# 机器学习简介

## 定义

数据 – 自动分析 – 获取模型 – 利用模型对未知数据预测

传统软件，输入+规则=输出

人工智能：输入+输出，推导（学习）出规则，进一步预测其他输入的输出

## 工作流程

获取数据 – 数据基本处理 – 特征工程 – 机器学习模型训练 – 模型评估

|  |
| --- |
| **完整机器学习项目的流程（拓展阅读）：**  1 抽象成数学问题  明确问题是进行机器学习的第一步。机器学习的训练过程通常都是一件非常耗时的事情，胡乱尝试时间成本是非常高的。  这里的抽象成数学问题，指的明确我们可以获得什么样的数据，抽象出的问题，是一个分类还是回归或者是聚类的问题。  2 获取数据  数据决定了机器学习结果的上限，而算法只是尽可能逼近这个上限。  数据要有代表性，否则必然会过拟合。  而且对于分类问题，数据偏斜不能过于严重，不同类别的数据数量不要有数量级的差距。  而且还要对数据的量级有一个评估，多少个样本，多少个特征，可以估算出其对内存的消耗程度，判断训练过程中内存是否能够放得下。如果放不下就得考虑改进算法或者使用一些降维的技巧了。如果数据量实在太大，那就要考虑分布式了。  3 特征预处理与特征选择  良好的数据要能够提取出良好的特征才能真正发挥作用。  特征预处理、数据清洗是很关键的步骤，往往能够使得算法的效果和性能得到显著提高。归一化、离散化、因子化、缺失值处理、去除共线性等，数据挖掘过程中很多时间就花在它们上面。这些工作简单可复制，收益稳定可预期，是机器学习的基础必备步骤。  筛选出显著特征、摒弃非显著特征，需要机器学习工程师反复理解业务。这对很多结果有决定性的影响。特征选择好了，非常简单的算法也能得出良好、稳定的结果。这需要运用特征有效性分析的相关技术，如相关系数、卡方检验、平均互信息、条件熵、后验概率、逻辑回归权重等方法。  4 训练模型与调优  直到这一步才用到我们上面说的算法进行训练。现在很多算法都能够封装成黑盒供人使用。但是真正考验水平的是调整这些算法的（超）参数，使得结果变得更加优良。这需要我们对算法的原理有深入的理解。理解越深入，就越能发现问题的症结，提出良好的调优方案。  5 模型诊断  如何确定模型调优的方向与思路呢？这就需要对模型进行诊断的技术。  过拟合、欠拟合 判断是模型诊断中至关重要的一步。常见的方法如交叉验证，绘制学习曲线等。过拟合的基本调优思路是增加数据量，降低模型复杂度。欠拟合的基本调优思路是提高特征数量和质量，增加模型复杂度。  误差分析 也是机器学习至关重要的步骤。通过观察误差样本全面分析产生误差的原因:是参数的问题还是算法选择的问题，是特征的问题还是数据本身的问题……  诊断后的模型需要进行调优，调优后的新模型需要重新进行诊断，这是一个反复迭代不断逼近的过程，需要不断地尝试， 进而达到最优状态。  6 模型融合  一般来说，模型融合后都能使得效果有一定提升。而且效果很好。  工程上，主要提升算法准确度的方法是分别在模型的前端（特征清洗和预处理，不同的采样模式）与后端（模型融合）上下功夫。因为他们比较标准可复制，效果比较稳定。而直接调参的工作不会很多，毕竟大量数据训练起来太慢了，而且效果难以保证。  7 上线运行  这一部分内容主要跟工程实现的相关性比较大。工程上是结果导向，模型在线上运行的效果直接决定模型的成败。 不单纯包括其准确程度、误差等情况，还包括其运行的速度(时间复杂度)、资源消耗程度（空间复杂度）、稳定性是否可接受。  这些工作流程主要是工程实践上总结出的一些经验。并不是每个项目都包含完整的一个流程。这里的部分只是一个指导性的说明，只有大家自己多实践，多积累项目经验，才会有自己更深刻的认识。 |

## 代码框架

|  |
| --- |
| # 导入模块  import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split, GridSearchCV from sklearn.preprocessing import StandardScaler  # 1、获取数据‘  # 2、数据基本处理  ## 确定目标值、特征值  ## 处理缺失值  ## 数据集划分  # 3、特征工程  ## 特征预处理  ### 标准化  ### 归一化  ## 特征抽取  ### 字典、文本特征抽取  ## 特征降维  ### 低方差过滤  ### 相关系数  ### 主成分分析PCA  # 3、机器学习  ## 实例化估计器  ## 训练  # 4、模型优化  ## 交叉验证  ## 网格搜索 |

