《机器学习算法的数学解析与 Python 实现》

一、概述

- 拟合
  - 欠拟合
    - 算法模型的预测准确性不够
  - 过拟合
    - 泛化性不好
    - 训练集数据的分布情况可能与真实情况的分布情况略有不同，算法模型太注重细节，导致真正运用于真实环境时，预测精度下降

- 模型
  - 某种机器学习算法在设定参数后的产物

- 数据集
  - 训练集
  - 测试集

- 数据
  - 一条数据为一个样本（Sample）
    - 形式类似 `一维数组`
    - 通常包含多个特征（Feature）
    - 用于 `分类` 问题的数据集，还会包含 `类别` （Class Label）信息
    - `回归` 问题的数据集，则包含一个 `连续型` 的数值

- 特征
  - 某个对象的几个记录维度
  - 数据形式类似一维数组，特征就是数组的值

- 向量
  - 线性代数
  - ML 模型算法的运算均基于线性代数法则
  - 一条样本数据就是以一个向量的形式输入模型的

一条监督学习数据的向量形式：

```
[特征X1值，特征X2值，...，Y1值]
```

- 矩阵
  - 线性代数
  - 可以看成由向量组成的数组，形式上接近二维数组
  - 数据集，通常以矩阵的形式输入模型

```
[[特征X1值，特征X2值，...，Y1值],
[特征X1值，特征X2值，...，Y2值],
...
[特征X1值，特征X2值，...，Yn值]]
```

常用函数

- 假设函数（Hypothesis Function）
  - `H(x)`
    - `x` 理解成矩阵形式的数据
    - 返回一个结果，就是预测结果

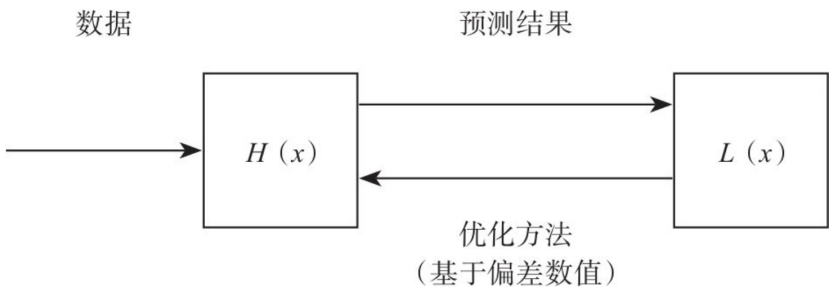
- 损失函数（Loss Function）
  - 不断逼近，衡量工具来度量当前距离目标
  - 又叫 `目标函数`

    - `L(x)`
        - `x` 是假设函数的预测结果
    - 返回值越大，结果偏差越大

- 成本函数（Cost Function）
    - `J(x)`

损失函数是单次衡量标准，成本函数相当于损失函数的总和

基本模式



- 优化方法

`新参数值 = 旧参数值 - 损失值`

- 牛顿法
- 拟牛顿法
- 共轭梯度法等

目的：通过调整假设函数的参数，令损失函数的损失值达到最小。

- 梯度下降
    - 微积分
    - （Gradient Descent）
    - 常用的一种优化方法
    - 某个函数在某点的**梯度**指向该函数取得最大值的方向

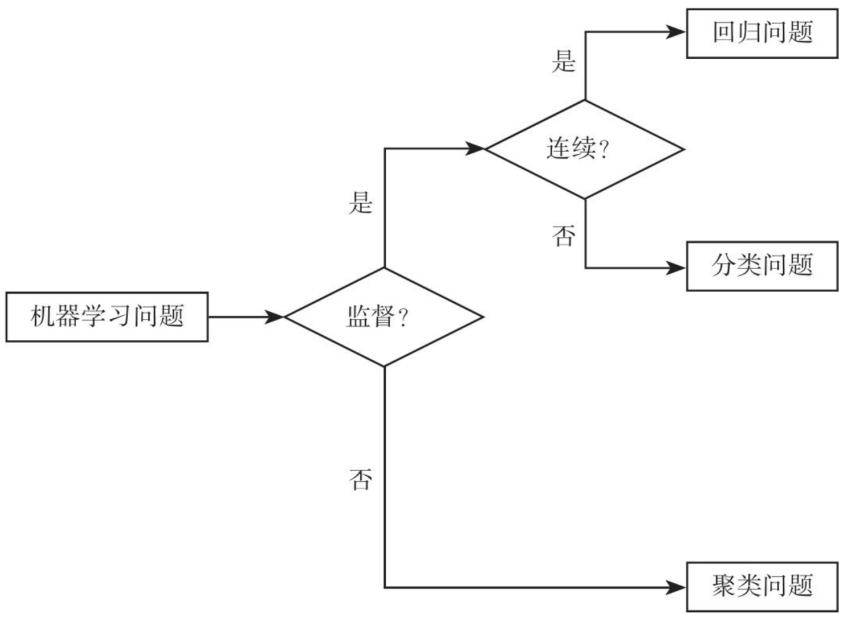
- 反方向就是取得最小值的方向

> 只要对损失函数采用梯度下降法，让假设函数朝着梯度的负方向更新权值，就能达到令损失值最小化的效果

如果样本数量庞大，完成一次完整的梯度下降需要耗费很长时间。

在实际中会根据情况调整每次参与损失值计算的样本数量。

- 每次迭代都使用全部样本的，称为 `批量梯度下降` （Batch Gradient Descent）
- 每次迭代只使用一个小样本的，称为 `随机梯度下降` （Stochastic Gradient Descent）
  - 计算样本小，随机梯度下降的迭代速度更快
  - 但更容易陷入局部最优，而不能到达全局最优点



- 是否有监督?
  - 无监督学习（Unsupervised Learning）
  - 有监督学习（Supervisored Learning）
    - 回归问题
      - 结果连续
    - 分类问题
      - 结果分散

常用的 ML 算法：

1. 线性回归算法
    a. 用线性方法解决回归问题
2. Logistic 回归分类算法
    a. 线性回归算法的孪生兄弟
    b. 核心思想仍然是线性方法，套了 Logistic 函数
        i. 具有解决分类问题的能力
3. KNN 分类算法
    a. 通过“找最近邻”的思想解决分类问题
4. 朴素贝叶斯分类算法
    a. 根据概率，解决分类问题
5. 决策树分类算法
6. 支持向量机分类算法
    a. Logistic 回归分类算法是最基本的线性分类算法
    b. 支持向量机则是线性分类算法的最高形式
    c. 最“数学”的一种算法
    d. 将线性不可分的数据点映射成线性可分，再用最简单的线性方法来解决问题
7. K-means 聚类算法
    a. 无监督学习，不需要依赖样本标记
    b. 无监督学习的代表
8. 神经网络分类算法
    a. 深度学习

算法的性能衡量指标

- TP true positive
- TN true negative
- FP false positive
- FN false negative

正类，负类，事实相符

三个指标：

1. 准确率（Accuracy）

$$\frac{TP+TN}{TP+FN+FP+TN}$$

为 true 的情况占比

2. 精确率（Precision）

又叫查准率

$$\frac{TP}{TP+FP}$$

为 true 的情况，在 positive 正类的占比

3. 召回率（Recall）

又叫查全率

$$\frac{TP}{TP+FN}$$

全部事实是正类

- 数据决定了算法的能力上限
- 特征工程
  - 选取合适的特征

二、环境

- numpy
  - 科学计算
- scikit-learn
  - 机器学习
  - `fit`
  - `predict`
- pandas
  - 数据处理
  - 核心数据类型

    - `Series`
        - 一维，一个统计功能增强版的 List 类型
    - `DataFrame`
        - 多维，由多个 Series 组成

三、线性回归算法

两条主线：

- 问题
- 模型

1. 线性回归

Linear Regression

分为两块：

- 线性
    - 线性模型
- 回归
    - 回归问题

> 用线性模型解决回归问题

也可以解决分类问题

ML 是问题导向的，有了问题，才提出解决方法

线性回归：

- 线性方程
- 偏差度量

回归问题：各数据点沿着一条主轴来回波动

回归问题的两个代表性特征：

- 记录历史值
- 预测未来值

回归问题和分类问题区别：

- 根据预测值类型的不同，分为两种：
  - 一种是连续的
    - 结果是连续的，就是预测问题
      - 线性连续和非线性先序
      - 通常用 int/float 类型
  - 一种是离散的
    - 离散型数值，缺乏中间过渡值
    - 通常用 bool 类型

- 回归问题：
  - 一类预测连续值的问题
    - 数学模型，回归模型
      - 线性回归，是回归的一种

线性模型

假设函数是一类函数，起预测作用

线性方程就是线性回归模型的假设函数

- 对数函数
- 指数函数

$y = kx + b$

- k 斜率

    - 旋转
- b 截距
    - 平移

线性模型拟合能够调节的参数，主要就这两个

在 ML 中，

- 斜率 k，使用字母 `w` 表示，权值（Weight）

通过调整权值来达到目的的过程叫做 `权值调整` 或者 `权值更新`

代数角度解释

以三个输入维度 A、B、C来预测 P 为例，线性方程：

$$F = W_1 * A + W_2 * B + W_3 * C$$

调整相关维度的权值

特征：数据集点沿线性分布

已知条件整理成数据集，矩阵

2. 算法原理

基本思路

拟合

在错误中学习，首先知道偏离了多少，然后向减少偏差的方向调整权值。

两个步骤：

- 偏差度量
    - 如何度量

    - `损失函数`
- 权值调整
    - 加还是减，数值多少
    - `优化方法`

1. 假设函数的数学表达式

$$\hat{y} = \omega^T x_i + b$$

所有假设函数都习惯用 $\hat{y}$ 代表预测结果

ω^T 线性代数符号，`转置`

$\omega^T x_i$ 求 ω 与 x_i 的乘积，这两个都不是标量，而分别代表着一个 n 维向量

ω 代表着 $\omega = [\omega_1, \ldots, \omega_n]$

向量没有乘积，在线性代数中，求两个向量的内积（Dot Product），内积得到的结果是一个标量

内积唯一要求，维度相同，按位相乘然后再求和

$[1,3,5]^T[2,4,6] = 1 \times 2 + 3 \times 4 + 5 \times 6 = 2 + 12 + 30 = 44$

$y = kx + b$

$\hat{y} = \omega^T x_i + b$

- 直线方程
    - 两个标量乘积
    - 平面空间
- 线性方程
    - 两个n维向量的内积
    - 多维空间

线性回归模型，用线性方程进行预测。

线性回归的假设函数就是线性函数，

$$H(x) = \omega^T x_i + b$$

预测函数输入数据，等于给 x 赋值（多维矩阵），预测函数经过计算后能够返回一个结果，就是预测值。

2. 损失函数的数学表达式解析

度量偏差，减少偏差

- 预测值 $\hat{y}$
- 真实值 y

线性回归实际是用直线进行拟合，出现偏差，即线性方程作出来的直线和实际的点存在 距离

使用几何意义上的“距离”来进行度量

因此，线性回归的损失函数选择使用 L2 范数来度量偏差

$$L(x) = \left\| \hat{y} - y \right\|_2^2$$

范数正则化，简称范数，或正则化

范数通常与数字一起，表示 L_n 范数

$$\|\|_n$$

在欧几里得空间，最常解除的几何空间中， L_2 范数表示的就是欧几里得距离（Euclidean Distance，简称欧式距离），两点之间的连线长度

L_2 范数包含根号，为了方便计算，损失函数直接在外面加了个平方，与根号抵消。

操作会对计算结果进行同步放大

3. 优化方法的数学表达

使优化方法将偏差减到最小

通常使用梯度下降等

两个要素：

- 损失函数
- 最小化

$$\underset{w,b}{min} \left\| \hat{y} - y \right\|_2^2$$

调节方法

$$\omega_{新} = \omega_{旧} - 学习率 \times 损失值$$

通过梯度下降等优化方法求得最小值时，损失值通过损失函数对 ω 求偏导计算求得，这个偏导也称为梯度，通过损失值来调节 ω，不断缩小损失值直到最小，正是梯度下降的得名来由。

`学习率`

是一个由外部输入的参数，被称为“`超参数`”

理解为 ω 通过这一次错误学到多少，想要 ω 多调整一点，就把学习率调高一点。

也不是越高越好，过高的学习率可能导致调整幅度过大，错过了`最佳收敛点`，导致无法求得真正的最小值。

4. 范数

范数种类多，但ML中常用的就是 L_1 和 L_2 范数

L_1 范数（Lasso Regression）

向量中每个元素绝对值的和

计算分2步：

1. 逐个求得元素的绝对值

2. 然后相加求和即可

L_1 范数正则化定义的数学表达式

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

L_2 范数（Ridge Regression）

出现频率更高

表示向量中每个元素的平方和的平方根

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

L_0 范数指向量中非0元素的个数

线性回归算法的具体步骤，3步：

1. 为假设函数设定参数 ω，通过假设函数画出一条直线，即根据输入的点通过线性计算得到预测值
2. 将预测值带入损失函数，计算出一个损失值
3. 通过得到的损失值，利用梯度下降等凸优化方法，不断调整假设函数的参数 ω，使得损失值最小。这个不断调整参数 ω 使得损失值最小化的过程就是线性回归的学习过程，通常称为训练模型。

3. 在 Python 中使用线性回归算法

Scikit-Learn 根据模型细分算法族：

- `.linear_model` 线性模型
- `.neighbors` 最近邻，KNN
- `.naive_bayes` 朴素贝叶斯
- `.tree` 决策树
- `.svm` 支持向量机
- `.neural_network` 神经网络

```
from sklearn import linear_model

# 训练线性回归模型
model = linear_model.LinearRegression()
model.fit(x,y)

# 进行预测
model.predict(x_)
```

实际

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-3, 3, 30)
y = 2 * x + 1

plt.scatter(x, y)
plt.show()
```

线性回归算法的 fit 方法需要传入的 x 和 y 是两组矩阵

```
x: [[样本1], [样本2], [样本3], ... [样本n]]
y: [[样本1标记值], [样本2标记值], [样本3标记值], ... [样本n标记值]]
```

转换

```
x=[[i] for i in x]
y=[[i] for i in y]
```

为了检验训练的结果，还需要提供一组测试用的 x_

```
x_=[[1], [2]]
```

使用 predict 进行预测

```
array([3.], [5.])

