《Python与数据挖掘》

第7章 分类与预测

讲师：武永亮

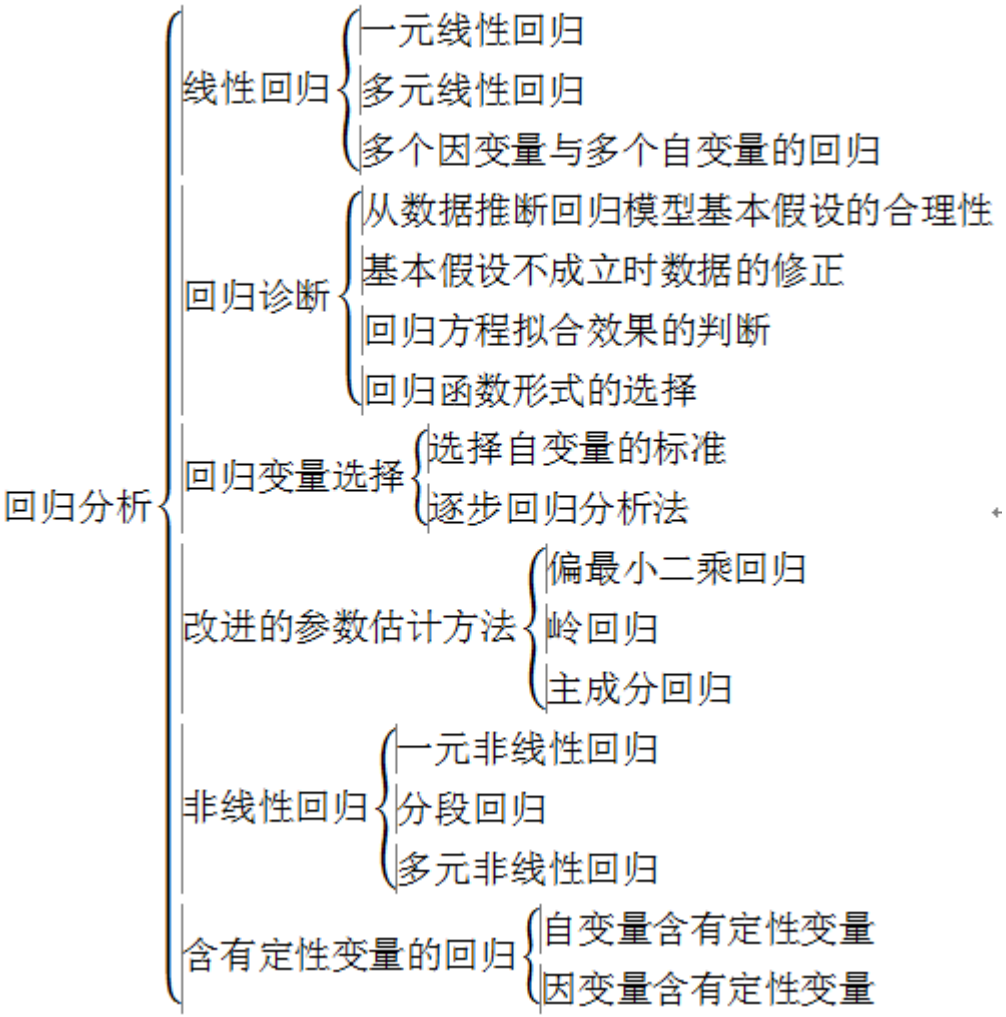
目录

分类与预测的概述

- 狭义的数据挖掘（或机器学习）有一个较为固定的流程，包括获取数据、数据清洗、选择合适模型、应用算法求解、参数调优与验证等。同时，因为相关任务往往受到数据变化，计算能力和经验性判断等的限制，所以这个过程中没人能一劳永逸。这个流程中的每一处细节处理，是数据挖掘人才的试金石。
- 分类与预测是机器学习中有监督学习任务的代表。一般认为：广义的预测任务中，要求估计连续型预测值时，是“回归”任务；要求判断因变量属于哪个类别时，是“分类”任务。

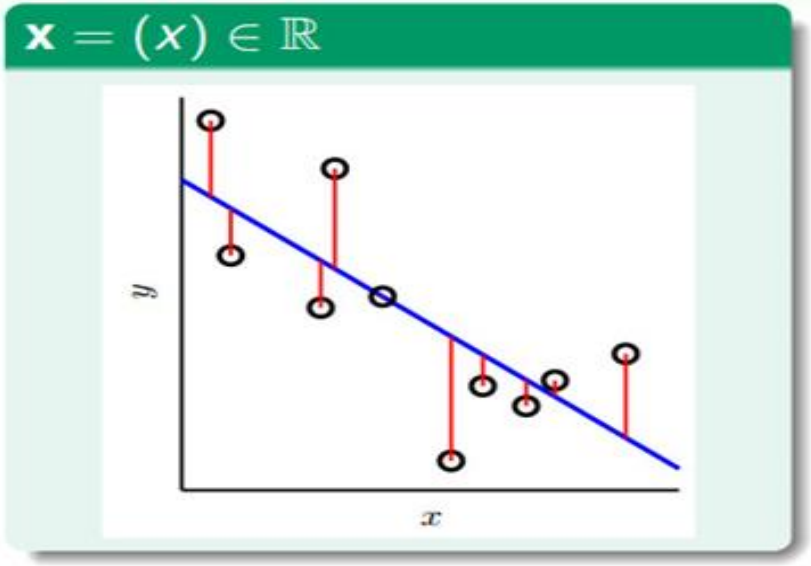
回归分析

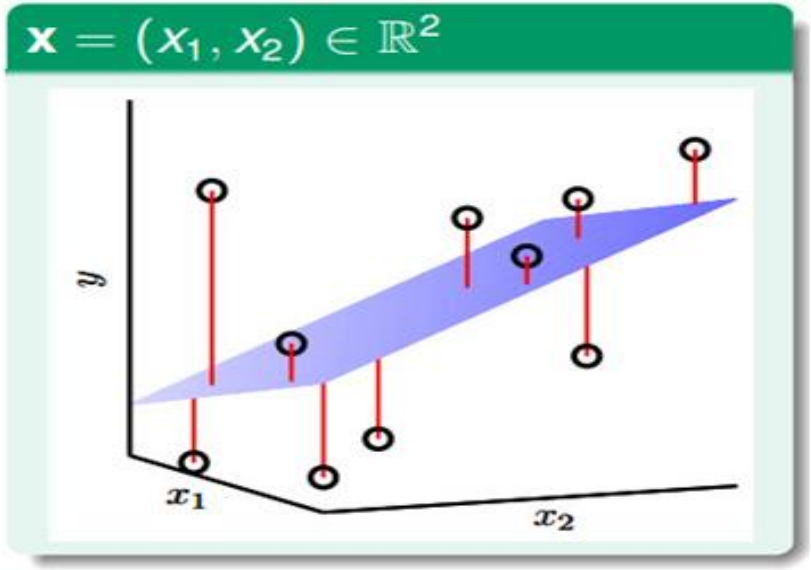
- 回归分析是一项预测性的建模技术。它的目的是通过建立模型来研究因变量和自变量之间的显著关系，即多个自变量对（一个）因变量的影响强度，预测数值型的目标值。
- 常用的回归分析技术是线性回归、逻辑回归、多项式回归和岭回归等。作为入门书籍，在此主要介绍前两种模型的原理和具体实现。