## 获取到的数据集介绍

### 专有名词

* 样本
* 特征
* 目标值（标签值）
* 特征值

### 数据类型构成

类型一：特征值加目标值：目标值是离散还是连续

类型二：只有特征值，没有目标值

### 数据划分

训练数据（训练集） -- 构建模型 – 占比70%~80%

测试数据（测试集） -- 模型评估 – 占比20%~30%

### 数据获取

<https://www.kaggle.com/>

竞赛和数据网站

## 数据基本处理

对数进行缺失值、去除异常值等处理

## 特征工程

### 定义

把数据转换成为机器更容易识别的数据

### 为什么需要特征工程

数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已

### 包含内容

特征提取

特征预处理

特征降维

### 机器学习

选择合适的算法对模型进行训练

### 模型评估

对训练好的模型进行评估

## 库的安装

matplotlib==2.2.2

numpy==1.14.2

pandas==0.20.3

tables==3.4.2

jupyter==1.0.0

pip3 install scikit-learn

# 机器学习算法分类

## 监督学习 -- 有特征值，有目标值

目标值连续-- 回归

目标值离散-- 分类

## 无监督学习 -- 仅有特征值

## 半监督学习

有特征值，但是一部分数据有目标值，一部分没有

## 强化学习

动态过程，上一步数据的输出是下一步数据的输入

四要素：agent, action, environment,Reward,

# 模型评估

## 分类模型评估

准确率

精确率

召回率

F1-score

AUC指标

## 回归模型评估

均方根误差

相对平方误差

平均绝对误差

相对绝对误差

决定系数

## 拟合

欠拟合：判断不出：训练集不好，测试集也不好

过拟合：判断错误：训练集很好，测试集不好

# jupyter

|  |
| --- |
| C:\Users\liusx\AppData\Local\Temp\1648357113(1).png |

## 基础

### **按H可以看快捷键**

* windows账户名字不能是中文，不然jupyter用不了
* ==============jupyter notebook============
* 先进入到目标文件夹
* 在路径处输入cmd打开命令行
* 更改jupyter的配置文件，更改jupyter notebook的默认打开方式为chrom
* 命令行输入jupyter notebook进入环境
* 新建python3脚本
* 点击输入框进入编辑模式
* ctrl+enter运行
* shift+enter运行并到下一栏
* ===========命令模式===========
* 按esc进入命令模式
* 按a添加一行在上面
* 按b添加一行在下面
* 按dd删除一行
* 按m添加笔记
* 按z回退
* crtl+home跳到第一个cell
* crtl+end 跳到最后一个cell
* cell前面有\*号，表示正在运行
* ==========编辑模式==========
* 按enter进入编辑模式
* ctrl 点击鼠标进入多光标操作
* ctrl+z 回退
* ctrl+y 重做
* tab 补全代码
* ctrl+/ 添加注释
* ===========markdown=========
* esc进入命令模式后按m可快捷切换为markdown格式
* ‘# ‘ 一级标题
* ‘## ‘ 二级标题，以此类推
* tab 缩进
* 插件pip install jupyter\_contrib\_nbextensions -i <https://pypi.douban.com/simple>

## 故障

|  |
| --- |
| import \_ssl报错：  检查环境变量和对应的解释器选择无误  D:\data\wechat\WeChat Files\wxid_jdmay7f9jag622\FileStorage\Temp\1683125924484.png |

# Matplotlib

## 基础绘图

* =======================绘图流程==========================
* 导入 import matplotlib.pyplot as plt
* plt.rcParams['font.sans-serif']=['SimHei'] #设置字体，避免没有字体库
* 1.创建画布 plt.figure(figsize=(5,5),dpi=100)
* 2.建立数据 x=[1,2,3] y=[4,5,6]
* 3.绘制图像 plt.plot(x,y, label = '北京', color = 'g',linestyle = '-.')
* plt.title(**'GDP时间增长'**) # 增加标题
* 4.添加图例 plt.legend(loc='best') # loc是位置
* 5.添加x、y轴刻度 plt.xticks(位置列表，文本列表) #注意位置列表的数子大小
* plt.yticks(位置列表，文本列表)
* 6.添加网格 plt.grid(True,linestyle = '-',alpha = 0.5) #是否添加，样式，透明度
* 7.添加描述 plt.xlabel(‘时间‘,fontsize = 20) plt.title(‘温度时间变化’,fontsize = 20)
* 8.图像保存 plt.savefig(‘./date/test.png’) # .是上级目录的意思
* 9.显示图像 plt.show() #同时会释放内存中的资源
* 10.关闭 plt.close()
* 安装字体SimHei.tff : <https://zhuanlan.zhihu.com/p/50957403>

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  *# 1.创建画布* plt.rcParams[**'font.sans-serif'**]=[**'SimHei'**] plt.figure(figsize=(8,5),dpi=100)  *# 2.建立数据* x = [2000+i **for** i **in** range(10)] y = [1658,1687,894,1568,2321,1444,2311,4546,5555,6879] y2 = [2658,2687,1894,1598,3321,1444,3311,4546,5155,6279]  *# 3.绘制图像 # plt.plot(x,y, label = '北京', color = 'g') # 折线图 # plt.plot(x,y2, label = '上海', color = 'r') # 折线图* plt.bar(x, y, width=0.8, align=**'center'**,color=[**'g'**]) *# 柱状图* plt.xticks(x, [**f'{**i**}year' for** i **in** range(10)]) plt.legend(loc=**'best'**) plt.title(**'GDP时间增长'**, ) plt.show() |