# model.coef_ 和 model.intercept_ 查看 w 和 b
```

4. 使用场景

适用于线性分布的场景

线性回归能预测成功，是因为现实分布确实是依从线性的。

- 优点
  - 简单，可解释性强
- 缺点
  - 线性模型不能表达复杂的模式
- 应用领域
  - 金融、气象

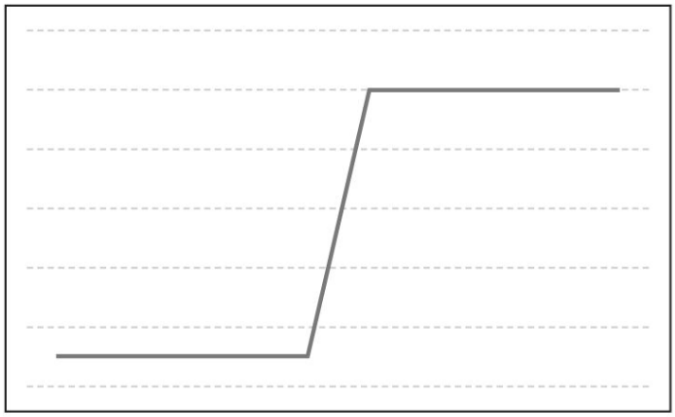
经典问题：波士顿房价预测问题

四、Logistic 回归分类算法

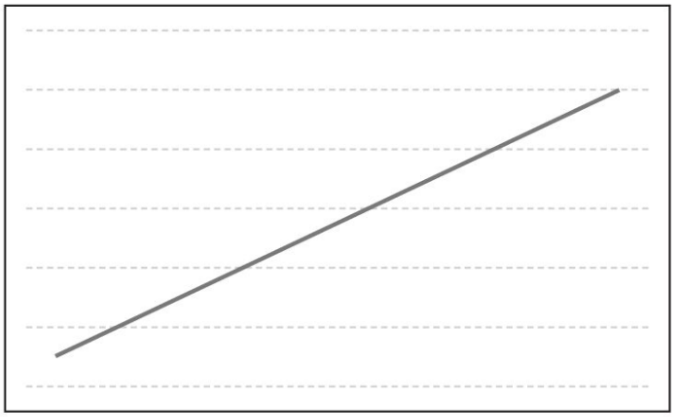
回归问题，分类问题

线性模型+Logistic 函数，解决分类问题

离散数据总带着“阶跃”这么一种特征



a）离散数据图像



b）线性数据图像

如果待分类别只有两个，通常称之为二元分类（Binary Classification）问题，较多使用 `Logistic 函数` 来解决。

超过两个，则称为多分类（Multi-class Classification）问题，较多使用 `Softmax 函数` 来解决

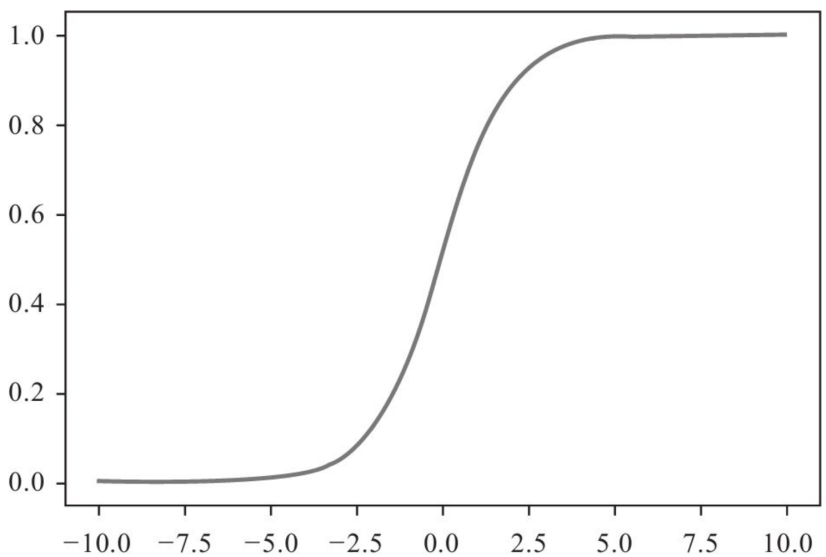
- 正类 Positive
  - 正样本
- 负类 Negative
  - 负样本

将多分类问题，以二叉树形式，转换为多个二元分类问题

Logistic 函数由统计学家皮埃尔·弗朗索瓦·韦吕勒发明于19世纪，有很多名字。

在神经网络算法中，称为 Sigmoid 函数

也称为 Logistic 曲线



阶跃函数（Step Function，又称 Heaviside Function）

阶跃函数不可导

两条直线+一个点，在点上的函数值为一个确数

奇异函数

阶跃函数的图像是不连续的，不连续的函数同样不可导。

在 ML 中，可导性非常重要，否则就无法搭配使用梯度下降等优化算法，使得偏差最小

2. 可导的阶跃函数

Logistic 函数，即是可导函数，又是一种阶跃函数。

缩放，0坐标

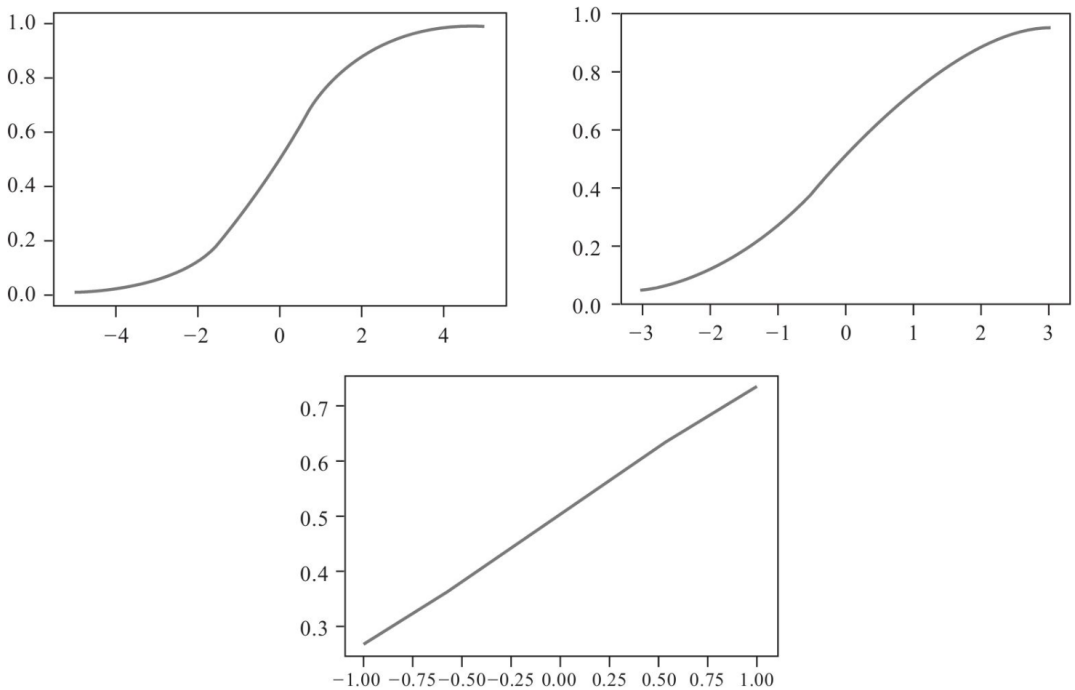


图4-6　三种条件下的Logistic函数图像

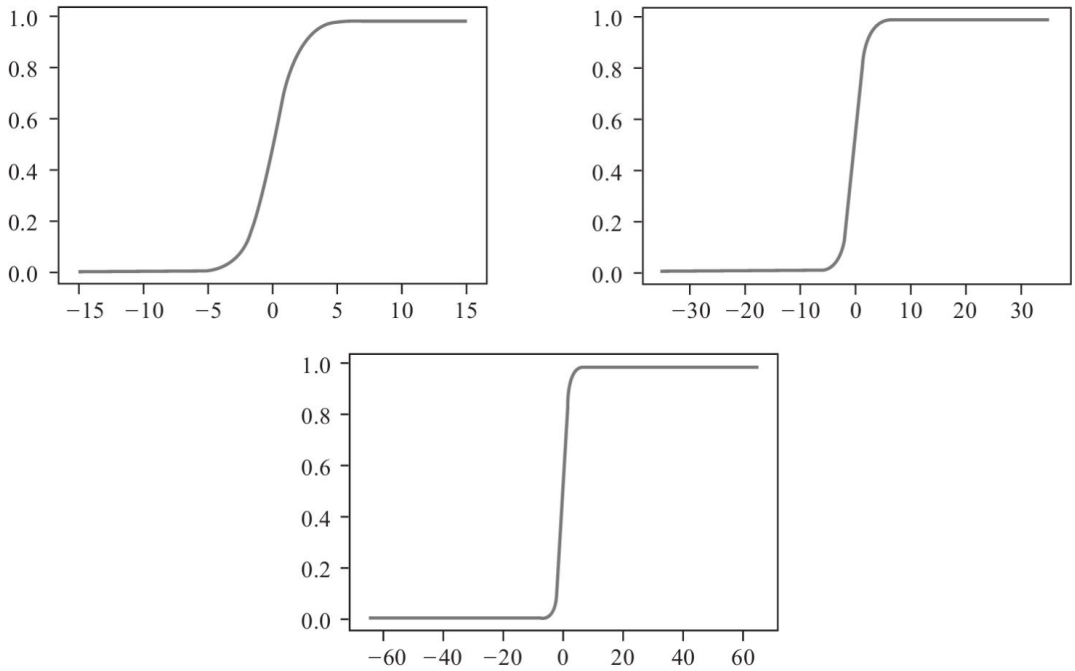


图4–7　三种大跨度值下的Logistic函数图像

把 Logistic 函数作为连接线性连续值和阶跃离散值的桥梁

- X 轴的值越是小于 0，Y 轴的值则越接近于 0
- X 轴的值越是大于 0，Y 轴的值则越接近于 1

有了 Logistic 函数进行映射，线性模型不再需要输出某个特定的值，而只要满足“让输出尽可能地接近 0 或者 1”即可

对数几率回归

Logistic Regression（LR 算法）

2. 算法原理

类别：

- 数字形式，1 或 0
- 向量形式
  - 当前深度学习在分类问题上使用最多的一种形式
  - 特别是在多分类问题上多采用这种形式
  - 用向量中元素顺序代表类别

    - 如有 A、B、C 三类，可以用 [x1，x2，x3] 这种的向量元素依次代表
    - 预测结果为哪一类，就把向量中的对应元素置1，其余为0
    - B = [0,1,0]
- 概率值形式
    - 部分算法不能给出二值，而是每个类的可能概率
    - A、B、C 对应 [0.8435, 0.032, 0.000419]

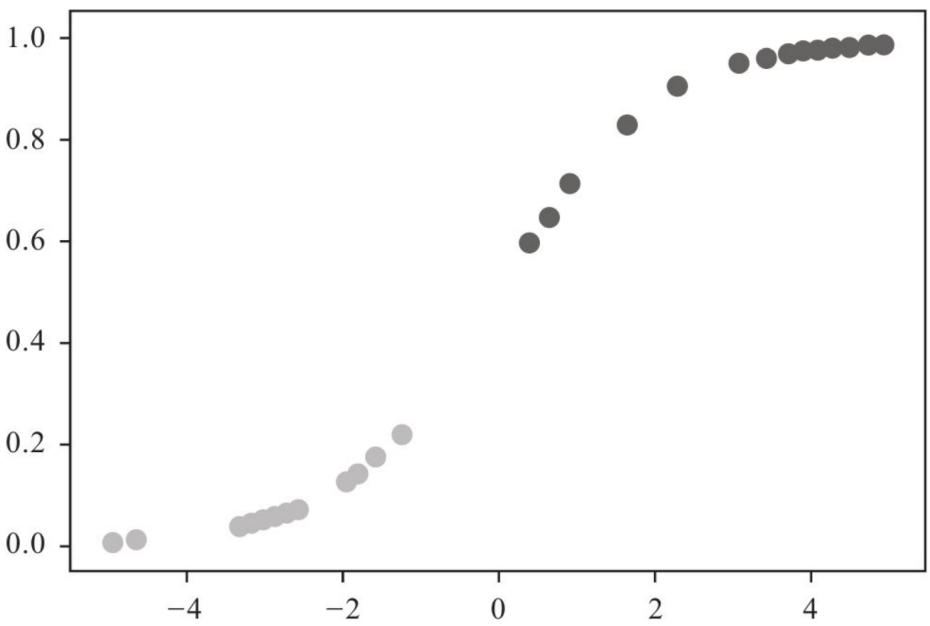
```
if (线性模型输出的连续值 > 0):
    return 1
else:
    return -1
```

要求预测值距离 0 点越远越好

Sgn 函数

$$sng(x) = \begin{cases} 1, \ x > 0 \\ 0, x = 0 \\ -1, \ x < 0 \end{cases}$$

Sgn 函数存在不可导的问题，所以使用 Logistic 函数



数学解析

1. 数学表达式

$$Logistic(z) = \frac{1}{1+e^{-z}}$$

其中以 e 为底的 指数函数 $e^x = exp(x)$

$$Logistic(z) = \frac{1}{1+exp(-z)}$$

Logistic 回归的假设函数，就是套上 Logistic 函数的线性方程，也就是把线性方程表达式带入上式的 z

$$H(x) = \frac{1}{1+e^{-(\omega^T x_i+b)}}$$

函数产生的预测值沿 S 形分布，能够产生离散的结果

2. 损失函数

$$L(x) = -ylogH(x) - (1-y)log(1-H(x))$$

分类问的预测值是离散的，其损失函数与回归函数不同

假设函数的值域，(0,1)，输出区间符号概率的要求

正类的概率

如果把预测结果看作概率，则损失函数：

$$L(x) = -H(x_i)_i^y(1-H(x_i))^{1-y_i}$$

是根据概率设计出来，由两部分相乘，但由于y的值只会为0或1，所以每次只会有一个部分能够输出值。

当 y=1 时，1-y=0，第二部分值为1，反之同理

- 当 y=1，预测正确，预测值无限接近 1，损失值为 -1
- 如果预测错误， $H(x_i)^{y_i}$ 的值为0，损失值为0
- 预测错误的损失值 > 预测正确的损失值

似然函数 $P(Y|X;w)$ ，预测值和实际结果越相似，似然函数的值越大。

我们希望的是预测值与实际结果相差越大，函数的值越大，而只要对似然函数 `取负` ，即可

第一版的损失函数虽然能够表达预测值和实际值之间的偏差，但存在一个致命的问题：

- 不是一个凸函数

将导致无法使用梯度下降等优化方法使得损失值最小

解决方法：对数函数，取 log

对数函数：

- 单调函数
- 底数 > 1 时
  - 单调递增
- 0 < 底数 <1 时
  - 单调递减

ML 中大量使用了 log，但底数均大于 1，单调递增

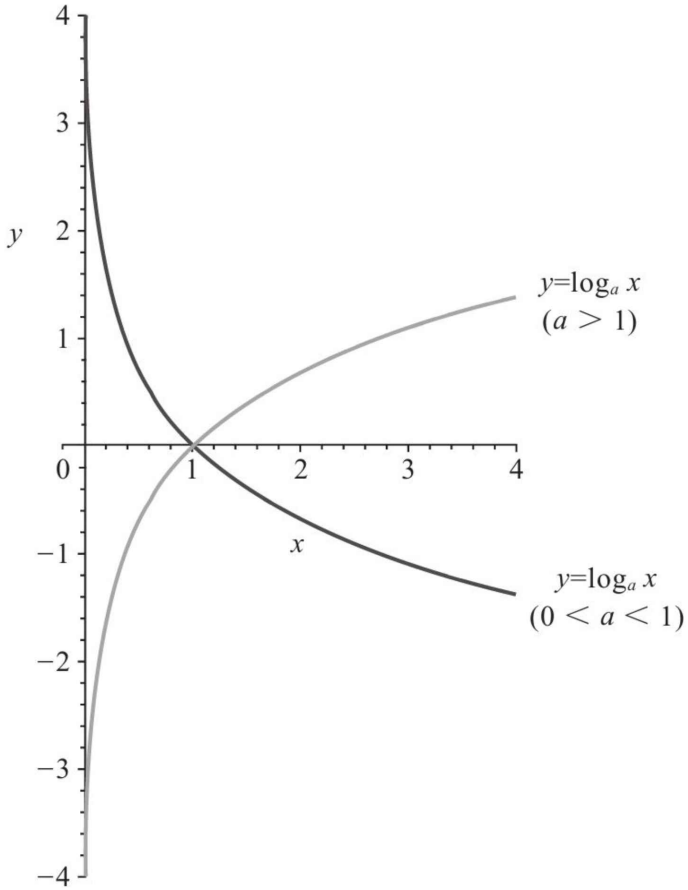


图4–9　两种底数条件下的对数函数图像

取 log 后涉及对数运算

- 乘法
- 除法
- 指数

取对数后，乘法变加法

$$2^2 \times 2^3 = 2^{2+3}$$

$$\log_a(MN) = \log_a M + \log_a N$$

具体步骤：

输出是一个离散值

三步：

1. 为假设函数设定参数 ω，通过假设函数计算出一个预测值
2. 将预测值带入损失函数，计算出一个损失值
3. 通过得到的损失值，利用梯度下降等优化方法调整参数 ω
    a. 不断重复，使得损失值最小

3. 在 Python 中使用 Logistic 回归算法

在 Scikit-Learn 中，线性模型算法族都在 linear_model 类库下

1. LinearRegression 类

对应线性回归算法，也称为普通最小二乘法（Ordinary Least Square， OLS），用于预测回归问题

损失函数：

$$L(x) = \min_{w} \left\| Xw - y \right\|_2^2$$

调用 fit 来拟合，系数存储在 `coef_` 中

2. Ridge 类

对应 Ridge 回归算法，又称为 `岭回归` ，用于预测回归问题，在线性回归的基础上添加了 L_2 正则项，使得权重 weight 的分布更为平均

损失函数：

$$L(x) = \min_{w} \left\| Xw - y \right\|_2^2 + a \left\| w \right\|_2^2$$

左侧与线性回归算法的损失函数一致，额外添加了右侧的L2 正则表达式，其中 a 是一个常数，根据经验设置

3. Lasso 类

对应 Lasso 回归算法，添加L1正则项

$$L(x) = \min_{w} \frac{1}{2n} \left\| Xw - y \right\|_2^2 + a \left\| w \right\|_1$$

左侧相比于线性回归，多了一个 $\frac{1}{2n}$，其中 n 是样本数量，在优化过程的运算中不会发生变化，是一个常量，并不会对权重 ω 的调整产生影响

右侧使用 L1 正则项

4. LogisticRegression 类