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：

$$f(x) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b \times 1$$

- 其中权重wi和常数项b是待定的，这就意味着将输入的自变量按一定比例加权求和，得到预测值输出。
- Python有强大的第三方扩展模块scikit-learn，实现了大部分的数据挖掘基础算法，包括线性回归。
- 我们使用sklearn快速实现线性回归模型。这个例子就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。

线性回归例子

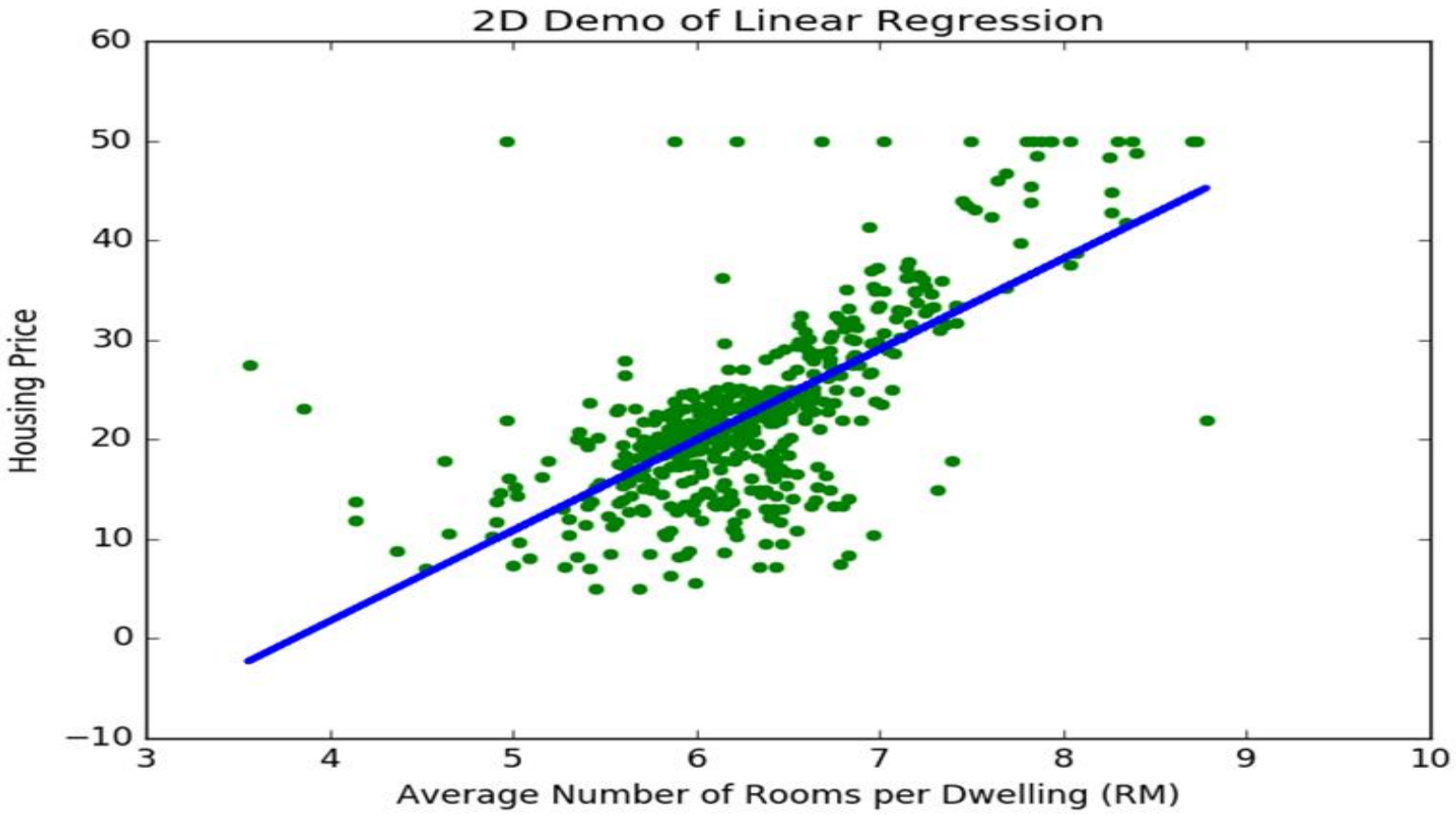
- 波士顿房价预测的部分数据如下表：

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：上面的数据表中第3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。

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，我们可以由线性模型拓展出一类广义线性模型，即通过非线性的**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地表示为：