* =====================多坐标系绘图流程=======================
* 导入 import matplotlib.pyplot as plt
* 1.创建画布 fig,axes = plt.subplots(nrows=1,ncols=2,figsize=(20,8),dpi=100)
* 2.建立数据 x=range(40) y\_beijing=range(0,80,2) y\_shanghai=range(1,81,2)
* 3.绘制图像 axes[0].plot(x,y\_beijing,label=’北京’,color=’g’,linestyle=’-.‘)

axes[1].plot(x,y\_shanghai,label=’上海’,color=’r’,linestyle=’-‘)

* 4.添加图例 axes[0].legend(loc=0) axes[1].legend(loc=0)
* 5.添加x、y轴刻度

|  |
| --- |
| y\_ticks=range(0,100,5)  x\_ticks=x[::5]  x\_ticks\_labels=[f’11点{i}分’ for I in range(40)][::5]  axes[0].set\_xticks(x\_ticks)  axes[0].set\_yticks(y\_ticks)  axes[0].set\_xticklabels(x\_ticks\_labels)  axes[1].set\_xticks(x\_ticks)  axes[1].set\_yticks(y\_ticks)  axes[1].set\_xticklabels(x\_ticks\_labels) |

* 6.添加网格

|  |
| --- |
| axes[0].grid(True,linestyle = '-',alpha = 0.5) #是否添加，样式，透明度  axes[1].grid(True,linestyle = '-',alpha = 0.5) #是否添加，样式，透明度 |

* 7.添加描述

|  |
| --- |
| axes[0].set\_xlabel(‘时间‘,fontsize = 15)  axes[0].set\_ylabel(‘温度‘,fontsize = 15)  axes[0].set\_title(‘北京40分钟温度变化‘,fontsize = 20)  axes[1].set\_xlabel(‘时间‘,fontsize = 15)  axes[1].set\_ylabel(‘温度‘,fontsize = 15)  axes[1].set\_title(‘上海40分钟温度变化‘,fontsize = 20) |

* 8.图像保存 plt.savefig(‘./date/test\_mut\_axes.png’) # .是上级目录的意思
* 9.显示图像 plt.show() #同时会释放内存中的资源
* 10.关闭 plt.close()

|  |
| --- |
| def figure(num=None, # autoincrement if None, else integer from 1-N  figsize=None, # defaults to rc figure.figsize  dpi=None, # defaults to rc figure.dpi  facecolor=None, # defaults to rc figure.facecolor  edgecolor=None, # defaults to rc figure.edgecolor  frameon=True,  FigureClass=Figure,  clear=False,  \*\*kwargs  )  解释：  num: int 或者 string 类型，相当于该 figure 的 id。如果没有定义，则每次创建时 num 为整型，并自动增加。如果 num 已经定义（即拥有该 id 的 figure 已经存在），则使该 figure 处于活跃状态，并返回该 figure 的引用。如果 num 的值为 string，则窗口标题被设置为该 num。  figsize: tuple 类型，如figsize = (9,6) ，代表设置 figure 的长为9英寸，宽为6英寸。（1inch = 2.54cm）  dpi : 整型，代表图片的分辨率。  facecolor： 背景颜色。  edgecolor： 边框颜色。  frameon：布尔型，默认值 True 为绘制边框，如果为 False 则不绘制边框。  clear: 布尔型，如果 figure 已经存在并且 clear = True ，则清除该 figure 上已经绘制的东西。 |

## Matplotlib三层架构

### 容器层

* Canvas：最底层的系统层，相当于画板，上面放置画布
* Figure：用户操作的第一层，相当于画布
* Axes：用户操作的第二层，相当于绘图区

### 辅助显示层

* 绘图区Axes除数据绘制出的图像以外的内容
* 添加X轴、Y轴描述，标题等

### 图像层

* 通过plot等函数绘制出来的图像

## 绘制数学图像

|  |
| --- |
| import numpy as np  #准备数据  x = np.linspace(-10,10,1000,ture) #序列起始值，终止值，样例数量，是否包含终止值  y = np.sin(x)  y1 = x+1  #创建画布  plt.figure(figsize=(10,3),dpi=100)  #绘制  plt.plot(x,y)  plt.plot(x,y1)  #显示  plt.show() |

## 其他基本图像绘制

|  |
| --- |
| import matplotlib.pyplot as plt  散点图：  api: plt.scatter(x,y)  x, y → 散点的坐标，float or array-like, shape (n, )  s → 散点的面积，float or array-like, shape (n, ), optional  c → 散点的颜色（默认值为蓝色，'b'，其余颜色同plt.plot( )）  marker → 散点样式（默认值为实心圆，'o'，其余样式同plt.plot( )）  alpha → 散点透明度（[0, 1]之间的数，0表示完全透明，1则表示完全不透明）  linewidths →散点的边缘线宽  edgecolors → 散点的边缘颜色  柱状图：  api: plt. bar(x, y, width=0.8, align='center',color=[‘g’]) #ticks修改坐标名称  直方图：  api:plt.hist(x,bins=None) #bins为分组数，直方图纵坐标是x里元素出现的次数  饼图：  api: plt.pie(x,label=,autopct= "%.1f%%",colors) # label每部分名称，autopct占比显示指定 |