```
from sklearn.linear_model import LogisticRegression
# 知名的鸢尾花分类数据集，是一个分类问题的数据集
from sklearn.datasets import load_iris

# 载入数据集
X, y = load_iris(return_X_y=True)
# 训练模型
clf = LogisticRegression().fit(X, y)

# 使用模型进行分类预测
clf.predict(X)
```

模型自带默认的性能评估器

```
clf.score(X,y)
```

4. 使用场景

在多特征、多类别的数据环境下，Logistic 回归容易出现过拟合的情况，表现不如二元分类领域。

Logistic 回归还可以作为分类模型的 `基线模型` 。

- 优点
  - 简单，可解释性强，计算代价较低
- 缺点
  - 分类的效果有时不好，容易 `欠拟合`
- 应用
  - 二分类领域，或作为其他算法的部件
    - 如作为神经网络算法的激活函数

案例

- 研究点击率（Click Through Rate，CTR）的变化规律
- Google 提出了 LR-FTRL 算法

五、KNN 分类算法

1. 多数表决进行分类

梯度下降等优化方法减少偏差值时，可能出现陷入“局部最优解”的问题

解决方案：

先获得某种 `全局性的统计值` ，然后在全局统计值的基础上完成分类等预测工作，避免陷入局部最优解

KNN 算法没有通过优化方法来不断减少偏差的显式学习过程，这意味着用不上损失函数和优化方法这套机制

KNN “多数表决”

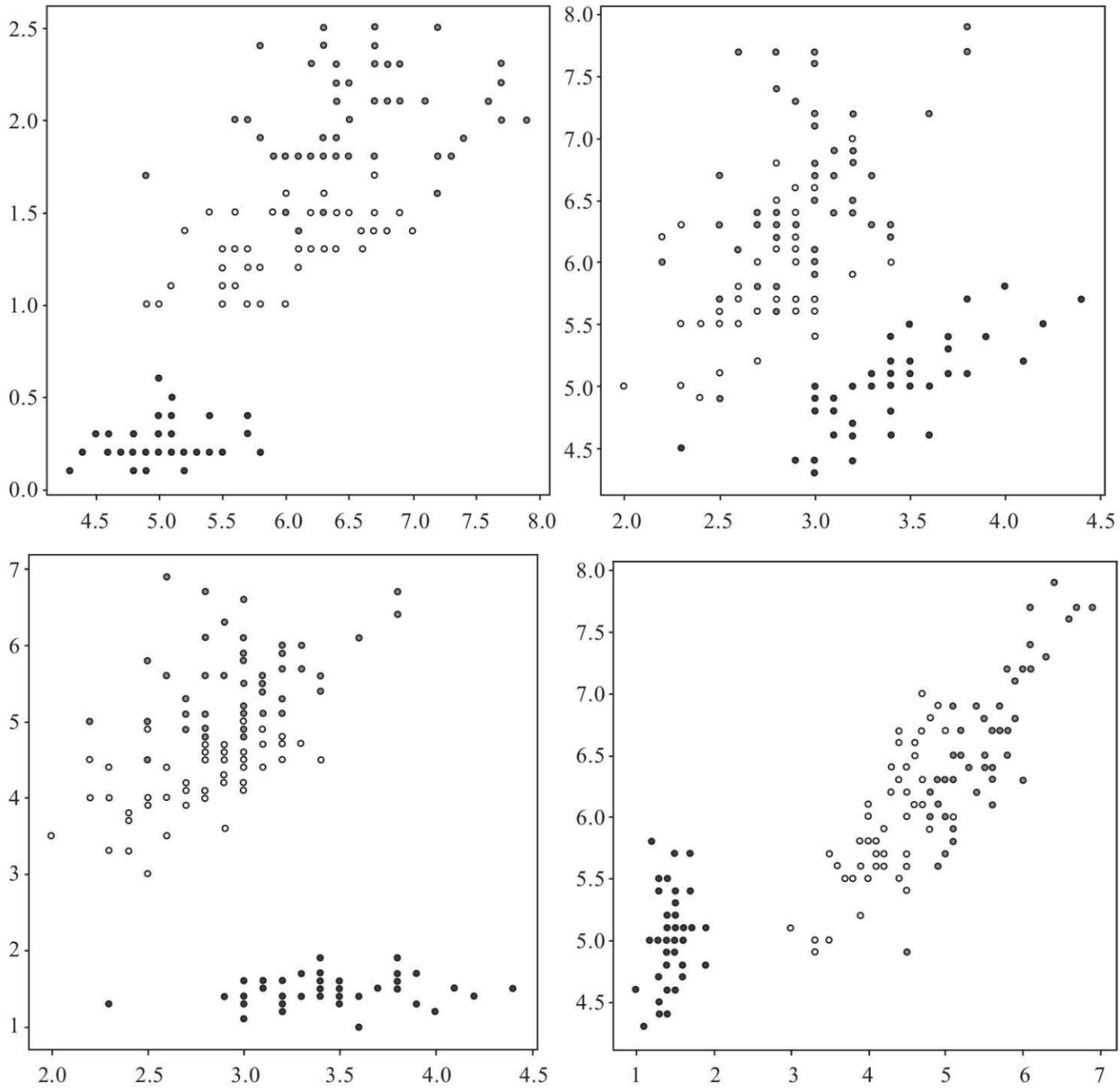
“表决权”界定问题，需要依赖“距离”来进行度量

多数表决，和距离是 KNN 中重要的两个概念

新样品和哪一类的样本共同点最多，最为相像。就分成哪一类

“同类相吸”是 KNN 分类算法的指导思想，从而模型可以脱离对偏差的依赖，而同样起到分类的效果。

假设，样本有4特征，对应4个维度的数据，每次取2个维度，作为 X/Y 轴坐标，得到 16 张图像。



如何选取维度，是 KNN 算法乃至机器学习都需要重点关注的问题，选取合适的维度可以事半功倍。

“先有结论再找证据”

维度超过3个之后，存在无法可视化的问题

> 根据各个维度的值，看看临近的点都是什么类，按多数表决原则，哪些类占大多数，这个新样本就属于哪一类。

表决权问题

由距离决定，根据样本各维度的值，可以作出一个个数据点。只需要度量点与点之间的距离

以该点为圆心，找到临近的点有哪些，只有在范围内的点，才拥有表决权。

全体表决，导致“滚雪球”

- 衡量距离

KNN 的具体含义

（K-NearestNeighbor） K个最近邻，最近邻算法
K 值是多少，就代表使用了多少个最近邻

类似命名：K-means 算法

2. 算法原理

KNN 关键在于最近邻，以待分类样本点为中心，距离最近的K个点，这 K 个点中什么类别占比最多，待分类样本就属于什么类别。

- 怎么确定 K
  - 根据交叉验证等实验方法，结合经验进行设置
  - 一般情况下，K在 3~10 之间
- 怎么确定 NN
  - 用什么方法度量“最近”

L2 范式，欧几里得距离

`闵可夫斯基距离（Minkowski Distance）`

数学解析

闵可夫斯基距离的数学表达式：

$$d_P(x,y) = \left(\sum_{i=1}^{n} \left| x_i - y_i \right|^P \right)^{1/P}$$

闵可夫斯基距离是一组距离的定义，看作一个代数形式的模板，通过给 P 设置不同的值，得到不同的距离表达式。

当 P=1 时，称为 曼哈顿距离

$$d(x,y) = \sum_{k=1}^{n} \left| x_k - y_k \right|$$

当 P=2 时，称为 欧几里得距离

$$d_2 \ (x,y) \ = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

度量两点之间的直线距离，表达式和 L2 范式一样

KNN具体步骤

- 问题域
  - 有监督学习的分类问题
- 输入
  - 向量 x ：样本的多种特征信息值
  - 向量 y ：对应的类别标签
- 输出
  - 预测模型，表示是否是 正类的概率

三步：

1. 找 K 个最近邻

    a. 距离最近的 “TOP K”
2. 统计最近邻的类别占比
3. 选取占比最多的类别

3. 在 Python 中使用 KNN 分类算法

在 `neightbors` 类库下

- `KNeighborsClassifier`
    - 经典 KNN 分类算法
- `KNeighborsRegressor`
    - 利用 KNN 算法解决回归问题
- `RediusNeighborsClassifier`
    - 基于固定半径来查找最近邻的分类算法
- `NearestNeighbors`
    - 基于 `无监督` 学习实现 KNN 算法
- `KDTree`
    - 无监督学习下基于 KDTree 来查找最近邻的分类算法
- `BallTree`
    - 无监督学习下基于 BallTree 来查找最近邻的分类算法

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier

X, y = load_iris(return_X_y=True)

clf = KNeighborsClassifier().fit(X,y)
clf.predict(X)
```

