型，模型的准确度高。
- 类神经网络有良好的推广性，对于未知的输入亦可得到正确的输出。
- 类神经网络可以接受不同种类的变量作为输入，适应性强。
- 对异常值不敏感。
- 对噪声数据有比较高的承受能力。
- 以用于分类预测问题。
- 对数据的基本关系几乎不需要做出假设。

神经网络

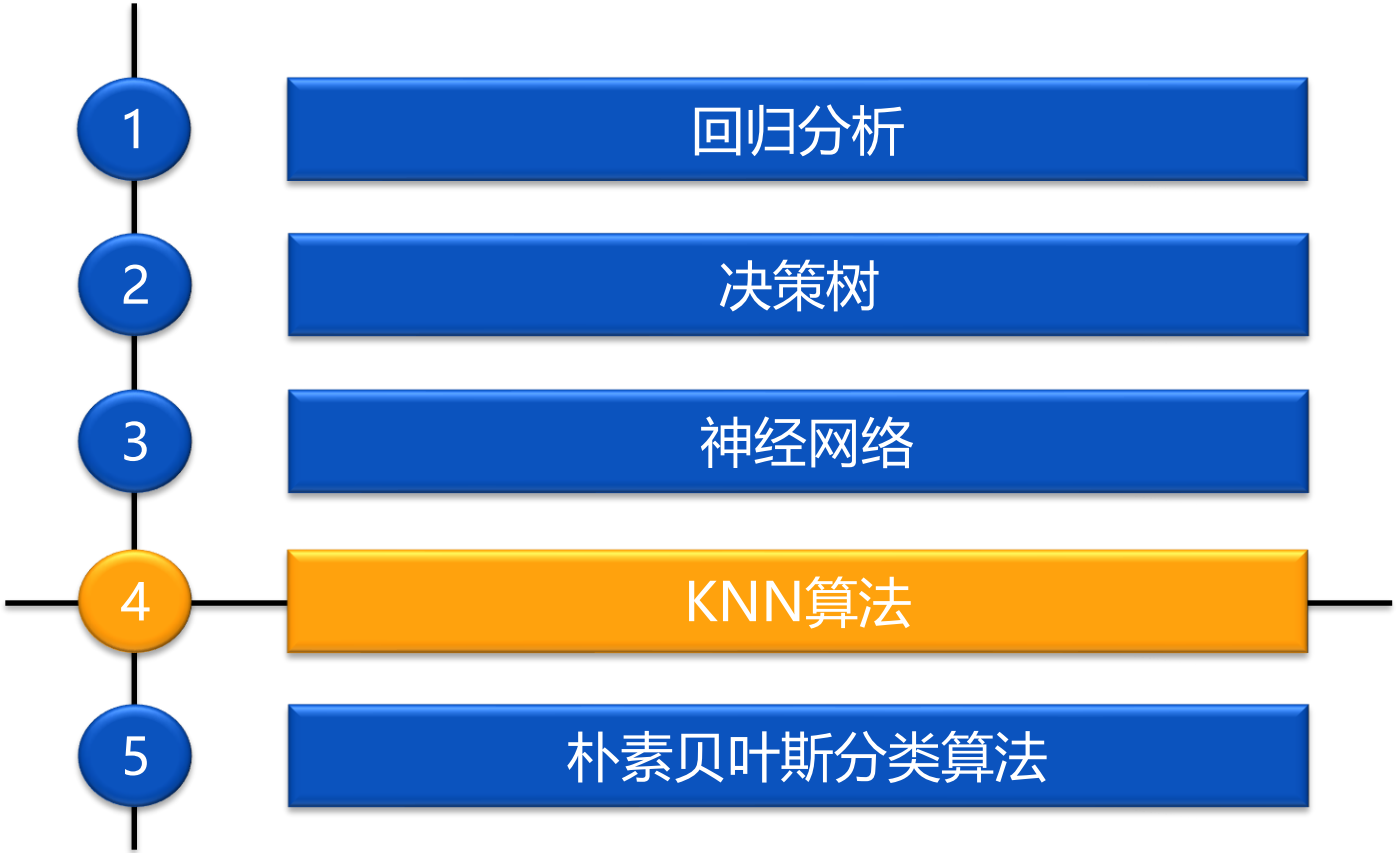
缺点

- 计算量大，训练速度慢，特别是在网络拓扑结构复杂的情况下。
- 容易出现过度拟合。
- 由于其结构的复杂性和结论的难以解释性，神经网络在商业实践中远远没有回归和决策 树应用的广泛，人们对它的理解、接纳还有待提高

其他神经网络

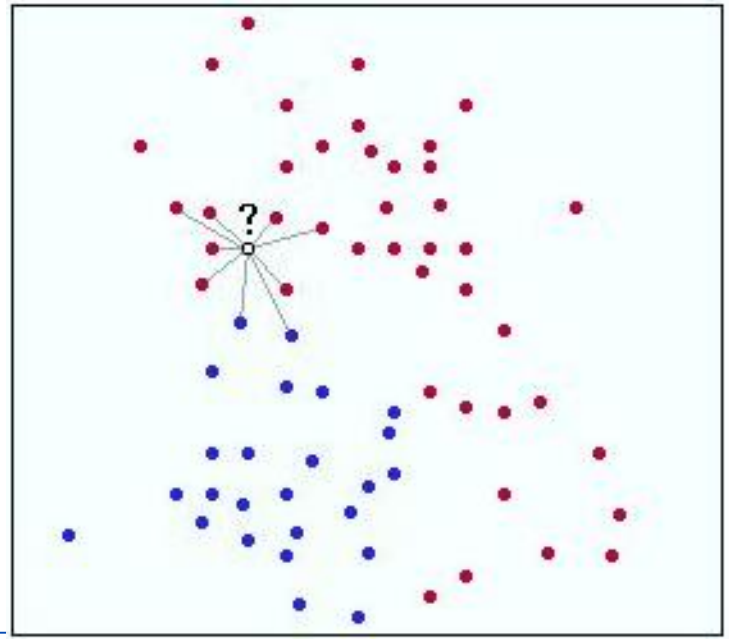
- 卷积神经网络(Convolutional Neual Network,CNN)是一种前馈神经网络，与BP神经网络不同的是，它包括卷积层(alternating convolutional layer)和池层(pooling layer)，在图像处理方面有很好的效果，经常用作解决计算机视觉问题。
- 递归神经网络(RNN)分为时间递归神经网络(Recurrent Neural Network)和结构递归神经网络(Recursive Neural Network)。