## seaborn库

基于matplotlib的高级封装库

|  |
| --- |
| 接口演示：  from sklearn.datasets import load\_iris import matplotlib.pyplot as plt import pandas as pd import seaborn as sn   def polt\_iris(pd\_df, df\_col1\_name, df\_col2\_name):  sn.lmplot(x=df\_col1\_name, y=df\_col2\_name, data=pd\_df, fit\_reg=True, hue='鸢尾花种类')  # data- df数据类型  # x - 图坐标x的数据来源df的key  # y - 图坐标x的数据来源df的key  # hue - 图坐标的数据按df的此key分类为不同颜色  # fit\_reg - 是否进行线性拟合  plt.xlabel(df\_col1\_name)  plt.ylabel(df\_col2\_name)  plt.title(f'鸢尾花的{df\_col1\_name}和{df\_col2\_name}坐标图')  # 标题  plt.show()   iris = load\_iris() iris\_pd = pd.DataFrame(iris.data, columns=iris.feature\_names) print(iris.target\_names) iris\_pd['鸢尾花种类'] = iris.target  polt\_iris(iris\_pd, iris.feature\_names[0], iris.feature\_names[1]) |

# numpy

## 基础

* ==========================定义===========================
* 开源的python科学计算库
* 用于快速处理任意维度的数组
* numpy中，存储对象是ndarray
* ==========================创建===========================
* np.array([ ])
* ==========================优势===========================
* 内存是一体式存储，读取效率高（python是分配式存储，效率低但是可以同时存储多种类型变量）
* 支持并行运算，底层使用c语言，内部释放了GIL，运行效率高

|  |
| --- |
| %time sum=sum(s) # 计算这条命令的运行时间  %timeit sum=sum(s) # 精确计算这条命令的运行时间 |

## 创建N维数组

### 属性

|  |
| --- |
| import numpy as np  narray=np.array([2,3],[4,5])  narray.shape #数组维度元组，几维几行几列 shape[0]#第1维的数量  narray.ndim #数组维度  narray.size #数组元素个数  narray.itemsiaze #一个数组元素的长度(字节)  narray.dtype #数组元素类型 |

### 类型

|  |
| --- |
| import numpy as np  narray=np.array([2,3],[4,5],dtpye=np.float32)  #dtpye = bool,int,float,str…… |

## 操作N维数组

### 生成数组的方法

|  |
| --- |
| import numpy as np  ======================生成0和1数组======================  np.ones((x,y)) #参数个数代表生成数组维，a, b, c,依次表示外层到内层，填充1  np.ones\_like(w) #复制w数组的属性，填充1  np.zero([x,y]) #参数个数代表生成数组维，a, b, c,依次表示外层到内层，填充0  np.zero(w) #复制w数组的属性，填充0  ========================复制数组=========================  a=np.array([1,2,3])  a1=np.array(a) #深拷贝，复制数据重新创建内存  a2=np.asarray(a) #浅拷贝，复数据的内存地址，相当于创建快捷方式  =====================生成固定范围数组=======================  np.linspace(-10,10,1000,ture) #序列起始值，终止值，样例数量，是否包含终止值  np.arange(10,50,2) #起始值，终止值，间隔  np.logspace(x,y,count) #生成从10的x次方到10的y次方，生成count个数 |

### 正态分布数学基础

|  |
| --- |
| =====================正态分布数学基础========================    若随机变量X服从一个数学期望为μ、方差为σ2的正态分布，记为N(μ，σ2)。其概率密度函数为正态分布的期望值μ决定了其位置，其标准差σ决定了分布的幅度。当μ = 0,σ = 1时的正态分布是标准正态分布。  ==========================数学期望=========================  离散型随机变量的一切可能的取值与对应的概率乘积之和  =======================方差===============================    方差刻画了随机变量的取值对于其数学期望的离散程度。（标准差、方差越大，离散程度越大，数据越分散） |