性能评估器评分

```
clf.score(X, y)
```

4. 使用场景

可以解决有监督学习的分类问题、回归问题及无监督学习等

不用进行迭代逼近，只需要进行一次计算，算法复杂度低

- 优点
  - 新加入数据时，不必对整个数据集进行重新训练，可实现在线训练
- 确定
  - 对样本分布敏感，正负样本不平衡时会对预测有明显影响，数据规模大时计算量将加大
- 应用领域
  - 模式识别、文本分类、多分类领域

OCR（Optical Character Recognization，光学符号识别）

1. 确定文字所在位置区域
2. 提取特征
3. 通过KNN最近邻分类算法，判断所提取的相关特征属于哪个字符

六、朴素贝叶斯分类算法

统计学两大学派：

- 频率学派
- 贝叶斯学派

1. 朴素贝叶斯

- 条件概率
- 先验概率
- 后验概率
- 似然度

- 似然函数

基本思想

- 朴素
  - 是一种带有假设的限定条件
- 贝叶斯
  - 公式

> 指的是在“朴素”假设条件下运用“贝叶斯公式”

- $P(X)$ 表示 X 出现的概率
- $P(X|Y)$ 条件概率，在 Y 发生的条件下， X 发生的概率

贝叶斯的基本逻辑

`核心是条件概率`

贝叶斯公式预测的核心思想是：`“看起来更像”`

两轮过程：

- 第一轮的分级
  - 已知类别而统计特征，即某一特征在该类中的出现频率，是把类别分解成特征概率的过程
- 第二轮的还原
  - 已知特征而推测类别，将第一轮的结果用上，把知道统计情况的特征还原成某一类的过程

似然度（Likelihood）

2. 算法原理

$$P(\text{类别}C_1|\text{特性}X_1,\text{特性}_2,\dots)$$

在特征 X_1 、特征 X_2 、特征 X_3 等共同发生的情况下，类别 C_1 发生的概率

可以通过 `似然度` ，类求得 `后验概率`

朴素+贝叶斯

朴素（naive）地认为特征之间都是彼此独立的，使得贝叶斯公式的计算可以很大简化。

2. 数学解析

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

- $P(y)$ 称为 先验概率
- $P(y|x)$ 称为 后验概率
- $P(x|y)$ 称为 似然度
  - 似然度也是朴素贝叶斯分类算法所需要“学习”的对象

把 y 看作某个类，而把 x 当做特征，相应的贝叶斯公式为：

$$P(y|x_1,\dots,x_n) = \frac{P(y)P(x_1,\dots,x_n|y)}{P(x_1,\dots,x_n)}$$

朴素：假设特征与特征之间是相互独立、互不影响的。

简化式子， 某个特征的似然度 简化为：

$$P(x_i|y,x_1,\dots,x_{i-1},x_{i+1},\dots,x_n) = P(x_i|y)$$

马尔可夫链

后验概率：

$$P(y|x_1,\dots,x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

- $\propto$ 正比于，只需要“正比于”而不必“等于”

利用后验概率进行预测，核心方法是通过似然度预测后验概率，而学习的过程就是不断提高似然度的过程。

既然后验概率已经正比于似然度，那么提高似然度的同时，自然也就达到了提高后验概率的目的。

为了更方便使用而进行简化

与后验概率的完整等式比较可知，如果采用等式，就仍然需要统计特征共同出现的概率。

$$P(y|x_1,\times,x_n)=\frac{P(y)\prod_{i=1}^{n}P(x_i|y)}{P(x_1,\ldots,x_n)}$$

- $\prod$ 连乘号
  - 在概率里，相乘意味着求两个事件同时发生的概率
  - 连乘就是这几个事件共同发生的概率

朴素贝叶斯的优化方法

> 通过比较不同特征与类之间的似然关系，最后把似然度最大的那个类作为预测结果

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

- $\arg\max_{y}$ 参数取得什么值时，这个函数才能取得最大值，返回的是这个参数的值
- $P(y)\prod_{i=1}^{n}P(x_i|y)$ 达到最大值时， y 的取值是多少
- y 代表的是类，每个类和特征的似然度，即 $P(x_i|y)$ 是不同的
- $P(y)$ 是先验概率，是一个固定值

朴素贝叶斯算法其实是一次查表的过程，而不是过往的迭代逼近，因此不需要驱动迭代逼近的假设函数和损失函数

在部分情况如正态分布下， $P(x|y)$ 的值可以构成一个函数，称为 `似然函数（Likelihood Function）`

- 可以通过调整参数来表示不同的似然值

这种情况下，又可以采用迭代逼近方法

与之对应的`优化方法`叫做`极大似然估计（Maximum Likelihood Estimate， MLE）`

3. 具体步骤

- 问题域
  - 有监督学习的分类问题
- 输入
  - x 样本的多种特征信息值
  - y 对应的结果数值
- 输出
  - 预测模型，为线性函数

步骤

1. 统计样本数据
   a. 需要统计先验概率 $P(y)$ 和似然度 $P(x|y)$
2. 根据待预测样本所包含的特征，对不同类分别进行后验概率计算
   a. 比如总的特征有A/B/C三项，但待测样本只包含A/C两项
   b. 那 y_1 后验概率的计算方法就是 $P(y_1)P(A|y_1)P(B|y_)$
3. 比较 $y_1, y_2, \ldots, y_n$ 的后验概率，哪个最大就将其作为预测值输出

3. 在 Python 中使用朴素贝叶斯分类算法

- `sklearn.naive_bayes`
- 根据似然度计算方法不同，分为几个具体的算法分支
  - 多项式朴素贝叶斯（Multinomial Naive Bayes）
    - `MultinomialNB`
  - 伯努利分布朴素贝叶斯
    - `BernoulliNB`

- 高斯分布朴素贝叶斯
  - `GaussianNB`
- 等一共4类子算法

```
from sklearn.datasets import load_iris
from sklearn.naive_bayes import MultinomialNB
X, y = load_iris(return_X_y=True)
clf = MultinomialNB().fit(X,y)
clf.predict(X)
```

评分

```
clf.score(X,y)
```

4. 使用场景

- 优点
  - 运用了统计学成熟的理论，可解释性强，对于大规模数据集训练效率较高
- 缺点
  - 假设特征独立
- 应用领域
  - 垃圾邮件分类，文本分类

七、决策树分类算法

不是一款算法，而是一类算法

这类算法都有着类似的树形结构

除了深度学习，一切都离不开决策树算法

1. 决策树分类

Decision Tree

- XGBoost 算法
- Lightgbm 算法

> 原理：if-else 层层嵌套

区别主要在 `纯度度量` 等细节上选择了不同的解决方案

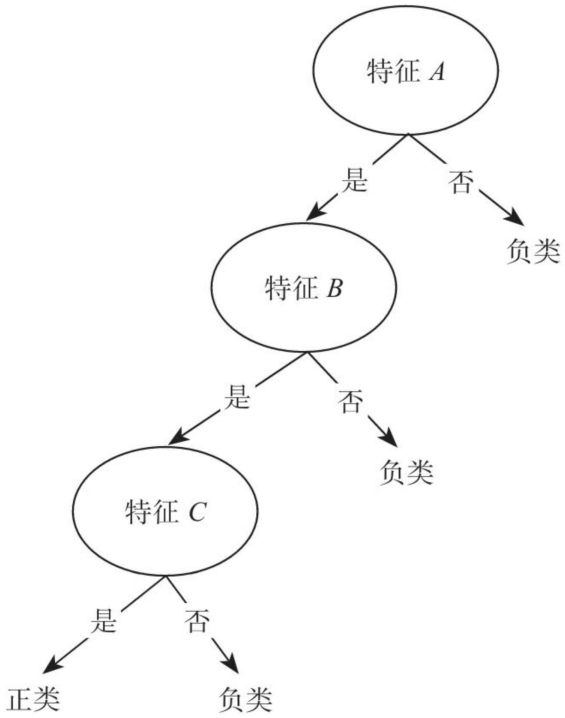
> 如何选择判断条件来生成判断分支是决策树算法的核心要点，有人称之为节点划分，也有节点分裂，指的都是生成 if-else 分支的过程。

- 决策树的分类方法
- 分支节点划分
- 纯度度量

表7–1　二元分类数据集数值表

编号	*A*	*B*	*C*	类别
1	是	是	是	正
2	是	是	否	负
3	否	是	是	负
4	是	否	是	负
5	否	否	否	负

```
if (特征A的值为“是”):
    if (特征B的值为“是”):
        if(特征C的值为“是”):
            类别=正类
        else:
            类别=负类
    else:
        类别=负类
else:
    类别=负类
```




```
有高额头吗?
有。
有亮亮的眼睛吗?
有。
有大马蹄吗?
有。

...
```

两件事：

1. 选择模型
    a. 怎样挑选判别条件
2. 往模型填数据

叛变条件从何而来？

特征维度，也是一个集合，叫做 `特征维度集`

数据样本的特征维度都可能与最终的类别存在某种关联关系，决策树的判别条件正是从这个特征维度集里产生的。

元素数据里只能称为`属性（Attribute）`，真正有助于分类的才能叫特征

把特征维度集，称为属性集

决策树最终是要解决分类问题，最理想的情况当然是选好决策条件后，正好把数据集按正类和负类分成两个部分。

“纯度”，集合中归属同一类别的样本越多，就说这个集合的纯度越高。

通过子集的纯度，越高，杂志越少，分类效果就越好。

- 节点纯度的度量规则

最著名的决策树算法一共有三种：

1. ID3
2. C4.5
3. CART

分别采用了`信息增益`、`增益率`和`基尼指数`，这三种不同的指标作为决策条件的选择依据。

这些指标都有一个共同的目的：提高分支下的节点纯度（Purity）

决策树算法中使用了大量二叉树进行判别，在一次判别后，最理想的情况就是二叉树的一个分支纯粹是正类，另一个分支纯粹是负类，完整和准确地完成了一次分类。

纯度三点：

- 当一个分支下的所有样本都属于同一个类时，纯度达到最高值
- 当一个分支下样本所属的类别一半是正类一半是负类时，纯度取得最低值
- 纯度考察的是同一个类的占比，并不在乎该类属于正还是负