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

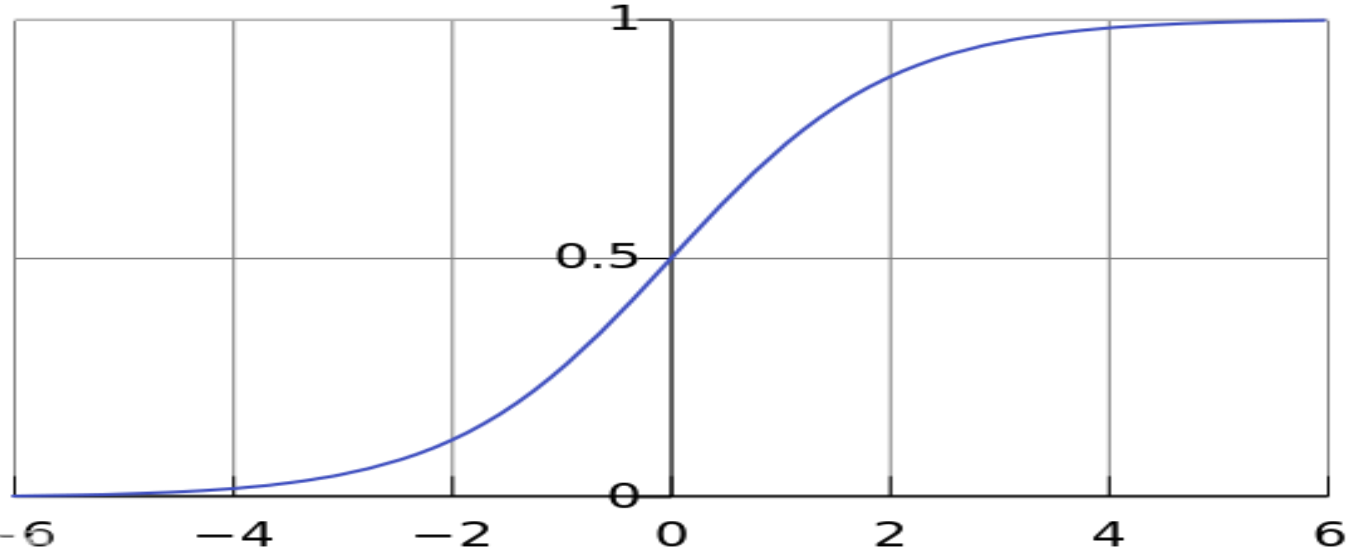
- 其中，η 是待定的参数；$T(y)$ 是充分统计量，通常 $T(y)=y$。
- 而伯努利分布的概率密度函数为：

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

- 其中，h是由假设衍生的关联函数，y的可能取值为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 $(-\infty,+\infty)$ 变化时，h 的取值范围恰好为(0,1)。由于h 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

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

- 确定合适的权重w，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术语正类的概率。因此，关联函数亦被称为“激活函数”，如同神经元接受的足够的刺激，才会变得兴奋。
- 通常，确定权重w的方法是：最大似然估计（maximum likelihood estimation）

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, prefix='rank', columns=['rank'],
drop_first=True)
print data_dum.tail(5)    # 查看数据框的最后五行
# 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是类 C_i中的样本数。对一个给定样本，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划分样本的信息熵值为 $$E(A)=\sum_{j=1}^{k}\frac{s_{1j}+s_{2j}+\cdots+s_{mj}}{s}I(s_{1j},s_{2j},\cdots,s_{mj})$$
- 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。

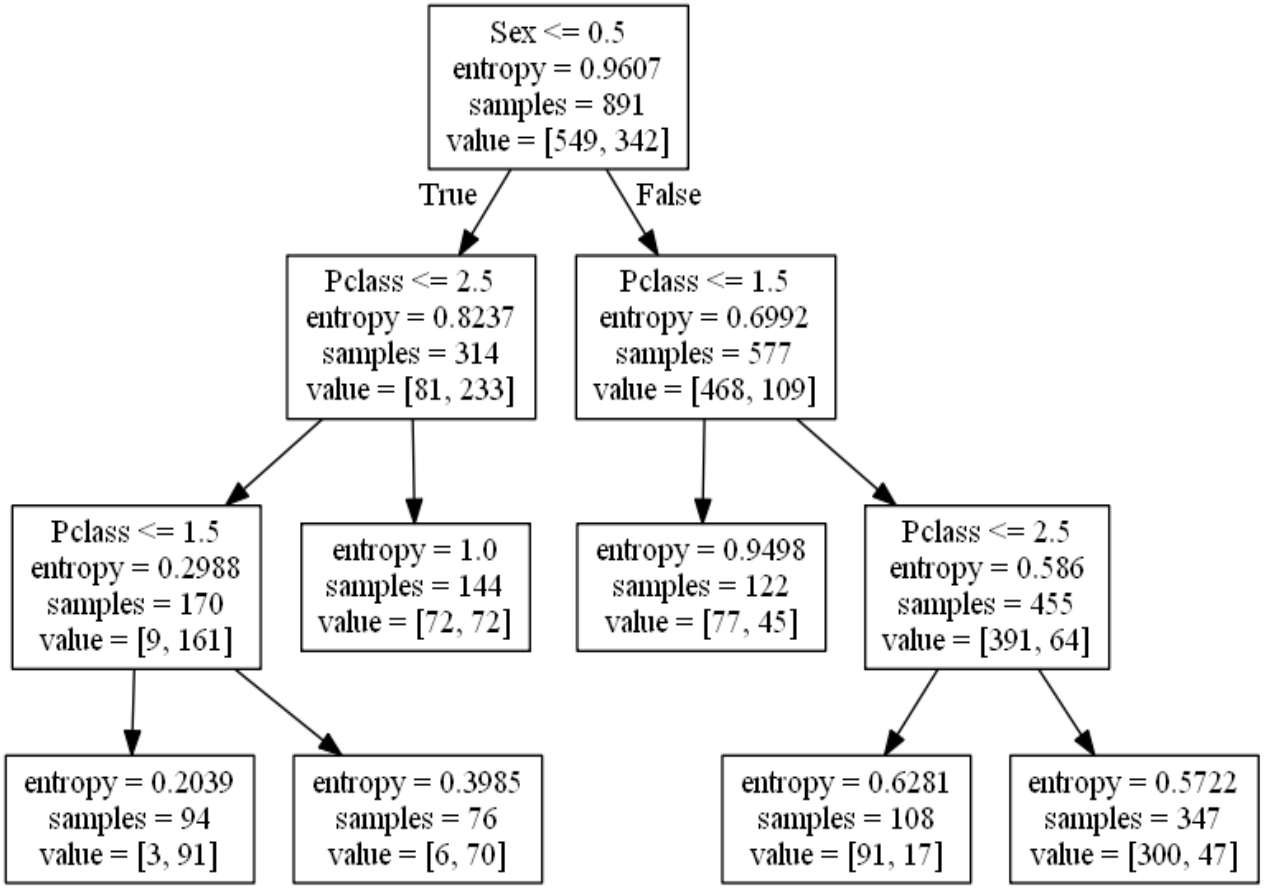
ID3算法实现

- 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
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