- RNN主要用于处理序列数据。在BP神经网络中，输入层到输出层各层间是全连接的，但同层之间的节点却是无连接的。这种网络对于处理序列数据效果很差，忽略了同层间节点的联系，而在序列中同层间节点的关系是很密切的。

目录



kNN算法概述

- kNN算法是k-Nearest Neighbor Classification的简称，即k-近邻分类算法。它的思想很简单：一个样本在特征空间中，总会有k个最相似（即特征空间中最邻近）的样本。其中，大多数样本属于某一个类别，则该样本也属于这个类别。
- 近朱者赤，近墨者黑。
- 是理论上比较成熟的方法，也是最简单的机器学习算法之一。
- 行业应用：
  - 客户流失预测
  - 欺诈侦测等（更适合于稀有事件的分类问题）



kNN算法

- 计算步骤如下：

  1）算距离：给定测试对象，计算它与训练集中的每个对象的距离

  2）找邻居：圈定距离最近的k个训练对象，作为测试对象的近邻

  3）做分类：根据这k个近邻归属的主要类别，来对测试对象分类

- kNN是一种懒惰算法，平时不好好学习，考试（对测试样本分类）时才临阵磨枪（临时去找k个近邻）。懒惰的后果：构造模型很简单，但在对测试样本分类地的系统开销大，因为要扫描全部训练样本并计算距离。