纯度的度量方法

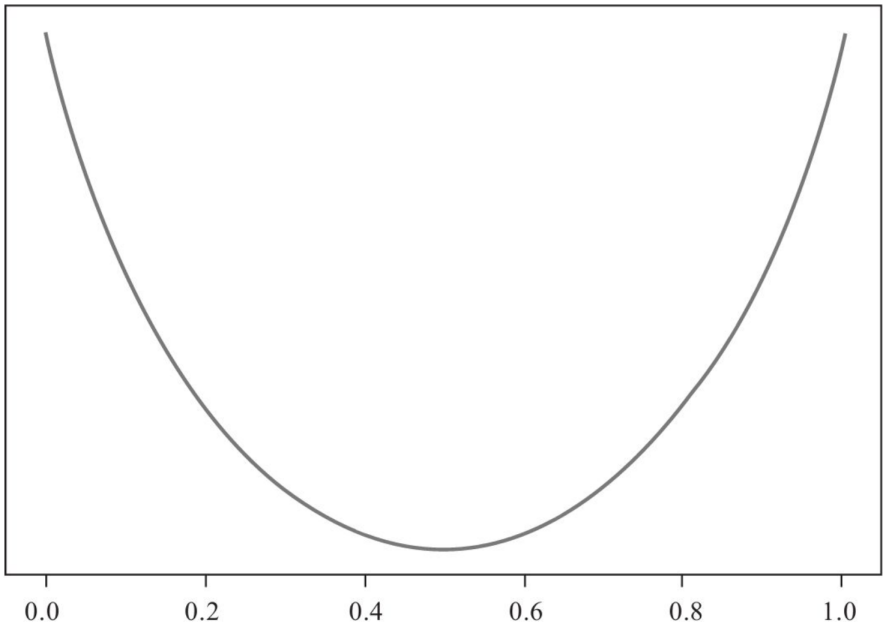


图7-2　纯度函数的函数图像

满足以上三点的图像

转换为损失值来表现。对纯度度量函数的要求正好与纯度函数的要求相反，因为纯度值越低意味着损失值越高，反之则越低。

所以纯度度量函数所做出来的图像正好相反

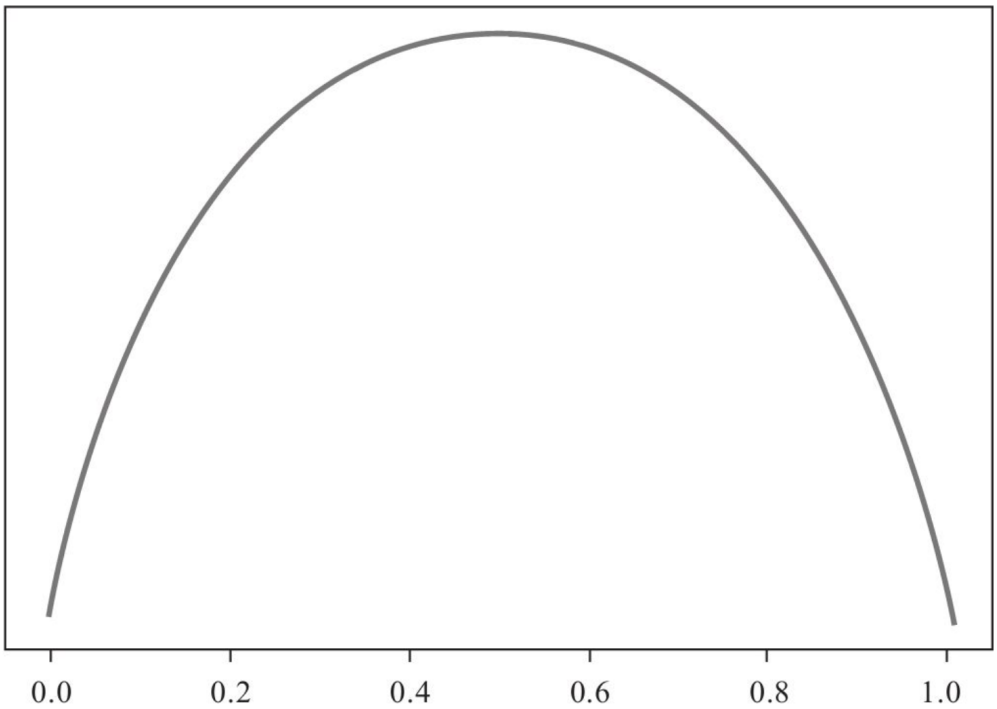


图7-3　纯度度量函数的函数图像

这就是度量纯度函数所作出来的图像。

信息增益、增益率和基尼指数这三种指标虽然在数学形式上看着不同，但作为决策条件的选择依据，图像类似。

剪枝问题

过拟合是决策树分类算法容易出现的问题，影响分类的有效性

两个停止条件：

1. 可供进行分支判断的属性维度已经全部用完
    a. 容易出现过拟合

“假性关联”问题

实际无效的属性维度，出现过度学习的情况，出现过拟合，分类有效性降低

剪枝算法

- 根据剪枝操作触发时机的不同，分为两种
    - 预剪枝
    - 后剪枝

- 预剪枝

在分支划分前进行剪枝判断

- 后剪枝

分支划分之后

- 剪枝判断
    - 遵从一个原则，如果剪枝后决策树模型的分类在 `验证集` 上的有效性能够得到提高
- 剪枝操作
    - 预剪枝
        - 不让分支生成
    - 后剪枝

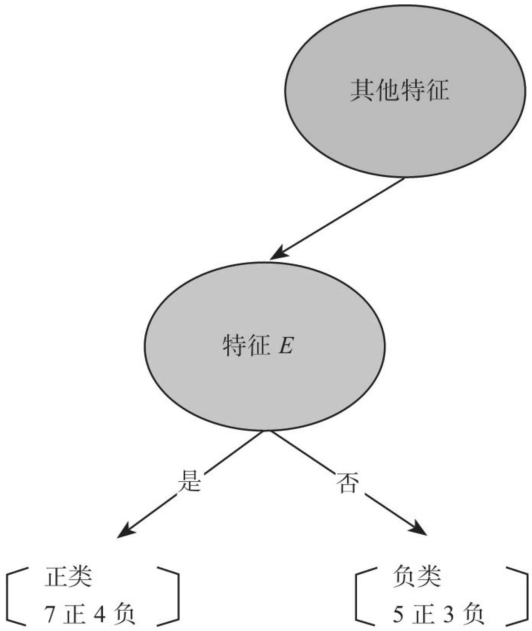


图7–4　通过某一个“冗余”特征可能得到的分类结果

剪枝前：

19个样本，7个正类，3个负类一共10个样本正确分类，这时特征 E 的分类有效性为 $\frac{10}{19}$

如果进行剪枝，根据特征 E 划分的子树去掉，取当前集合中占比最大的类，正类

则可以将 12 个正类样本正确划分，分类有效性为 $\frac{12}{19}$

剪枝后的有效性高于剪枝前的有效性，因此判断为进行剪枝

2. 算法原理

1. 基本思路

需要首先提供判别条件，从何而来？

- 来源
  - 特征维度也作为一个集合，称为 `特征维度集` ，或者 `属性集` 。
  - 判别条件从集合中来
- 选择
  - 选择哪个？
  - 比较
    - 比较标准
      - 纯度
        - 哪个特征维度“提纯”效果最好，就选

节点分裂

- 子树
- 递归生成子树

停止分裂问题，3种停止条件：

- 到达终点
  - 数据集已经完成了分类
- 自然停止
  - 如果特征维度已经全部用上了
  - 就以占比最大的类别作为当前节点的归属类别
- 选不出来
  - 提纯效果完全一样
  - 分裂到此为止

也可以通过外部阈值：

- 深度
- 叶子节点的个数等

2. 数学解析

主要区别：如何度量不同特征条件下分类结果的纯度

- ID3
- C4.5
- CART

1. 信息熵

（Information Entropy）

衡量不确定性的指标，情况越乱，信息熵越大。

信息熵的数学表达式：

$$H(X) = -\sum_{k=1}^{N} p_k \log_2(p_k)$$

- p 指概率
- X 进行信息熵计算的集合

相乘、求和、取反

在二元分类问题中，如果样本属于类别 a，a占比100%（b占0），$P_a = 1$ 时，信息熵 H 为：

$$H(1) = -(1 \times \log_2^1 + 0) = (1 \times 0 + 0) = 0$$

如果各占 50%

$$H(1) = -(0.5 \times \log_2^{0.5} + 0.5 \times \log_2^{0.5}) = -(0.5 \times -1 + 0.5 \times -1) = 1$$

信息熵函数图像

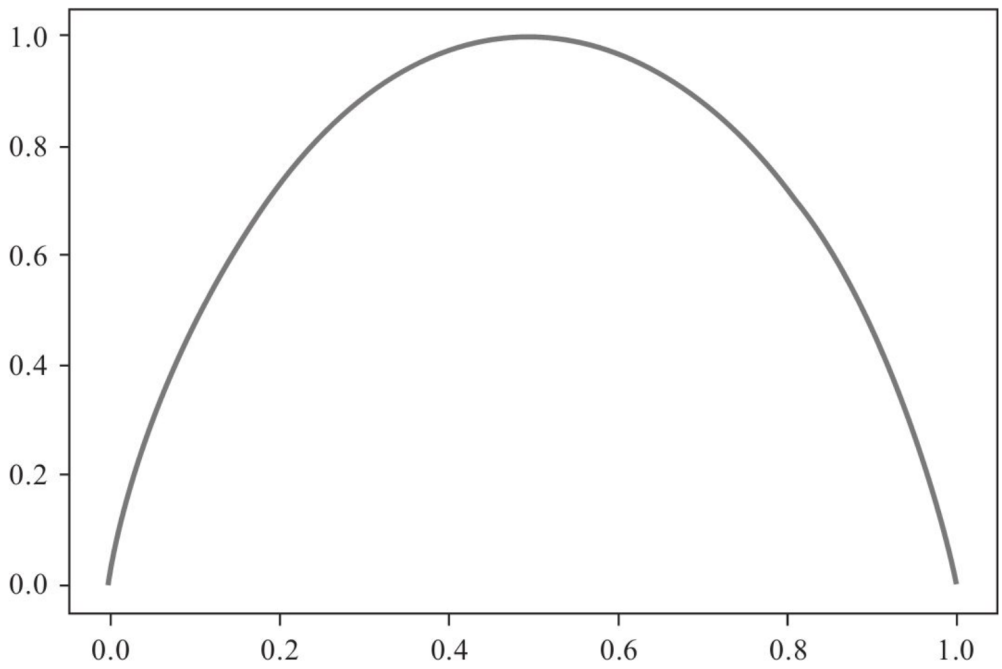


图7-8　信息熵的函数图像

信息熵以整个集合作为计算对象

怎样利用信息熵从特征维度集中选择决策条件呢?

不同的决策树算法有不同的方法

ID3 算法使用了 `信息增益 G`

经过一次分类，子集的纯度更高，说明是正面作用，纯度提升越多，说明选择的判别条件越核合适，可以作为一种不同特征维度之间的比较方法。

ID 3选择用信息熵来衡量样本集合的纯度，提纯效果好坏可以通过比较划分前后集合的信息熵来判断，`用划分前集合的信息熵-按特征属性a划分后集合的信息熵`，就得到`信息增益`。

$$G(D,a)=H(x)-\sum_{v=1}^{V}\frac{\left|D^v\right|}{\left|D\right|}H(D^v)$$

- $G(D,a)$ 集合 D 选择特征属性 a 划分子集时的信息增益。
  - 被减数是集合 D 的信息熵
- 减数
  - V 按特征维度 a 划分后，有几个子集
    - v 划分后的某一个子集
  - $\left|D\right|$ 集合的元素个数
  - $\left|D^v\right|$ 划分后的某个子集的元素个数
  - $\frac{\left|D^v\right|}{\left|D\right|}$ 一个子集的元素个数，在原集合的总元素个数的占比。就是该子集信息熵所占的权重，越大，权重越高

> 用原集合的信息熵，减去划分后产生的所有子集的信息熵的加权和，得到按特征维度 a 进行划分的`信息增益`
>
> 增益越大，提纯效果越好

ID3 喜欢选择值比较多的特征维度作为判别条件

> 特征维度的值越多，子集被切分的越细，纯度相对也是会提升，与设计初衷不符

改进版的 ID3算法，C4.5算法

唯一的区别在于，用信息增益比来代替信息增益。

信息增益与特征维度的`固有值（Intrinsic Value）`比

$$G_r=\frac{G(D,a)}{IV(a)}$$

特征维度的值越多，固有值越大。

消除多值在选择特征维度时所产生的影响

$$IV(a) = \frac{\left|D^v\right|}{\left|D\right|} \log_2 \frac{\left|D^v\right|}{\left|D\right|}$$