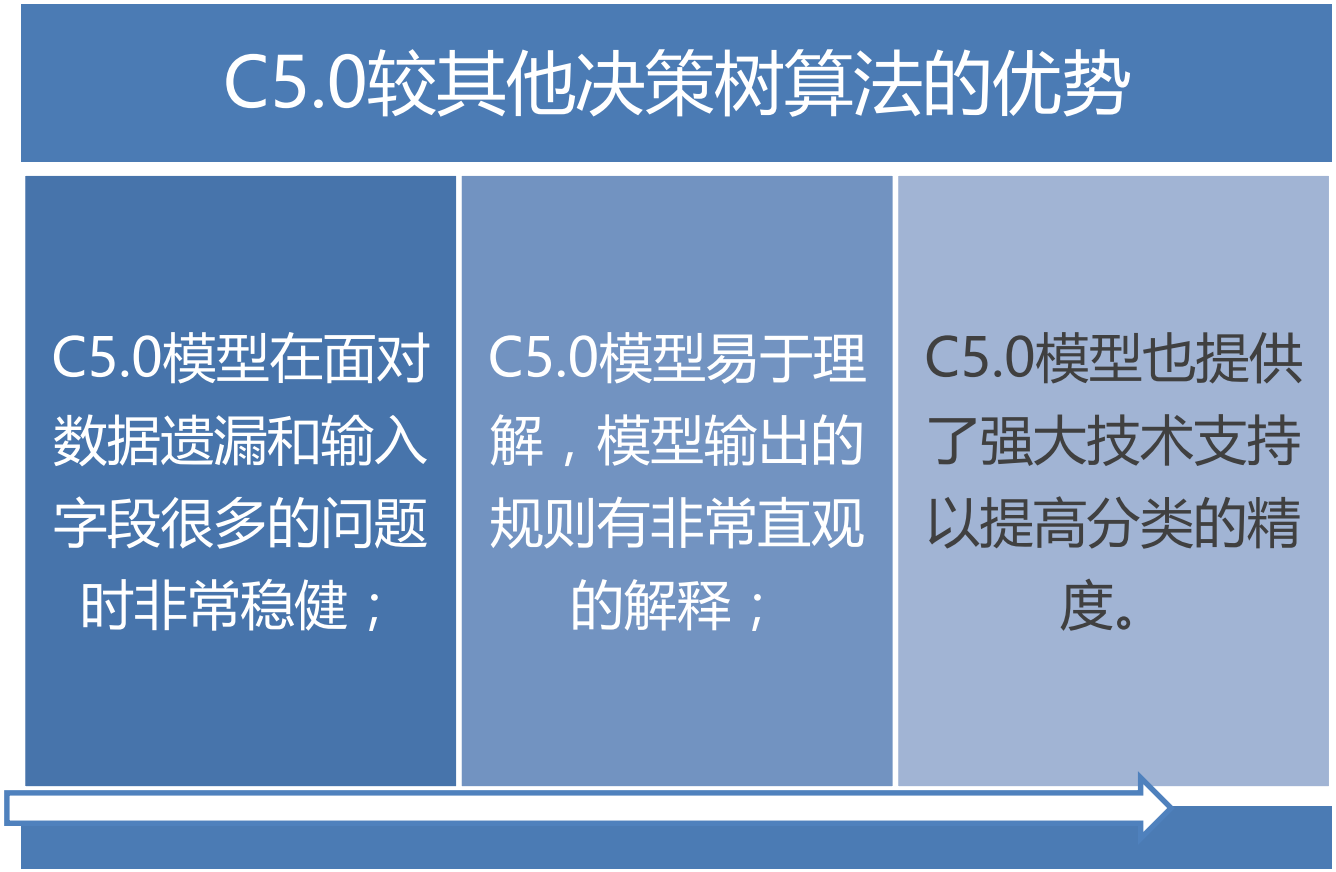
C4.5算法的缺点

- 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

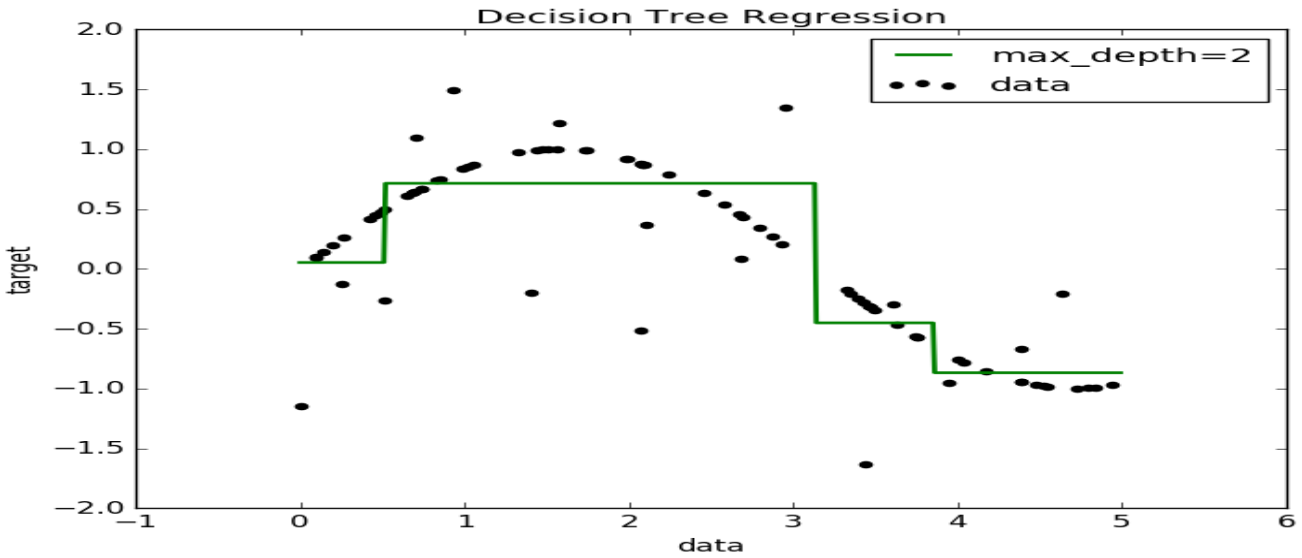
- C5.0算法是C4.5算法的修订版，适用于处理大数据集，采用Boosting方式提高模型准确率，又称为Boosting Trees，在软件上计算速度比较快，占用的内存资源较少。
- C5.0作为经典的决策树模型算法之一，可生成多分支的决策树，C5.0算法根据能够带来的最大信息增益的字段拆分样本。
- 第一次拆分确定的样本子集随后再次拆分，通常是根据另一个字段进行拆分，这一过程重复进行直到样本子集不能再被拆分为止。最后，重新检查最低层次的拆分节点，那些对模型值没有显著贡献的样本子集被剔除或者修剪。