kNN算法

算距离

- 什么是合适的距离衡量？

  距离越近应该意味着这两个点属于一个分类的可能性越大。

- 计算的距离衡量包括欧式距离、夹角余弦等。

  欧式距离：

$$d(x,y)=\sqrt{\sum_{k=1}^{n}(x_k-y_k)^2}$$

  注：有些距离测量方法会受维数的影响，特别是欧氏距离在属性数增加时判断的能力减弱。

- 对于文本分类来说，使用余弦(cosine)来计算相似度就比欧式(Euclidean)距离更合适。

kNN算法

典型的距离定义

距离	定义式	说明
绝对值距离	$d_{ij}(1)=\sum_{k=1}^{p}\lvert x_{ik}-x_{jk}\rvert$	绝对值距离是在一维空间下进行的距离计算
欧式距离	$d_{ij}(2)=\sqrt{\sum_{k=1}^{p}(x_{ik}-x_{jk})^2}$.	欧式距离是在二维空间下进行的距离计算
闵可夫斯基距离	$d_{ij}(q)=\left[\sum_{k=1}^{p}(x_{ik}-x_{jk})^q\right]^{1/q}$，$q>0$.	闵可夫斯基距离是在 q 维空间下进行的距离计算
切比雪夫距离	$d_{ij}(\infty)=\max_{1\le k\le p}\lvert x_{ik}-x_{jk}\rvert$.	切比雪夫距离是 q 取正无穷大时的闵可夫斯基距离，即切比雪夫距离是在 $+\infty$ 维空间下进行的距离计算
Lance 距离	$d_{ij}(L)=\sum_{k=1}^{p}\frac{\lvert x_{ik}-x_{jk}\rvert}{x_{ik}+x_{jk}}$	减弱极端值的影响能力
归一化距离	$d_{ij}=\sum_{k=1}^{p}\frac{\lvert x_{ik}-x_{jk}\rvert}{\max(x_k)-\min(x_k)}$	自动消除不同变量间的纲量影响，其中每个变量 k 的距离取值均是 [0,1]