2. 基尼指数

CART 算法是当前最为常用的决策树算法之一，在决策条件的选择上没有沿用信息熵，而是采用了`基尼指数`，原理相似

$$Gini(D) = 1 - \sum_{k=1}^{N} p_k^2$$

基尼指数最大值 `0.5`

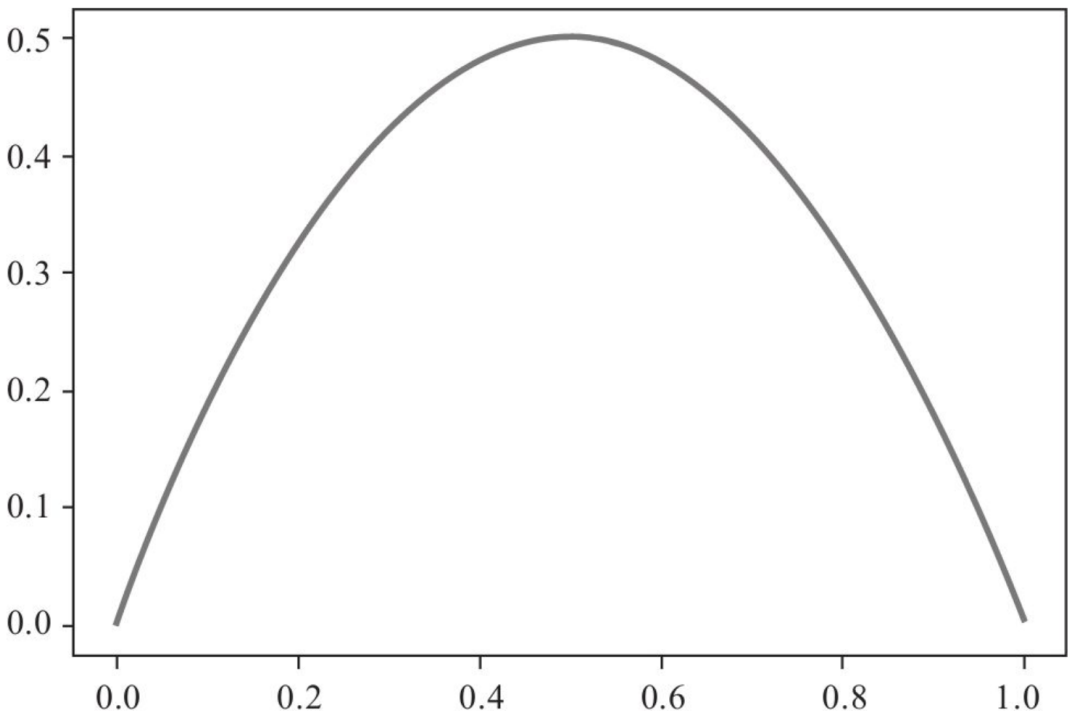


图7–9　基尼指数的函数图像

$$Gini_a = \sum_{v=1}^{V} \frac{\left|D^v\right|}{\left|D\right|} Gini(D^v)$$

元素占比 * 基尼指数，求和

3. 具体步骤

- 问题域

    - 有监督学习的分类
- 输入
- 输出

四步：

1. 选定纯度度量指标
2. 利用纯度度量指标，依次计算数据集中各特征的纯度，选取纯度能达到最大的那个特征作为该次的条件判断
3. 利用该特征作为条件判断的切分数据集，同时将该特征从切分后的子集中提出
4. 重复2和3，直到再没有特征，或切分后的数据集均为同一类

3. 在 Python 中使用决策树分类算法

`sklearn.tree`

- 解决分类问题
- 解决回归问题

- `DecisionTreeClassifier`
    - 分类算法
        - “criterion”的参数
        - 给这个参数传入字符串“gini”，使用基尼指数（默认）
        - 传入“entropy”，使用信息增益
- `DecisionTreeRegressor`
    - 反回归问题
- `ExtraTreeClassifier`
    - 在决策条件选择环境加入了随机性，不是从全部的特征维度集中选取，而是首先随机抽取 n 个特征维度来构成新的集合，然后再在新集合中选取决策条件。
    - n 的值通过参数 `max_features` 设置，为1时，相当于完全随机
- `ExtraTreeRegressor`
    - 同上

```python
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

X, y = load_iris(return_X_y=True)

clf = DecisionTreeClassifier().fit(X, y)
clf.predict(X)
```

得分

```python
clf.score(X, y)
```

4. 使用场景

两大优点：

- 分类逻辑非常清晰
- 采用树形结构进行分类，适合可视化

突出问题，过拟合

- 剪枝操作

- ID3 算法
- C4.5 算法
- CART 算法
- 在特征维度选择上都使用了统计学指标，默认特征维度之间彼此独立

- 应用领域
  - 决策问题

八、支持向量机分类算法

1. 支持向量机

三个重要构件

- 最大间隔
- 高维映射
- 核方法

线性分类器的最终形态

数据出现扰动泛化则误差会变得很大，无法有效进行正确分类

鲁棒性差

分割线距离两边都达到最大间隔

2. 支持向量

（Support Vector Machine，SVM）

- 间隔（margin）
- 只要找到两种不同的类之间的间隔，就能把两个类分开

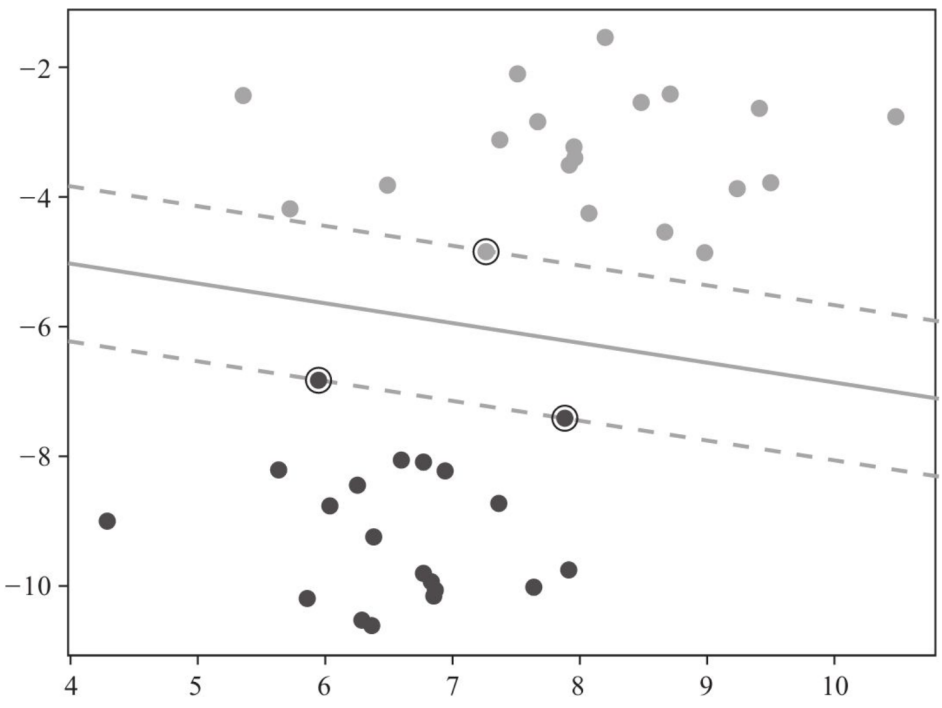


图8–1 “间隔”实际是通过两侧的样本点划分而成，这些样本点就是“支持向量”

让间隔最大化，或者让间隔变得“最胖”，就是支持向量机的目标

间隔分为两种：

- 硬间隔
- 软间隔
  - 允许犬牙交错，有容错机制，有弹性
  - 尽可能把划错率降到最小

处于边缘的数据点就称为 `支持向量` ，支持向量是支持向量机的关注焦点

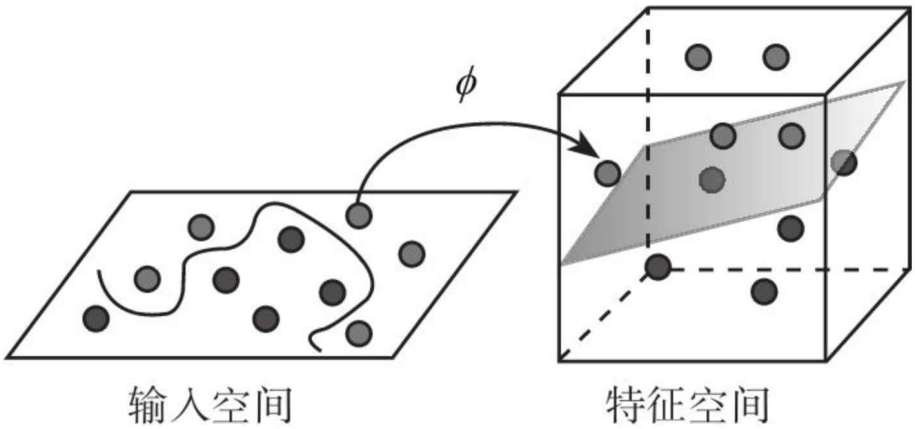
3. 线性不可分

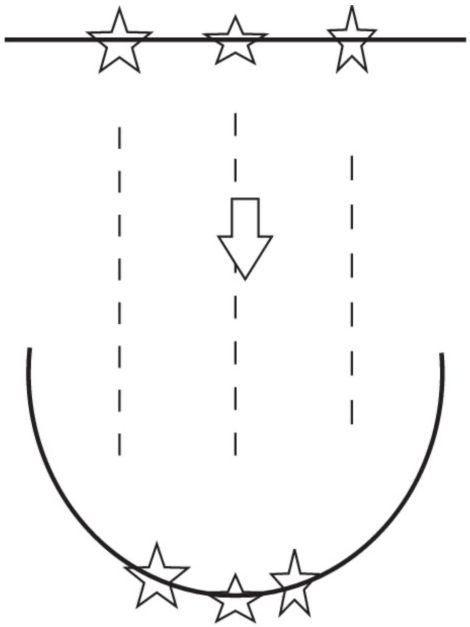
高维映射来解决

让线性不可分变为线性可分

增加维度

- 二维，线
- 三维，面
- 超三维，超平面





数据分布情况

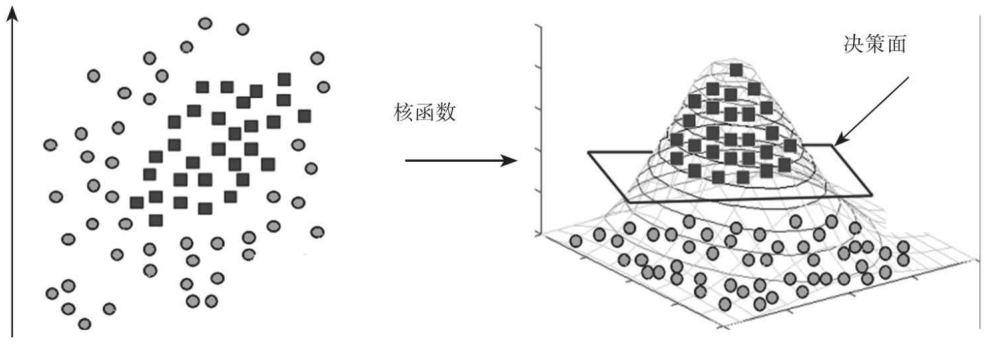


图8-4　利用“漏斗”使得二维的线性不可分数据变得可分（图片来自网络）

选择合适的映射函数

2. 算法原理

基本思路

1. 最大间隔

一种“线性分类器”，以“间隔”作为损失的度量，目标通过不断调整多维的“直线” - 超平面，使得间隔最大化。

“支持向量”，所有数据点钟直接参与计算使得间隔最大化的几个数据点

2. 高维映射

增加维度值，使得非线性分布出现了线性可分的差异，从而最终达到分离正负类的目的，实现用线性分类器对非线性可分样本点进行分类的效果

3. 核函数

（Kernel Function）功能就是映射

不是一种函数，而是一类功能性函数，能够在 SVM 中完成高维映射这种功能的函数都称为 `核函数` 。

- 增加空间的维度
- 完成对现有数据从原空间到高维空间的映射

4. 真正运行机制

三步：

1. 选取一个合适的数学函数作为核函数
2. 使用核函数进行高维映射，数据点在映射后由原本的线性不可分变为线性可分
3. 间隔最大化，用间隔作为度量分类效果的损失函数，最终找到能够让间隔最大的超平面，分类最终完成

5. 核技巧

- 核函数
  - 完成核方法提出的要求
  - 完成核技巧提出的要求
- 核方法（Kernel Method）
- 核技巧（Kernel Trick）
  - 避免高维向量的运算

计算间隔涉及向量点积运算

数学解析

1. 点到超平面的距离

以“间隔”作为损失函数，学习过程就是使得间隔最大化的过程。

间隔：作为 `支持向量的点` 到 `超平面` 的距离的 `和`

常见的距离公式

用 $\omega x + b$ 表示超平面，点到三维屏幕的距离公式：

$$d = \frac{\left|Ax_0 + By_0 + Cz_0 + D\right|}{\sqrt{A^2 + B^2 + C^2}}$$

类似的，点到 N 维超平面的距离 r

$$r^{(i)} = \frac{(\omega^T x^{(i)} + b)}{\left\|\omega\right\|}$$

- 其中 $\omega x^{(i)} + b$ 表示超平面的表达式
- 除数 $\left\|\omega\right\|$ 就是 L2 范式的简写

当 ω 是三维向量时，二者等价

SVM 使用这条公式来计算点到超平面的距离

2. 间隔最大化

使用 $y = 1$ 表示正类的分类结果，使用 $y = -1$ 表示负类的分类结果

既然 $y = \omega\ x + b$ 要么 >= 1，要么 <= -1，间隔是有正负类最近的两个数据点，也就是支持向量决定，因此间距距离也就可以表示为 $\frac{2}{\|\omega\|}$