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

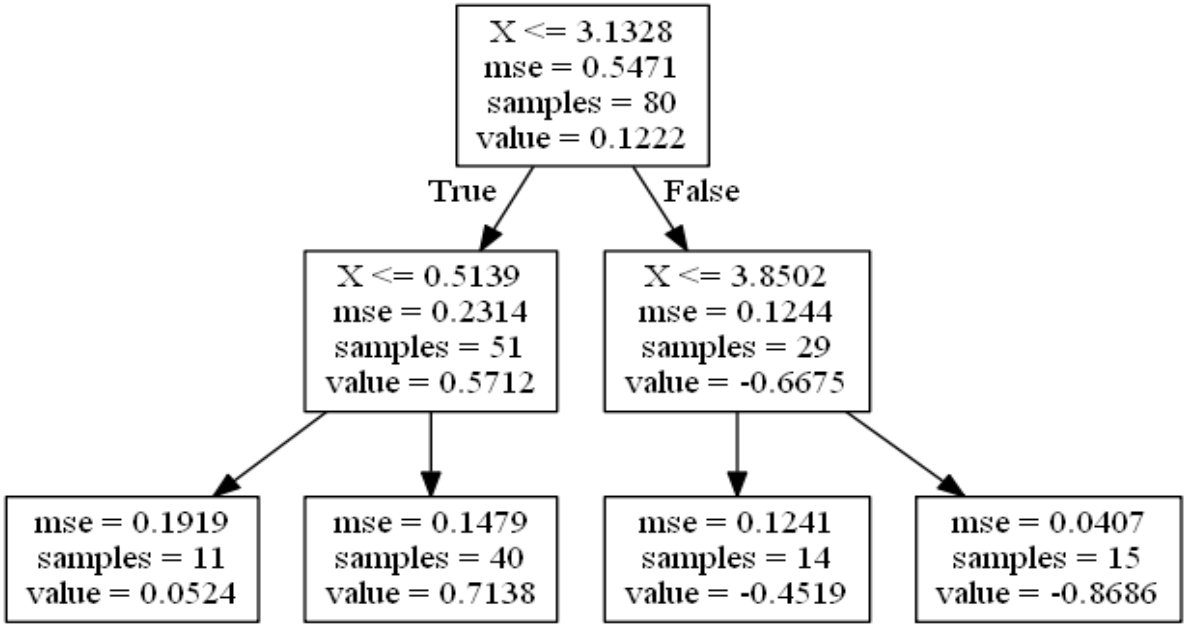
- 分类回归树（Classification And Regression Tree，CART）算法最早由Breiman等人提出，现已在统计领域和数据挖掘技术中普遍使用，Python中的scikit-learn模块的Tree子模块主要使用CART算法来实现决策树。