### 生成随机数random

|  |
| --- |
| import numpy as np  ##########################正态分布#############################  np.random.normal(loc=0.0, scale=1.0, size=None)  loc：float  ​此概率分布的均值（对应着整个分布的中心centre）  scale：float  ​此概率分布的标准差（对应于分布的宽度，scale越大越矮胖，scale越小，越瘦高）  size：int or tuple of ints  ​输出的shape，默认为None，只输出一个值。参数个数代表生成数组维，a, b, c,依次表示外层到内层。如size=10、size=(2,3)  ###################################################  np.random.standard\_normal(size=None)  返回指定形状的标准正态分布的数组。  #######################均匀分布##################################  np.random.uniform(low=0.0, high=1.0, size=None)  功能：从一个均匀分布[low,high)中随机采样，注意定义域是左闭右开，即包含low，不包含high.  参数介绍:  low: 采样下界，float类型，默认值为0；  high: 采样上界，float类型，默认值为1；  size: 输出样本数目，为int或元组(tuple)类型，例如，size=(m,n,k), 则输出mnk个样本，缺省时输出1个值。  返回值：ndarray类型，其形状和参数size中描述一致。  ####################################################  np.random.randint(low, high=None, size=None, dtype='l')  从一个均匀分布中随机采样，生成一个整数或N维整数数组，  取数范围：若high不为None时，取[low,high)之间随机整数，否则取值[0,low)之间随机整数。 |

### 数组的索引切片

|  |
| --- |
| a1 = np.array([ [[1,2,3],[4,5,6]], [[12,3,34],[5,6,7]]])  a1[start1:end1, start2:end2, start3:end3] #从第一维往后切片  # 二三行互换  arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])  # 把第二行和第三行互换  arr[[1, 2]] = arr[[2, 1]] |

### 数组的形状修改

|  |
| --- |
| s\_new = s.reshape([2,10]) #将数组s转换成2行 10列，注意转换总数必须相等  s\_new = s.reshape([-1,x]) #转换成x列（必须能整除）  s.resize([2,10]) # 无返回值，直接转换  s\_new = s.T #将s转置 |

### 转换

|  |
| --- |
| s\_new = s.astpye(np.int32) #数组s转变成int型  s\_new = s.tostring() #构造包含数组中原始数据字节的Python字节  s.tolist() # 数组转换成列表  # 二维数组展平为一维  arr = np.array([[1, 2], [3, 4]])  arr= np.ravel(arr) # [1 2 3 4] # 也可以写为arr.ravel()  # flatten展平和ravel一致，区别是改动ravel会影响原数组，faltten不会  np.pad(i, (5, 4), constant\_values=0) # 向i中填充字符0，左边5个右边4个 |

### 数组去重

|  |
| --- |
| s\_new = np.unique(s) # 给数组s去重  arr.unique() # arr数组去重 |

## 数组运算

### 逻辑运算

|  |
| --- |
| score = np.random.randint(40, 100, (4, 3))  score > 60 #判断每个元素大于，返回每个元素的bool  score[score > 60] = 1 #将大于的元素赋值为1 |

|  |
| --- |
| numpy.logical\_and是一个逻辑函数，它可以帮助用户逐元素地计算x1和x2的真值。如果x1.shape不等于x2.shape，它们必须能够广播到一个公共形状（这就成为输出的形状）。  import numpy as np  x = np.arange(5)  np.logical\_and(x>1, x<4)  输出：  array([False, False, True, True, False])  同理还有numpy.logical\_or |

### all和any

|  |
| --- |
| np.all(score > 60) #所有元素大于返回True  np.any(score > 60) #任何一个大于返回True  np.any(a,axis=0) # 按横轴计算。axis=1按纵轴计算。axis=None会展平一起计算 |

### np.where（三元运算符）

|  |
| --- |
| np.where(score>60, score<90, 1, 0) #满足条件的元素赋值为1，否则为0  np.where(np.logical\_and(temp > 60, temp < 90), 1, 0) #同时满足赋值为1  np.where(np.logical\_or(temp > 90, temp < 60), 1, 0) #满足一项复制为1  np.where(output == 1) # 找出output里为1的索引 |

### 功能

|  |
| --- |
| np.equal(arr1,arr2) # 判断两个数组对应位置是否相等，相等为True |

### 计数

|  |
| --- |
| numpy.count\_nonzero 是一个用于统计数组中非零元素个数的函数  语法是 numpy.count\_nonzero(a, axis=None, \*, keepdims=False)  a 是需要统计的数组，axis 是可选参数，指定沿哪个轴或哪些轴统计非零元素，keepdims 是可选的布尔参数，如果设置为 True，则计数的轴将保留在结果中  # 变种使用. a == 2 返回一个与 a 形状相同的布尔数组，其中值为 2 的位置为 True，其他位置为 False。然后，我们使用 np.count\_nonzero 来统计 True 的数量，即值为 2 的元素的数量。  count = np.count\_nonzero(arr == 2) # 统计数组中为2的个数 |

|  |
| --- |
| import numpy as np  a = np.array([[1,2,0],[0,1,2]])  count = np.sum(a == 2)  print(count) # 2  使用布尔索引 a == 2 来获得一个与 a 形状相同的布尔数组，然后使用 np.sum 来对布尔数组求和。由于在求和过程中，True 被视为 1，False 被视为 0，因此最终的结果就是值为 2 的元素的数量。 |

### 统计运算

|  |
| --- |
| np.max(score, axis=1) #返回每行最大值 # axis=0为列  np.min(score, axis=1)  np.median(score, axis=1) #返回每行中位数  np.mean(a, axis, dtype) #平均值  np.std(a, axis, dtype) #标准差  np.var(a, axis, dtype) #方差  np.argmax(axis=) — 最大元素对应的下标  np.argmin(axis=) — 最小元素对应的下标   * ###########TIPS############# * 所有统计函数不加axis参数时，是在整个数组中进行比较 |