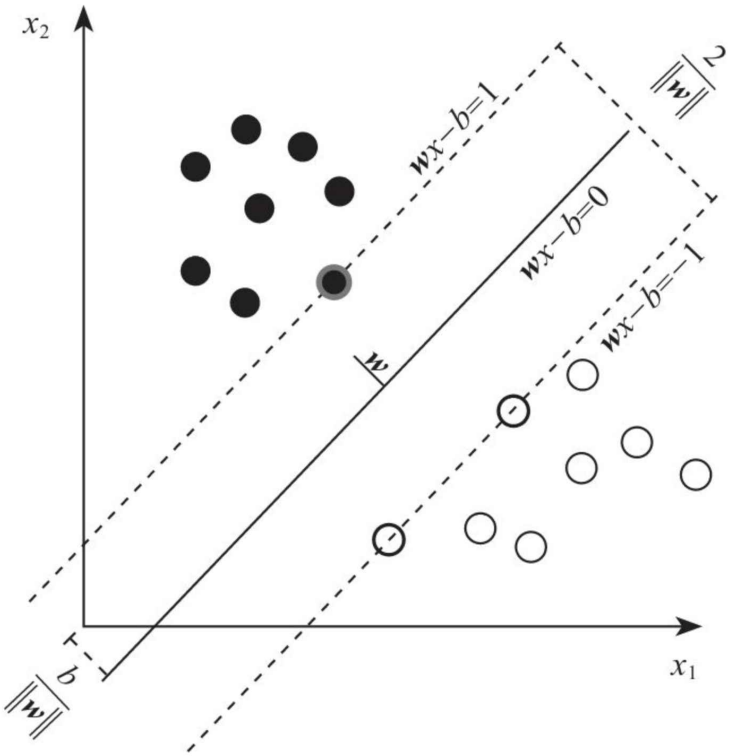


图8-5　通过支持向量计算间隔

目的是间隔最大化，2 是一个常数，最大化间距距离可以表示为：

$$\max \frac{1}{\|\omega\|} \ s.t., \ y+i(\omega^T x_i+b) \geqslant 1, \ i=1,\ldots,n$$

- 右边的 $s.t.$ 表示 subject to，意思是受到约束
  - 相当于“在...的条件下”，使得左边式子最大
  - 分母越小，分数越大

左式表示如下：

$$\min \frac{1}{2}\|\omega\|^2$$

这个式子就是要求极值，但后面还有约束条件

可以用 拉格朗日乘子法 转化成如下拉格朗日函数：

$$L(\omega,b,\alpha)=\frac{1}{2}\|\omega\|^2+\sum_{i=1}^{m}\alpha_i[1-y_i(\omega^T x_i+b)]$$

其中 α 被称为“拉格朗日乘子”。上式分别对 ω 和 b 求导，并令导数为0，右式转化为：

$$\sum_{i=1}^{m}\alpha_i-\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

问题变为：

$$\max_a \sum_{i=1}^{m} \alpha_i - \sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

约束条件为：

$$s.t. \sum_{i=1}^{m} \alpha_i y_i = 0,\ \alpha_i \geqslant 0$$

这个式子通常用二次规划算法 SMO（Sequential Minimal Optimization）算法求解。

- 使用拉格朗日乘子法搭配SMO算法求得间隔最大
- 转化式的末尾为计算 $x_i^T x_j$ ，也就是两个向量的内积
  - 正因为间隔最大化可以转化为向量内积的运算，才使得高维映射可以通过核技巧进行优化

3. 核函数

高维映射实际上也是一种函数映射，在 SVM 中，通常采用符号 ϕ 表示这个将数据映射到高维空间的函数，向量 x_i 经过高维映射后就变成了 $\phi(x_i)$ ，这时超平面的表达式也就相应变成了

$$\omega^T \phi(x_i) + b$$

根据间隔最大化的拉格朗日函数，需要进行两个向量的内积运算，映射后的内积运算为

$$\phi(x_i)^T \phi(x_j)$$

映射后向量变成高维向量

假设存在函数 K，能满足条件：

$$K(x_i, y_i) =< \phi(x_i) \cdot \phi(x_j) >= \phi(x_i)^T \cdot \phi(x_j)$$

这里的函数 K 就是我们介绍的核函数，有了核函数，所有设计 $\phi(x_i)\phi(x_j)$ 的内积运算都可以通过 $K(x_i, x_j)$ 简单求出，这就是为什么核函数一边完成核方法的高维映射，一边又要完成核技巧的求内积结果。

具体步骤

1. 选择核函数
2. 核函数完成高维映射并完成计算间隔所需的内积运算，求得间隔
3. 使用 SMO 等算法使得间隔最大

3. 在 Python 中使用 SVM 分类算法

`sklearn.svm`

- 分类问题
- 回归问题
- 无监督学习中的异常点检测

- `LinearSVC`
  - 基于线性核函数的 SVM 分类算法
- `LinearSVR`
  - 基于线性核函数的 SVM 回归算法
- `SVC`
  - 可以选择多种核函数
  - 通过 `kernel` 参数传入
    - `linear` 线性函数
    - `polynomial` 多项式函数
    - `rbf` 径向基函数（默认）
    - `sigmoid` 选择 Logistic 函数
    - `precomputed` 使用预设核值矩阵
- `SVR`
- `NuSVC`
  - 通过参数 `nu` 设置支持向量的数量
- `NuSVR`
- `OneClassSVM`
  - 解决无监督学习的异常点检测问题

4. 使用场景

小样本分类问题上的表现较好

只适用于二分类问题

九、K-means 聚类算法

无监督学习，聚类问题

最经典的聚类算法 K-means

1. 用投票表决实现“物以类聚”

半监督学习

聚类问题（Clustering），找相似

用“K”来决定归属类别

簇（Cluster）

> 样本数据集通过聚类算法最终会聚成一个个“类”，这些类就成为“簇”

“合并同类项”

聚类算法：

- 划分法
- 层次法
- 密度法
- 网格法

思路：

1. 预先设定有多少个簇
2. 其在聚类的过程中形成

K-means 聚K个类

度量“相似”的距离

KNN 使用距离作为度量工具

K-means 找“质心”

means 的由来

K 个中心点，称为“质心”

mean 是均值，可以用均值来调整质心，得到新质心的坐标值

根据全体拥有表决权的数据点的坐标来共同决定新的质心在哪里，而表决权则由簇决定。

在 K-means 聚类的过程中会多次经历质心计算，数据点归属的簇可能会频繁变动

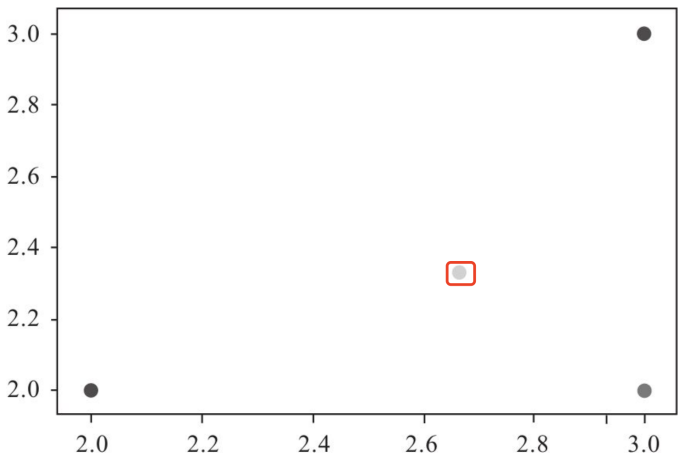


图9-3　新质心的位置示意图

2. 算法原理

聚类过程，看成是不断寻找簇的质心的过程，这个过程从随机设定 K 个质心开始，直到找到 K 个 `真正` 质心为止。

1. 有了 K 个质心
2. 对于聚成的 K 个簇，需要重新选取质心。运用了多数表决原则
3. 生成新的质心。
    a. 重复，当质心不再变化后，完成聚类

根据质心距离的远近完成一次聚类，形成 K 个类

2. 数学解析

- 找质心过程的本质

让簇内样本点到达各自质心的距离的总和最小。找到满足“最小”的K个质心

- 距离的度量

如何度量距离

欧几里得距离

度量距离的范式闵可夫斯基距离。令 P=2 时的闵可夫斯基距离就是欧几里得距离

$$d_2(x,y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

假设第 j 个簇内有 n 个数据点，根据上式，该簇的各个样本点到质心 μ_j 的距离的总和为：

$$\sum_{i=0}^{n}(\left\|x_i-\mu_j\right\|^2)$$

其中质心 μ_j 是簇内所有数据样本点取均值求出来的

找到 K 个簇组成的集合 C，使得每个簇内数据样本点到质心的距离都能达到最小，从而使距离的总和最小，也即：

$$\sum_{i=0}^{n}\min_{\mu_j\in C}(\left\|x_j-\mu_i\right\|^2)$$

将距离看成是簇内数据样本点与质心的平方误差，平方误差越小，说明簇内数据样本点围绕质心越紧，通过最小化平方误差，找到了K个簇

3. 具体步骤

- 问题域
  - 无监督的聚类算法
- 输入

    - 向量 X
- 输出
    - 预测模型，是否为该类
    - 输出对应的簇编号

五步：

1. 随机选取 K 个对象，以它们为质心
2. 计算数据集到质心的距离
3. 将对象划归（距离哪个质心最近）
4. 以本类内所有对象的均值重新计算质心，完成后进行第二步
5. 类不再变化后停止

3. 在 Python 中使用 K-means 聚类算法

最近邻模型算法族都在 `cluster` 下

- `Kmeans`
- `MiniBatchKMeans`
    - K-means 的变体，使用 mini-batch（小批量）来减少一次聚类所需的计算时间
    - 深度学习常用方法
- `DBSCAN`
    - 将聚类的类视为被低密度区域分割的高密度区域
- `MeanShift`
    - 以任意点作为质心的起点，根据距离均值将质心不断往高密度的地方移动，也即所谓 `均值漂移`
    - 当不满足漂移条件后说明密度已经达到最高，就可以划分成簇
- `AffinityPropagation`
    - 简称 `AP`
    - 聚类过程是一个“不断合并同类项”的过程，用类似归纳法的思想方法完成聚类，被称为“`层次聚类`”
- `K` 可以通过 `n_clusters` 设置，默认为 `8`

4. 使用场景

聚类过程中只涉及求均值运算，不需要进行其他太复杂的运算，执行效率较高

- 先验的设置 `K` ，由于需要求均值，要求数据集的维度属性类型应该是数值类型。

孤立点，会产生扰动

十、神经网络分类算法

1. 用神经网络解决分类问题

人工神经网络（Artificial Neural Network，ANN）

神经网络算法“三宝”：

- 神经元
- 激活函数
- 反向传播机制

神经元，又称 M-P 模型

感知机算法

- 还没有激活函数，输出都是输入的线性组合结果
- 只要简单引入非线性函数，就能打破局限

非线性函数以“激活函数”的身份被加进神经网络算法

“马文·闵斯基”

- 单层的神经网络永不可能解决非线性问题
- 没有一套很好的机制来解决多层神经网络的 `误差传递` 问题

反向传播（Back Propagation，BP）算法

- 有效解决了多层神经网络的误差传递问题

`Youshua Bengio` ， `Yann LeCun` ， `Geffrey Hinton` （深度学习三巨头）

- 卷积层
- 池化层
- 等部件

1. 神经元

两部分：

- 线性函数
- 非线性函数，称为激活函数（Activation Function，激励函数）
  - 对线性函数的输入结果进行非线性映射，然后将结果作为最终的输出。
  - Pytorch 等，就是用线性函数接激活函数来实现 `全连接层`

- 轴突（输出部分）
- 树突（输入部分）

2. 分类问题

激活函数与 Sgn 函数相似

```
if (满足激活条件):
    return 1
else:
    return 0
```

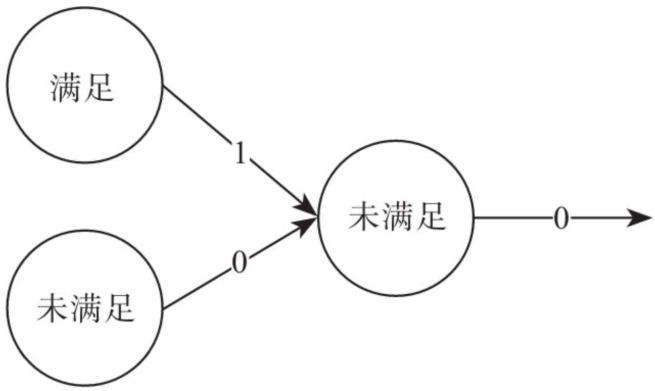
激活函数最终产生的同样是二元输出

神经网络的传递机制

- 感受刺激
- 传递兴奋

把信息一层一层接力传播下去

正向传播，通过正向传播来完成由输入数据到产生输出的过程



判断条件，满足激活条件的输出1，不满足输出0

激活条件很重要，根据刺激决定是否激活，激活就继续往前传导刺激，否则刺激就在此中断，不会对最终的输出产生影响。

典型的阶跃函数

Sgn函数，不可导

在神经网络中，通常把 Logistic 函数称为 Sigmoid 函数

Sigmoid 函数在 0 附近的函数图像变得非常平缓，特别是在 (-1, 1) 区间，几乎是一条直线，越接近 0 点，变化率就越小，在使用梯度下降等优化方法时，容易导致 `梯度弥散` 甚至 `梯度消失`