- 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。

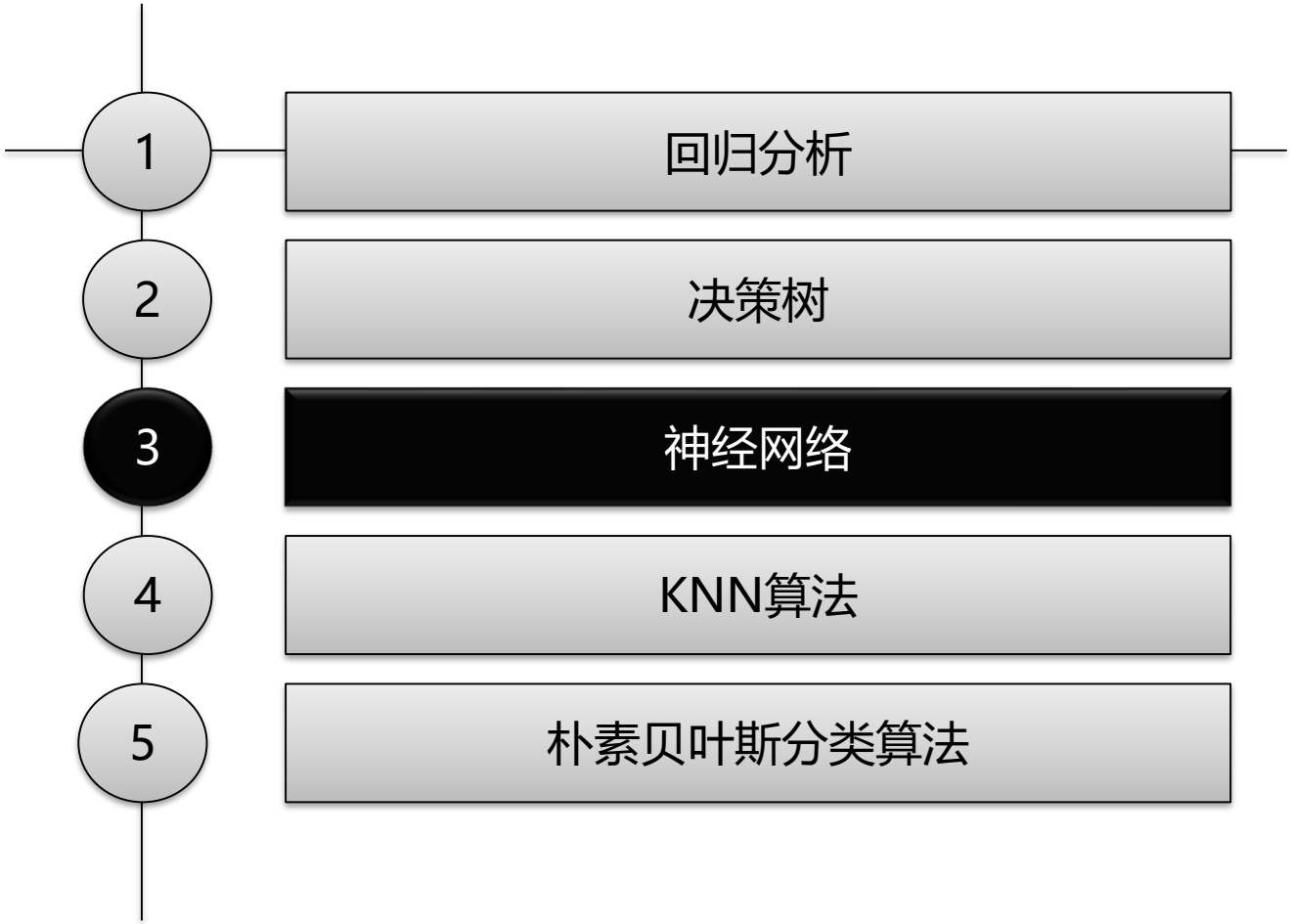
CART算法

- 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。

目录

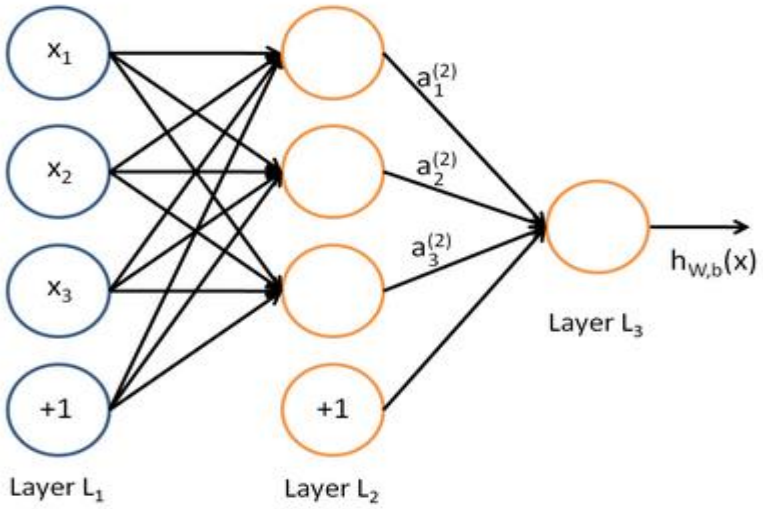


神经网络的概述

- 基于神经元的启发，科学家建立了一种新的运算模型，人工神经网络。神经网络是由大量的节点，或称为神经元，相互连接构成。
- 信息经过输入层进入神经网络，在节点中不断进行信息传输与运算，最后到达输出层，得到最终处理后的信息。
- 人工神经网络经过数据训练后，它就具有类似于人脑的能力，人工神经网络的研究使得“听歌识曲”，“图像识别”等应用得到高速发展。如果数据量足够用于训练和机器运算速度足够快，制造一个有人类智力的机器也是有可能的。

BP神经网络

- BP（Back Propagation）神经网络是一种处理分类和回归问题很有效的神经网络。本节我们重点介绍BP神经网络及其前向传播和反向传播的机制。



- 上图展现了一个3层的神经网络。我们使用圆圈来表示神经网络的节点，标上“+1”的节点被称为偏置节点。
- 第一层是网络的输入层，最后一层是输出层，其余的都称为隐藏层，如上图只有一个隐藏层，而且由3个隐藏单元组成(不包括偏置单元)。

向前传播

- 我们用 $a_i^{(l)}$ 表示第l 层第i 单元的激活值。当 l=1时，$a_i^{(1)} = x_i$。继续以上图的网络为例，给定参数集合（W,b）和激活函数f后，我们可以按照下面的公式计算第二层的激活值 $a^{(2)}$：

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$

- 我们用 $z_i^{(l)}$ 表示第加权和，如 $z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1)}x_j + b_i^{(1)}$，则 $a_i^{(l)} = f(z_i^l)$。我们可以使用矩阵乘法对上面的过程进行简化：

$$z^{(l+1)} = W^{(1)}a^{(l)} + b^{(1)}$$

$$a^{(l+1)} = f(z^{(l+1)})$$

- 具体地，在本例中：

$$z^{(2)} = W^{(1)}x + b^{(1)} \quad a^{(2)} = f(z^{(2)}) \quad z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} \quad h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

向前传播

- $h_{W,b}(x)$ 得出结果就是输出层的输出。
- 上面整个计算过程称为前向传播。设定参数矩阵和激活函数后，模型将信息一层层地从输入层往输出层传播，因此称为前向传播。
- 常用的激活函数有下面几种：

函数名称	函数公式	函数作用
sigmoid	$f(z)=\frac{1}{1+e^{-z}}$	S 型曲线，将 $(-\infty,+\infty)$ 映射到 $(0,1)$ 。
tanh	$f(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	双曲正切曲线，将 $(-\infty,+\infty)$ 映射到 $(-1,1)$ 。
softmax	$f(z_j)=\frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$	多分类中常用，能够将任意实数值映射为 $(0,1)$ 的概率值，并且 Z 的所有分量函数值和为 1。

反向传播

- 假设我们现有有一个数据集 $\{(x^{(1)},y^{(1)}),\cdots,(x^{(m)},y^{(m)})\}$它包含了m 个样本。我们首先设定代价函数，对于一个样例 $(x^{(i)},y^{(i)})$