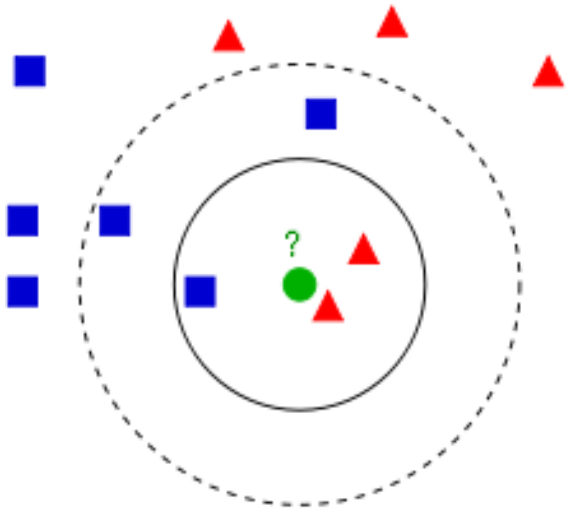
kNN算法

做分类

- 投票决定：少数服从多数，近邻中哪个类别的点最多就分为该类。

- 加权投票法：根据距离的远近，对近邻的投票进行加权，距离越近则权重越大（权重为距离平方的倒数）

- 投票法没有考虑近邻的距离的远近，距离更近的近邻也许更应该决定最终的分类，所以加权投票法更恰当一些。



kNN算法实现流程

a
- 计算已知类别数据集中的点与当前点之间的距离；

b
- 按照距离递增次序排序；

c
- 选取与当前点距离最小的k个点；

d
- 确定前k个点所在类别对应的出现频率；

e
- 返回前k个点出现频率最高的类别作为当前点的预测分类。

kNN算法

- 1、优点

  简单，易于理解，易于实现，无需估计参数，无需训练

  适合对稀有事件进行分类（例如当流失率很低时，比如低于0.5%，构造流失预测模型）

  特别适合于多分类问题(multi-modal,对象具有多个类别标签)，例如根据基因特征来判断其功能分类，kNN比SVM的表现要好

- 2、缺点

  懒惰算法，对测试样本分类时的计算量大，内存开销大，评分慢

  可解释性较差，无法给出决策树那样的规则。

kNN算法

Python实现

- sklearn库中提供KNeighborsClassifier实现kNN算法，此外，还提供RadiusNeighborsClassifier（非均匀采样时比较合适，以半径为选取方法）做最近邻分类
- sklearn.neighbors.KNeighborsClassifier(n_neighbors=5 #邻居数，默认为5
  , weights='uniform' #用于预测的权重方法
  , algorithm='auto' #用于计算最近邻的算法（ball_tree、kd_tree、brute、auto）
  , leaf_size=30 #传递给BallTree 或KDTree 叶大小
  , p=2 #
  , metric='minkowski' #用于树的度量距离
  , metric_params=None #度量参数
  , **kwargs)
- from sklearn.neighbors import KNeighborsClassifier

kNN算法

练习

- 使用kNN算法对iris数据集进行分类

目录

朴素贝叶斯

定义

- 朴素贝叶斯分类器是基于贝叶斯定理与特征条件独立同分布假设的分类方法。
- 贝叶斯公式：

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$

- 简单的说，贝叶斯定理是基于假设的先验概率、给定假设下观察到不同数据的概率，提供了一种计算后验概率的方法。
- 独立同分布：指随机过程中，任何时刻的取值都为随机变量，如果这些随机变量服从同一分布，并且互相独立，那么这些随机变量是独立同分布。
- 即：

$$P(x|c_k)=P(x_1,\mathrm{L}\ ,x_n|c_k)=\prod_{j=1}^{n}P(x_j|c_k)$$

朴素贝叶斯

原理（一）

- 对于贝叶斯网络分类器，若某一待分类的样本D，其分类特征值为 $x=(x_1,x_2,...,x_n)$，则样本D 属于类别yj 的概率 $P(Y=y_j|X=x_1,\mathrm{L}\ X=x_n),(j=1,\mathrm{L},m)$ 应满足下式： $\max P(Y=y_j|X=x)$
- 而由贝叶斯公式：

$$P(Y=y_j|X=x)=\frac{P(X=x/Y=y_j)P(Y=y_j)}{P(X=x)}$$

- 得

$$\max P(Y=y_j|X=x)\ \Rightarrow \max P\left(Y=y_j\right)P(X=x|Y=y_j)$$

- 因此估计 $P(Y=y_j|X=x)$ 的问题就转化为如何基于训练集数据D来估计先验概率 $P\left(Y=y_j\right)$ 和条件概率 $P(X=x|Y=y_j)$

朴素贝叶斯

原理（二）：$P\left(Y=y_j\right)$的计算

- 大数定理：

设μ_n是n次独立试验中事件A发生的次数，且事件A在每次试验中发生的概率为P，则对任意正数ε，有：

$$\lim_{n\to\infty} P(|\frac{\mu_n}{n}-p<\varepsilon|)=1$$

该定律是切贝雪夫大数定律的特例，其含义是，当n足够大时，事件A出现的频率将几乎接近于其发生的概率，即频率的稳定性。

- 当n足够大时，事件A出现的频率将几乎接近于其发生的概率，即频率的稳定性。
- 根据大数定理，当训练集包含充足的独立同分布样本时，$P\left(Y=y_j\right)$可通过各类样本出现的频率来进行估计。