因此，业界开始用 Tanh 函数取代 Sigmoid 函数作为激活函数。

Tanh 满足可导和阶跃，梯度更大，再用梯度下降等优化方法时收敛更快，所需要的学习时间更短。

0 的问题依然存在

又开发出 ReLu 函数，目前公认效果最好

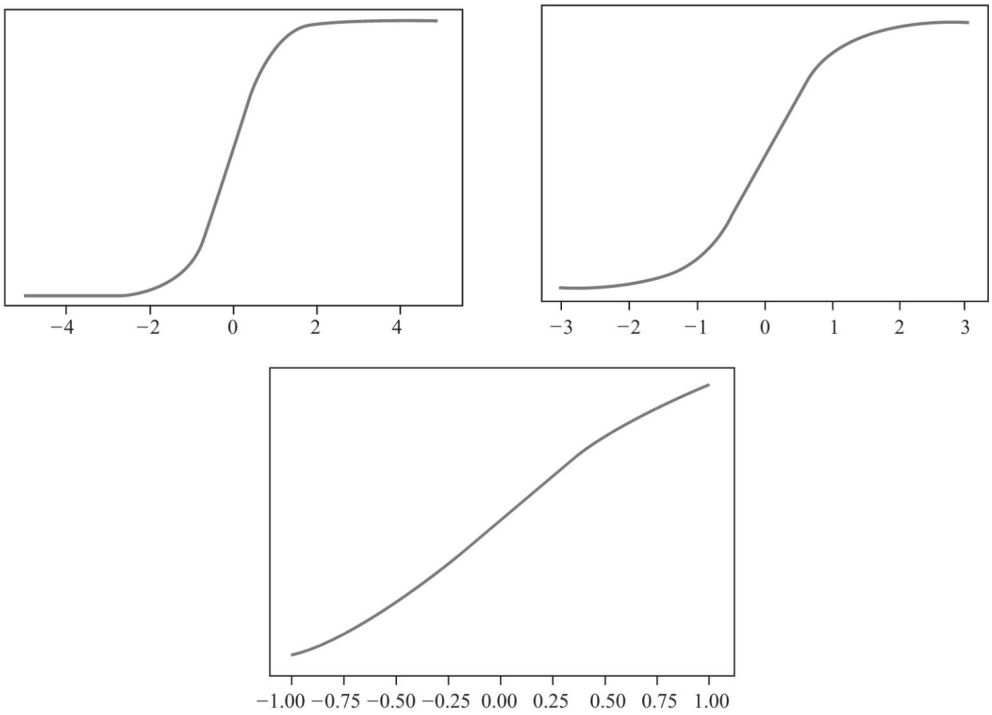


图10-5　三种条件下的Tanh函数图像

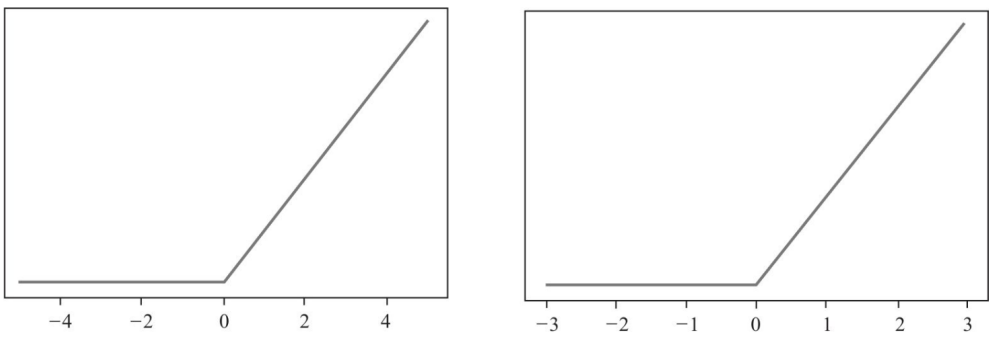


图10-6　三种条件下的ReLu函数图像

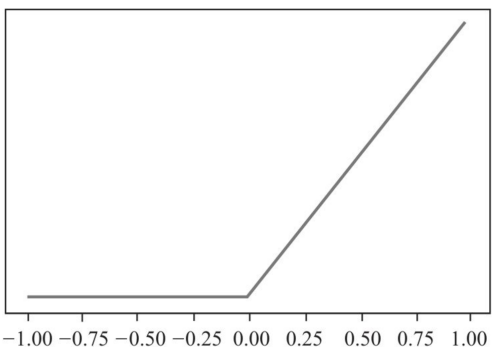


图10-6　（续）

层（Layer）

在神经网络中，数据是依靠神经网络的激活机制一个接一个地往下传递的。

把接受同样输入的神经元排列，从输入到输出的方向看，每一列就是一 层

通常由输入层、输出层和隐藏层组成：

- 直接接受输入数据的神经元是第一层，也称为 `输入层`
- 产生最终数据并输出到外部的是最后一层，也称为 `输出层`
- 其他神经元由于位于神经网络内部，称为 `隐藏层`

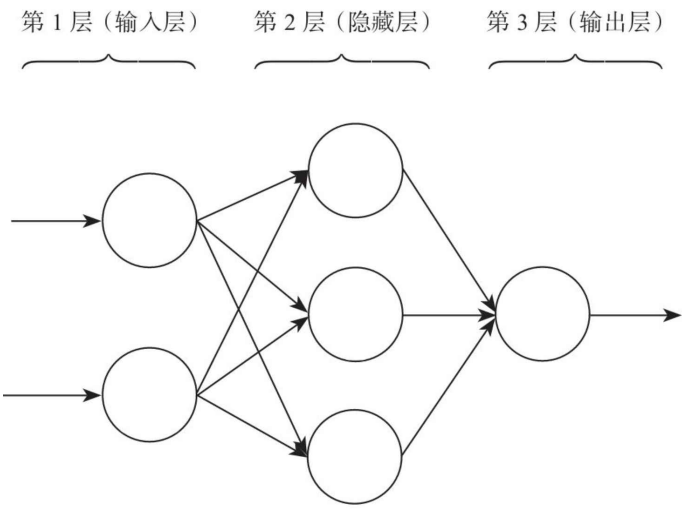


图10–7　神经网络结构示意图

5. 神经网络和深度学习

- 卷积（Convolution）层
- 池化（Pooling）层

传统神经网络：

- 激活（Activation）层
- 完全连接（Fully connectd）层

2. 算法原理

1. 基本思路

两个基本构件

- 包含非线性激活函数的神经元
- 神经元组成的网络

从神经元层面看，输入通过激励函数产生预测值，得到偏差后更新权值。

正向传播和反向传播机制

- 正向传播

    - 传播输入的功能
    - 击鼓传花
    - 一直到输出层产生输出，正向传播就完成了
- 反向传播
    - `偏差` 的传递也类似，但因为方向相反，所以称之为反向传播。
    - 首先通过输出层获取偏差，同样要计算一个值往后传递，但这时就不是通过激励函数了，而是要 `获取每个输入方向` 所贡献的 `偏差值` 。
    - 将偏差反向传播到隐藏层，让里层的神经元得到了偏差，就能把训练继续下去了
    - 不仅自己调整，还会继续向后传播偏差
        - 一直到输入层
- 整个神经网络就完成了一轮 `权值更新`

反向传播涉及偏差分配

- 根据权值取偏导数就可以计算出偏差分配
- 权值更新的方法与 Logistic 回归一样，同样是使用梯度下降

2. 数学解析

两个重要部分涉及数学：

- 激活函数
- 传递机制
    a. 正向传播
        i. 不断进行代数求值的过程
    b. 反向传播
        i. 依赖微分机理

1. 激活函数

Sigmoid 函数

Tanh 函数

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

双曲正切函数

- sin 正弦
- cos 余弦
- `h` 代表“双曲”
  - sinh 双曲正弦
  - cosh 双曲余弦

关注三点：

- Tanh 函数图像
- “激活”效果
- 代数运算

四个以自然常数 e 为底的指数函数进行四则运算

ReLu，线性整流函数（Rectified Linear Unix）

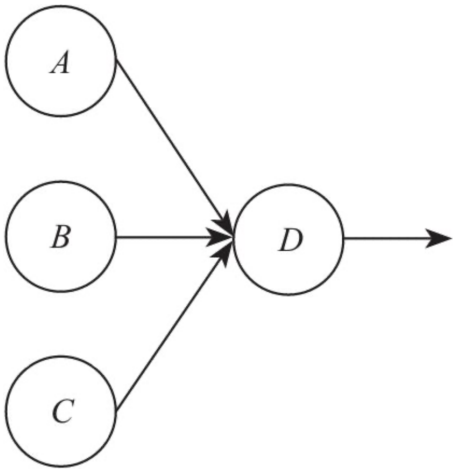
$$ReLu(x) = \max(0, x)$$

以 0 作为重要的分界点，当 x>0 时，是本身，典型的线性函数，其他情况恒等于0。这也是将 ReLu 称为线性整流函数的原因。

2. 反向传播

正向传播实际就是简单的代数赋值运算过程。

“正向”和“反向”是代数的运算方向，假设函数 f(x)=y，正向就是运算从 x 流向 y，反向就是运算从 y 流向 x



使用偏导

偏导是带有“方向”的，要求出来自神经元 A 的损失贡献，就对神经元 A 的方向求偏导。

省略激活函数，假设神经元 A 的表达式：

$$f_a(x) = w_1 x$$

同样，表示 B 和 C，假设神经元 D 的表达式：

$$f_d(x) = w_2 f_a(x) + w_3 f_b(x) + w_4 f_c(x)$$

如果已知神经元 D 输出的预测值和实际值的偏差，用 L 表示，如何求神经元 A 中参数 w_1 的调整值？也就是来自 w_1 的 损失贡献值 ，用偏导数可表示为 $\frac{\partial L}{\partial w_1}$ 。可以用“链式法则”求解，具体为：

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f_a}\frac{\partial f_a}{\partial w_1}$$

这样就求出了 w_1 对应的损失值。假设激活函数，无非多加一条导数式。

这个过程就是神经网络中偏差的反向传播，也是神经网络的学习过程。

3. 具体步骤

- 问题域：
  - 有监督学习的分类问题
- 输入
  - 向量 X，样本的多种特征信息值
  - 向量 Y，对应的结果数值

- 输出
  - 预测模型，为线性函数

五步：

1. 初始化神经网络中的神经元激励函数的权值
2. 输入层接收输入，通过正向传播产生输出
3. 根据输出的预测值，结合实际值计算偏差
4. 输出层接收偏差，通过反向传播机制让所有神经元更新权值
5. 第2~4步是神经网络模型一次完整的训练过程，重复进行训练过程，直到偏差最小。

3. 在 Python 中使用神经网络分类算法

`sklearn.neural_network`

被称为多层感知机（Multi-layer Perception）算法，MLP

- `MLPClassifier`
- `MLPRegressor`
- 基于 Bernoulli Restricted Boltzmann Machine 模型的神经网络分类算法
  - `BernoulliRBM`

4. 使用场景

“黑盒算法”

神经网络算法采用的梯度下降等优化算法，在部分情况下可能陷入局部最优解的情况，导致预测精度下降。

十一、集成学习方法

模型与模型之间的组织关系

集成学习（Ensemble Learning）

学习器（Learner）

- 基本学习器（Base Learning）
  - 同一种
- 弱学习器（Weak Learning）
  - 不同种

集成结构

- 并联
  - 训练过程并行
- 串联

整合结果：

- 平均分
  - 简单平均法
  - 加权平均法
- 投票法
  - 简单多数投票法
  - 绝对多数投票法
  - 加权投票法

2. 具体实现方式

1. Bagging 算法

Bootstrap Aggregation

并行集成学习方法

- 如何进行训练
- 如何完成预测

每个具体的学习器所使用的数据集以放回的采样方式重新生成。

在每个学习器生成训练集时，每个数据样本都有同样的被采样概率。

训练完成后，Bagging 采用投票的方式进行预测。

2. Boosting 算法

串行

前面学习器发生预测错误的数据，将在后面的训练中提高权值，而正确预测的数据则降低权值。

3. Stacking 算法

通过组合弱学习器使得预测能力增强，也就是弱学习器之间的地位是平等的。

学习器分两层：

- 第一层，若干弱学习器，分别进行预测
  - 把预测结果传递给第二层
- 第二层，通常只有一个模型
  - 给出最终预测结果

3. 在 Python 中使用集成学习方法

`ensemble` 类库下

- `RandomForestClassifier`
  - 随机森林（Random Forest）
  - 是 Bagging 集成学习算法的典型代表，选择以 CART 决策树算法作为弱学习器
- `RandomForestRegressor`
- `ExtraTreesRegressor`
  - 极端随机树
- `AdaBoostRegressor`
  - 使用 AdaBoost
  - 是最知名的 Boosting 算法之一
- `GradientBoostingClassifier`
  - Gradient Boosting 经常搭配 CART 决策树使用

    - 就是有名的梯度提升树（Gradient Boosting Decision Tree，GBDT）
- `GradientBoostingRegressor`

4. 使用场景

End：2024-08-03