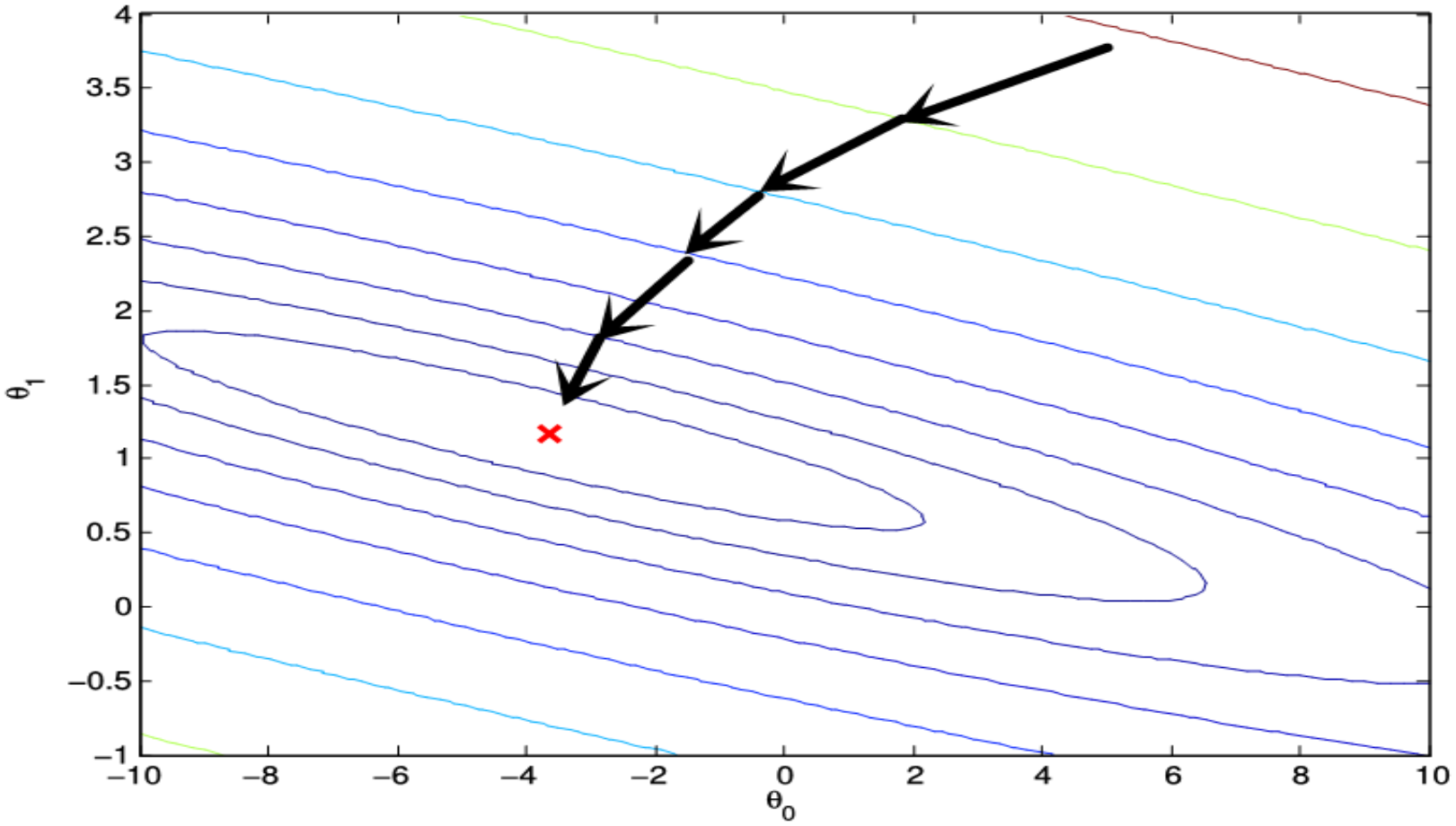
$$J(W,b;x^{(i)},y^{(i)})=\frac{1}{2}\left\|h_{W,b}(x^{(i)})-y^{(i)}\right\|^2$$

- 而对于整体代价函数我们定义为： $$J(W,b)=[\frac{1}{m}\sum_{i=1}^{m}J(W,b;x^{(i)},y^{(i)})]+\frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$
- 第一项表示残差平方和，而第二项是正则化项，目的是为了防止权重过大以致过度拟合。这个代价函数经常用于分类和回归问题。在二分类问题中，我们用 y=0和y=1 代表两种类型的标签。
- 有了代价函数，神经网络的任务就是使得“代价”尽量低。我们将使用梯度下降法对参数（W,b）进行优化，每一步迭代更新（W,b）使得代价函数的值不断减少。我们将使用W 和b 的偏导数对它们进行更新：

$$W_{ij}^{(l)}=W_{ij}^{(l)}-\alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b) \qquad b_i^{(l)}=b_i^{(l)}-\alpha\frac{\partial}{\partial b_i^{(l)}}J(W,b)$$

反向传播

- 梯度下降法示意图如下：



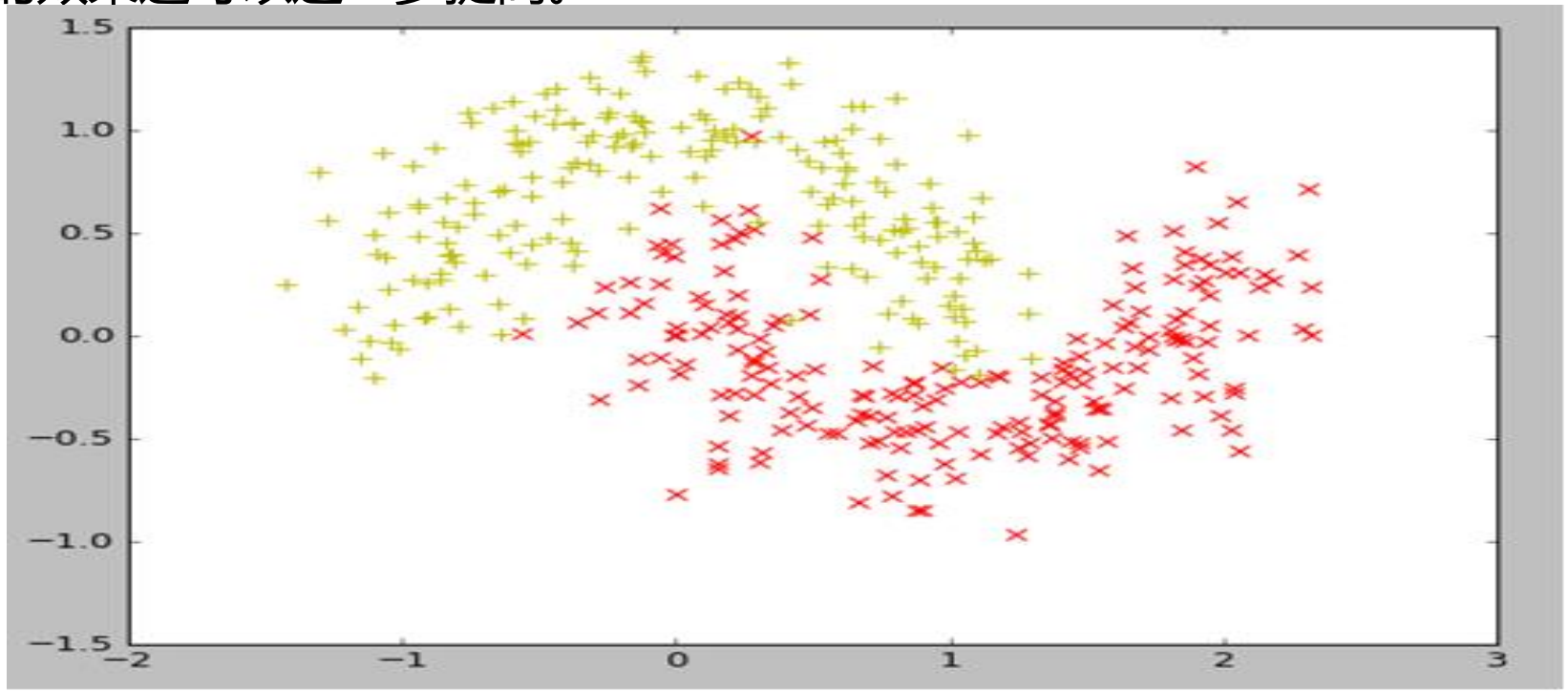
反向传播

- 给定一个样本 ，反向传播算法可以分为下面几个步骤：
- 利用前向传播算法，得到 L_2, L_3 直到输出层 L_{n_l} 的激活值。
- 计算输出层（第 n_l 层）的残差： $\delta^{(n_t)} = -(y - a^{(n_t)}) \bullet f'(z^{(n_t)})$
- 计算 $l = n_t - 1, n_t - 2, \cdots$ 的各层的残差： $\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \bullet f'(z^{(l)})$
- 计算最终需要的偏导数值： $\nabla_{W^{(t)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T$