### 算术运算

|  |
| --- |
| ===============数组与数==================  arr = np.array([[1, 2, 3, 2, 1, 4], [5, 6, 1, 2, 3, 1]])  arr + 1 #二维数组每个元素+1  arr / 2 ##二维数组每个元素/2  ===============数组与数组=================  数组运算,满足广播机制：  1.维度相等  2.shape(其中对应的地方为1,也是可以的)    3.相加时对于相加，相乘时对应相乘，和矩阵不一样 |

## 矩阵

### 加减运算

|  |
| --- |
|  |

### 乘法

|  |
| --- |
| * 必须是[m,n]\*[n,o]=[m,o],结果的第x行第y列等于[m,n]的x行和[n,o]的y列相乘相加 * 不满足交换律 A×B≠B×A * 满足结合律 A×（B×C）=（A×B）×C * 单位矩阵：它是个**方阵**，一般用 I 或者 E 表示，从**左上角**到**右下角**的对角线（称为主对角线）上的元素均为 1 以外全都为 0 * 矩阵的逆：如矩阵 A 是m×m 矩阵（**方阵**），如果有逆矩阵，则：AA-1 = A-1A = I。A⁻¹=A\*/|A| * 低阶矩阵求逆的方法: ​ 1.待定系数法 ​ 2.初等变换https://jingyan.baidu.com/article/1709ad8095e1924634c4f03a.html * 伴随矩阵是矩阵元素所对应的代数余子式，所构成的矩阵，转置后得到的新矩阵 |

### 矩阵乘法api

|  |
| --- |
| np.matmul(a,b) #a,b矩阵相乘，不可以乘标量  np.dot(a,b) #a,b矩阵相乘，可以乘标量 |

### 其他性质

|  |
| --- |
|  |

# Pandas

## 介绍

|  |
| --- |
| * 2008年WesMcKinney开发出的库 * 专门用于数据挖掘的开源python库 * 以Numpy为基础，借力Numpy模块在计算方面性能高的优势 * 基于matplotlib，能够简便的画图 * 独特的数据结构 * 增强图表可读性 * 便捷的数据处理能力 * 读取文件方便 * 封装了Matplotlib、Numpy的画图和计算 |

### 通用接口

|  |
| --- |
| =========================================================  date = pandas.date\_range(start,end,periods,freq=’B’)  **#生成一串日期**  start: 开始日期20220403  end：结束日期20220407  periods：时间天数，跨度。end和periods选其一。  freq：B为略过周末  属性：  date.date # 返回日期  参考：https://blog.csdn.net/tz\_zs/article/details/80912959  data = pd.to\_datetime(data.index,unit=’s’)  #将索引的日期变成日期格式，方便直接处理日期的格式  data.weekday -- 返回每个索引日期是周几  data.day -- 返回索引日期的日  ========================================================  将另一个df对象里的时间转换为time对象后再实例化为一个datetime64对象  time = pd.DatetimeIndex(pd.to\_datetime(facebook['time'],unit='s')) |

## 数据结构

### Series（一维）

|  |
| --- |
| # 导入pandas  import pandas as pd  #date可以是ndarray、list、dict，dict会自动匹配索引  s = pd.Series(data=None, index=None, dtype=None)  s.index  s.values |

### DataFrame（二维）

#### 基本属性

|  |
| --- |
| C:\Users\liusx\AppData\Local\Temp\1648969728(1).png  # 导入pandas  import pandas as pd  #####################创建#########################  s = pd.DataFrame(data=None, index=None, columns=None)  参数：  index：行标签。如果没有传入索引参数，则默认会自动创建一个从0-N的整数索引。  columns：列标签。如果没有传入索引参数，则默认会自动创建一个从0-N的整数索引。  行标签和列标签可以用列表  ######################属性##############################  s.shape  s.size # 大小  s.index #行索引  s.columns  s.values  s.T #转置  s.head(5) #显示前5行  s.tail(5) #显示尾5行  ###################修改索引###############################  s\_index = ["学生\_" + str(i) for i in range(s.shape[0])]  # 必须整体全部修改  s.index = s\_index  ###################重置索引###############################  s.reset\_index(drop=False) #drop:默认为False，不删除原来索引  ##################以某列值重设索引#########################  s.set\_index(keys, drop=True)  keys : 列索引名 或者 列索引名称的列表 #当是列表时，相当于是3维数组  drop : 默认True.当做新的索引，删除原来的列  ##################追加和合并DataFrame######################### |

#### 创建DataFrame的几种入参类型

|  |
| --- |
| C:\Users\liusx\AppData\Local\Temp\WeChat Files\eb5dcc19a350fced5f0ecf2fa19e3f4.jpg  C:\Users\liusx\AppData\Local\Temp\WeChat Files\ad05fb07a0c8dc63e2a9dfa5d6b4cd0.jpg  C:\Users\liusx\AppData\Local\Temp\WeChat Files\bad6d9a14faa4c14c0ad5515933c81f.jpg |