朴素贝叶斯

原理（三）： $P(X=x|Y=y_j)$

- 在朴素贝叶斯算法中，假设样本独立同分布，此时，

$$P\left(X=x|Y=y_j\right)=P\left(X=x_1,\mathrm{L}\ ,X=x_n|Y=y_j\right)$$

$$=\prod_{i=1}^{n}P\left(X=x_i|Y=y_j\right)$$

- 所以，

$$\max P\left(Y=y_j\right)P(X=x|Y=y_j)$$

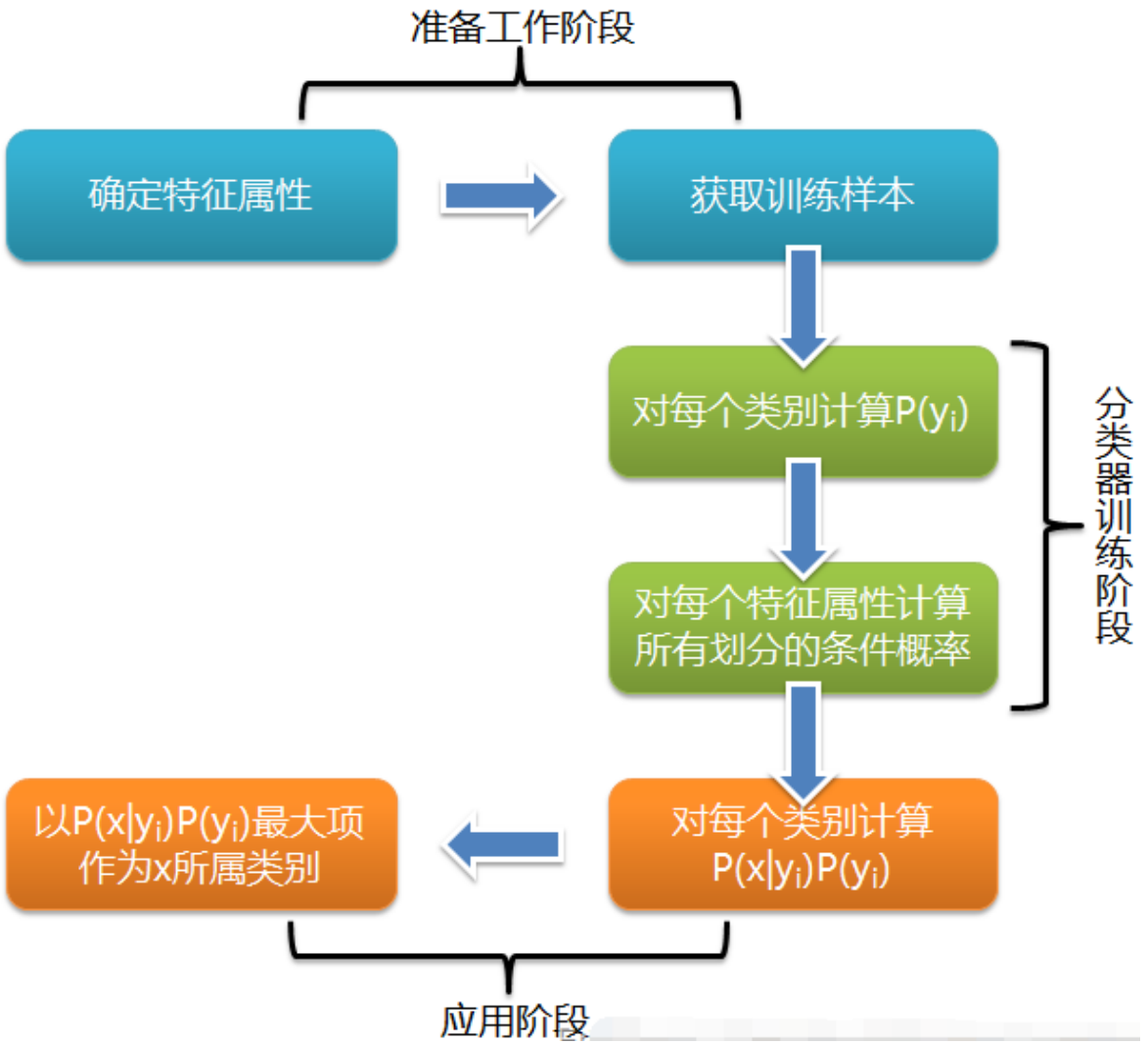
$$=\max P\left(Y=y_j\right)\prod_{i=1}^{n}P\left(X=x_i|Y=y_j\right)$$