$$\nabla_{b^{(t)}} J(W, b; x, y) = \delta^{(l+1)}$$

进行试验

- 我们尝试使用BP神经网络进行实验。数据集采用scikit-learn提供的make_moons数据集。产生的数据如下图 所示，’ +’ 表示女性病人，‘x’ 表示男性病人，x和y轴表示两个指标。
- 运行结果显示，BP神经网络的分类效果很不错，如果提高迭代次数分类的效果还可以进一步提高。



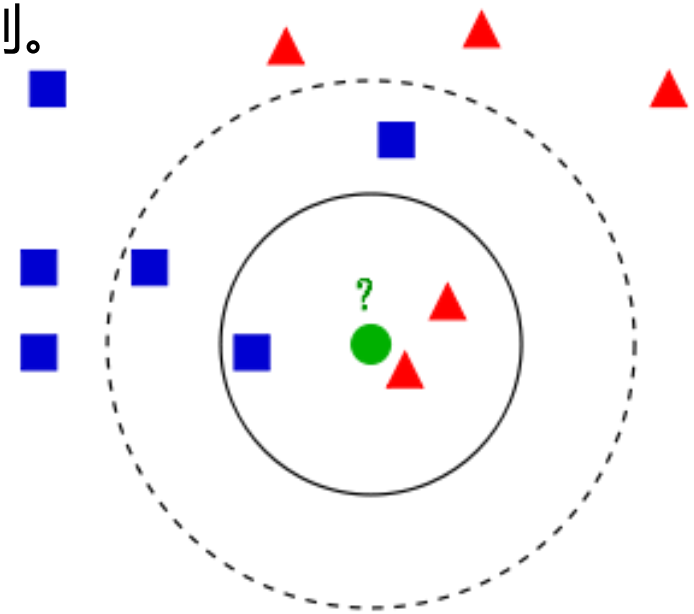
其他神经网络

- 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。



- 如上图 ，我们有两类数据：方块和三角形。它们分布在二维特征空间中。假设有一个新数据（用圆表示）需要预测其所属的类别，需要怎么来预测？

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；

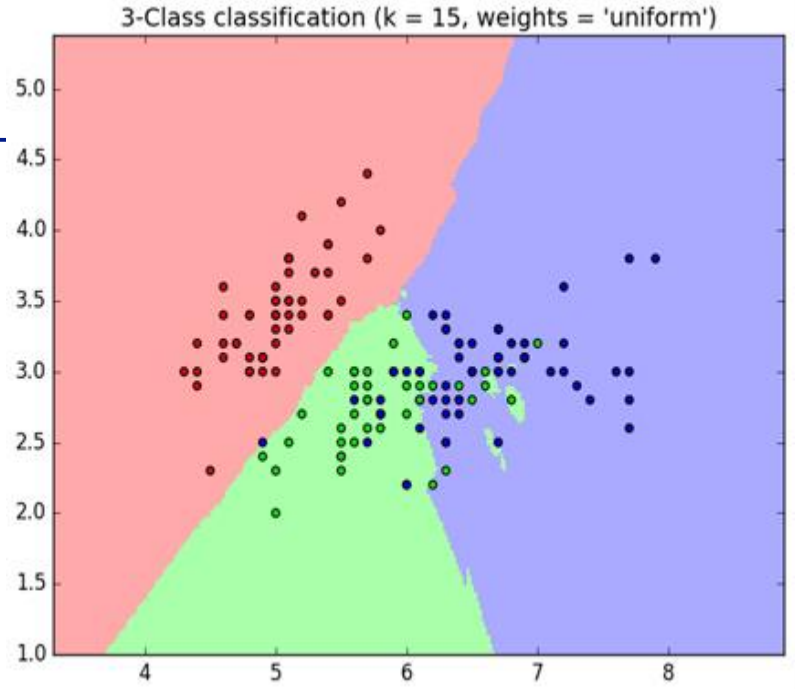
b
- 按照距离递增次序排序；

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。

算法实现




```
# 画出决策边界，用不同颜色表示
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
plt.figure()
# 绘制预测结果图
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
# 补充训练数据点
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("3-Class classification (k = 15, weights = 'uniform')")
plt.show()
```

目录

朴素贝叶斯

- 朴素贝叶斯算法是一个应用贝叶斯理论的一种有监督学习算法。它基于这样一个假设：特征之间是相互独立的。
- 给定一个分类标签 y和自由特征变量 $x_1,\cdots x_n$=1，表示样本具有特征 i，而 $x_i=0$. 表示样本不具有特征 i。如果我们想知道具有特征 1 到 n的向量是否属于分类标签 y_k，贝叶斯公式如下：

$$P(y_k \mid x_1,\cdots,x_n)=\frac{P(y_k)P(x_1,\cdots,x_n \mid y_k)}{P(x_1,\cdots,x_n)}$$

- 再由特征相互独立的假设：$P(x_1,\cdots,x_n \mid y_k)=\prod_{i=1}^{n}P(x_i \mid y_k)$
- 而由于 $P(x_1,\cdots,x_n)$ 已经给定，比较 $P(y_1 \mid x_1,\cdots,x_n)$ 和 $P(y_2 \mid x_1,\cdots,x_n)$，这与比较 $P(y_1)P(x_1,\cdots,x_n \mid y_1)$ 和 $P(y_2)P(x_1,\cdots,x_n \mid y_2)$ 等价。假设总共有 m种标签，我们只需计算$P(y_k)P(x_1,\cdots,x_n \mid y_k),k=1,2,\cdots,m$ ，取最大值作为预测的分类标签，即： $\hat{y}=\arg\max_k P(y)\prod_{i=1}^{n}P(x_i \mid y_k)$

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。

Python实现

- scikit-learn模块中有Naive Bayes子模块，包含了本节涉及到的所有贝叶斯算法。关键在于将分类器设置定为朴素贝叶斯分类器，接着调用分类器训练和进行分类，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!