### MultiIndex与Panel（三维）

|  |
| --- |
| 创建：  pd.MultiIndex.from\_arrays()  属性：  对象.index  对象.index.names  eg:  arrays = [[1, 1, 2, 2], ['red', 'blue', 'red', 'blue']]  pd.MultiIndex.from\_arrays(arrays, names=('number', 'color'))  # 结果  MultiIndex(levels=[[1, 2], ['blue', 'red']],  codes=[[0, 0, 1, 1], [1, 0, 1, 0]],  names=['number', 'color'])  创建：  pandas.Panel(data=None, items=None, major\_axis=None, minor\_axis=None)  参数：  data : ndarray或者dataframe  items : 索引或类似数组的对象，axis=0  major\_axis : 索引或类似数组的对象，axis=1  minor\_axis : 索引或类似数组的对象，axis=2  eg：  p = pd.Panel(data=np.arange(24).reshape(4,3,2),  items=list('ABCD'),  major\_axis=pd.date\_range('20130101', periods=3),  minor\_axis=['first', 'second']) |

## 基本数据操作

### 索引

|  |
| --- |
| # 读取文件  data = pd.read\_csv("./data/stock\_day.csv")  # 删除一些列，让数据更简单些，再去做后面的操作  data = data.drop(["ma5","ma10","ma20","v\_ma5","v\_ma10","v\_ma20"], axis=1)  data = head()  ======================直接索引=============================  data['open']['2018-02-27'] #先列后行  data[ ['open',’close’] ] # 取2列  #注意data[‘open’]和data[[‘open’]]不一样，前者只有值属于Series结构，后者有表头栏属于DataFranme结构  =========================loc和iloc==========================  # 使用loc:只能指定行列索引的名字  data.loc['2018-02-27':'2018-02-22', 'open']  # 使用iloc可以通过索引的下标去获取  # 获取前3天数据,前5列的结果  data.iloc[:3, :5]  =========================ix组合索引=========================  # 使用ix进行下表和名称组合做引  data.ix[0:4, ['open', 'close', 'high', 'low']    #获取行列索引  data.index.get\_indexer(['2018-02-23', '2018-02-22'])  data.columns.get\_indexer(['open', 'close', 'high', 'low']  # 推荐使用loc和iloc来获取的方式  data.loc[data.index[0:4], ['open', 'close', 'high', 'low']]  data.iloc[0:4, data.columns.get\_indexer(['open', 'close', 'high', 'low'])] |

### 赋值

|  |
| --- |
| data[‘open’] = data\_list #修改一列  data.insert(loc=len(df.columns), column='player', value=player\_vals) #在最后插入新列 |

### 排序

|  |
| --- |
| data = data.sort\_values(by=, ascending=) # 返回排序好的数据  参数：  by：指定排序参考的键,可以是列表多个值，当第一个值相等按列表第二个值排序  ascending:默认升序  ascending=False:降序  ascending=True:升序  =========================================================  data.sort\_index() # 按索引排序  =========================================================  series.sort\_values(ascending=True) #只有一列，不需要参数  series.sort\_index() |

### 筛选

|  |
| --- |
| 1. 筛选某列为某值   # 单值筛选  data = df[(df['列名']== 列值1)]  # 多值筛选  data\_many=df[(df['列名1']== 列值1)&(df['列名2']==列值2)]  data\_many1=df[(df['列名']== 19920812)|(df['date']==19920811)]  data\_many2=df[df['列名'].**isin**([19920807,19920814]) ]   1. 模式筛选   # 注意必须所有值都为str，可先转换data = data.astype(str)  # 开头包含某值的模式匹配  cond=df['列名'].str.startswith('值')  # 中间包含某值的模式匹配  cond=df['列名'].str.contains('值')   1. 范围筛选   # 筛选出基于两个值之间的数据：  cond=df[(df['列名1']>‘列值1’)&(df['列名1']<‘列值2’)]   1. 得到筛选的行索引   df = pd.DataFrame({'col1': [1, 2, 3, 3, 4], 'col2': [22, 33, 22, 44, 66]}, index=[1,2,3,4,5])  print(df)  a = df[(df.col1 == 3) & (df.col2 == 22)].index # .tolist(), .value, [0] ，三种方法取值 |

### 转换

|  |
| --- |
| df.value.tolist() # 二维转换为列表  df.columns.tolist() # 一维数组转换为列表  df = df.astype(str) # df所有元素转成str类型  pd.to\_datetime(df['date']) # 将某列转为时间格式 |

### 删除

|  |
| --- |
| # 使用的前提是，dataframe的index和columns用的是数字，利用了drop（）和range()函数。    DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')    # axis = 0，表示删除行； axis = 1 表示删除列。    # 想删除多行/列，用range即可，比如要删除前3行，drop(range(0,3)，axis = 0(默认为零，可不写))即可。 |