- 假设共有 m种标签，我们只需计算 $P(y_k)P(x_1,\cdots,x_n|y_k),k=1,2,\cdots,m$，取最大值作为预测的分类标签，即：

$$\hat{y}=\arg\max_k P(y)\prod_{i=1}^{n}P(x_i|y_k)$$

朴素贝叶斯

算法处理流程



高斯朴素贝叶斯

- 原始的朴素贝叶斯只能处理离散数据，当 $X_1,\cdots,X_n$ 是连续变量时，我们可以使用高斯朴素贝叶斯（Gaussian Naive Bayes）完成分类任务。当处理连续数据时，一种经典的假设是：与每个类相关的连续变量的分布是基于高斯分布的，故高斯贝叶斯的公式如下：

$$P(x_i = v \mid y_k) = \frac{1}{\sqrt{2\pi\sigma_{y_k}^2}} \exp\left(-\frac{(v-\mu_{y_k})}{2\sigma_{y_k}^2}\right)$$

- 其中 μ_y ， $\sigma_{y_k}^2$ 表示表示全部属于类 y_k 的样本中变量 X_i 的均值和方差

多项式朴素贝叶斯

- 多项式朴素贝叶斯（Multinomial Naïve Bayes）经常被用于处理多分类问题，比起原始的朴素贝叶斯分类效果有较大的提升。其公式如下：

$$P(x_i \mid y_k) = \frac{N_{y_k i} + \alpha}{N_y + \alpha n}$$

- 其中 $N_{y_k i} = \sum_{x \in T} x_i$ 表示在训练集T 中类 y_k 具有特征 i的样本的数量， $N_y = \sum_{i=1}^{|T|} N_{yi}$ 表示训练集 T中类 y_k 的特征总数。平滑系数 $\alpha > 0$ 防止零概率的出现，当 $\alpha = 1$ 称为拉普拉斯平滑，而 $\alpha < 1$ 称为Lidstone平 滑。

朴素贝叶斯

Python实现

- naive Bayes is a decent classifier, but a bad estimator
- 高斯朴素贝叶斯
- 构造方法：sklearn.naive_bayes.GaussianNB
- GaussianNB 类构造方法无参数，属性值有：
  - class_prior_ #每一个类的概率
  - theta_ #每个类中各个特征的平均
  - sigma_ #每个类中各个特征的方差
- 注：GaussianNB 类无score 方法

朴素贝叶斯

Python实现——高斯朴素贝叶斯

```
import numpy as np
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
Y = np.array([1, 1, 1, 2, 2, 2])
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X, Y)
```

朴素贝叶斯

Python实现

- 多项式朴素贝叶斯——用于文本分类
- 构造方法：

```
sklearn.naive_bayes.MultinomialNB(alpha=1.0 #平滑参数
                                  , fit_prior=True #学习类的先验概率
                                  , class_prior=None) #类的先验概率
```

朴素贝叶斯

Python实现——多项式朴素贝叶斯

```
import numpy as np
X = np.random.randint(5, size=(6, 100))
y = np.array([1, 2, 3, 4, 5, 6])
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(X, y)
```

Python实现

用高斯朴素贝叶斯对iris数据集进行分类，实现代码如下：

- from sklearn import datasets
- iris = datasets.load_iris() # 读取iris数据集
- from sklearn.naive_bayes import GaussianNB # 使用高斯贝叶斯模型
- clf = GaussianNB() # 设置分类器
- clf.fit(iris.data,iris.target) # 训练分类器
- y_pred = clf.predict(iris.data) # 预测
- print("Number of mislabeled points out of a total %d points : %d" % (iris.data.shape[0],(iris.target != y_pred).sum()))

Thank you!

泰迪科技：www.tipdm.com
热线电话：40068-40020