### 遍历

|  |
| --- |
| **import** pandas **as** pd  data = pd.read\_excel(**'18.excel-demo.xlsx'**)  data.iterrows() # 将数据按行迭代 data.to\_dict(**'dict'**) # 数据转字典，key为列名，v为字典{key为索引，v为值}  data.to\_dict(**'list'**) # 数据转字典，key为列名，v为列数据列表  data.to\_dict(**'records'**)# 每一行都形成字典，key为列名，v为单元值  data.to\_dict(**'records'**)#转字典，key为’index’,’columns’,’ data’,v为对应数据 |

### 替换

|  |
| --- |
| replace()方法的语法为：  DataFrame.replace(to\_replace=None,value=None,inplace=False, limit=None, regex=False, method='pad')。   * to\_replace - 要替换的值。可以是标量、字典、列表或序列。 * value - 替换值。可以是标量、字典、列表或序列。 * inplace - 可选参数。如果为 True，则在原地修改数据，而不返回新对象。 * limit - 可选参数。如果指定，则仅替换前 limit 个匹配项。 * regex - 可选参数。如果为 True，则将 to\_replace 视为正则表达式。 * method - 可选参数。指定替换方法。可以是 'pad'、'ffill'、'bfill' 或 None。   'pad' 或 'ffill' - 使用前一个值填充空缺。  ‘bfill' - 使用后一个值填充空缺。  None - 不使用任何填充方法。 |

### 其他

|  |
| --- |
| # 去重  data['Director'].unique()  pd.unique(data['Director'])  # 求次数  data['Genre'].value\_counts() # 统计每个出现的单个元素的次数  data['Genre'].value\_counts()  参数：  normalize：默认为 False。如果设置为 True，则返回每个唯一值出现的相对频率，而不是绝对频率。  sort：默认为 True。如果设置为 True，则按值出现的次数对结果进行排序。  ascending：默认为 False。如果设置为 True，则按升序对结果进行排序。  dropna：默认为 True。如果设置为 True，则不包括缺失值（NaN）。  # 列转换为行  data. stack()  # 返回最大值  idxmax() 函数可以用于查找 Series 中第一个缺失值的位置，因为它返回的是 Series 中第一个最大值的索引。当您在一个布尔值 Series 上调用 idxmax() 函数时，它将返回第一个 True 值的索引。  # any判断整个数据有没有缺失值. 有一个缺失值就有一个True，有True就返回True  data\_x.isnull().any()  # any判断某列有没有缺失值  data\_x.isnull().any()  # any判断某行有没有缺失值  data\_x.isnull().any(axis=1)  # 查看缺失值的位置  print(data\_y[data\_y.isnull()].index) # 查看第一个缺失值  print(list(data\_y.isnull().idxmax())) # 查看所有缺失值的位置 |

## DataFrame运算

### 算术运算

|  |
| --- |
| ========================加减乘除===========================  data['open'].add(1) #也可以data['open']+1 对应列全部+1  data['open'].sub(1) |

### 逻辑运算

|  |
| --- |
| ======================逻辑运算=============================  data["open"] > 23 # 返回索引和bool值  # 逻辑判断的结果可以作为筛选的依据  data[data["open"] > 23] # 返回bool为True的索引和值  data[ (data["open"] > 23) & (data["open"] < 24) ] #多个逻辑运算  ======================逻辑运算API===========================  data.query("open<24 & open>23") #返回int值  data["open"].isin([23.53, 23.85]) #返回bool值 |

### 统计运算

|  |
| --- |
| =========================统计函数=========================  min(最小值), max(最大值), mean(平均值), median(中位数), var(方差), std(标准差),mode(众数)，sum(求累计和)，abs(绝对值),prod(求累计乘), idxmax(计算最大值的索引),idxmin()  eg:  data.max(axis=0) # 求每一列的最大值  =======================累计统计函数=========================  cumsum 计算前1/2/3/…/n个数的和，即到对应索引的sum  cummax 计算前1/2/3/…/n个数的最大值  cummin 计算前1/2/3/…/n个数的最小值  cumprod 计算前1/2/3/…/n个数的积  data.cumsum()  eg:  import matplotlib.pyplot as plt  data['open’].cumsum()  **data.cumsum().plot()**  plt.show() |

### 自定义运算

|  |
| --- |
| DataFrame:  # 会对变量dataframe每个元素，对其使用func函数  apply(func, axis=0)  func:自定义函数  axis=0:默认是列，axis=1为行进行运算  eg:  data[['open', 'close']].apply(lambda x: x.max() - x.min(), axis=0) #求2列最大值最小值之差  eg:  def add\_extra2(nationaltiy, \*\*kwargs):  return kwargs.get(nationaltiy)    df['Extra'] = df.Nationality.apply(add\_extra2, 汉=0, 回=10, 藏=5) # 替换 |

|  |
| --- |
| Series:  # 对series对象的每个元素使用arg参数  data\_series.map(self, arg, na\_action=None)  arg:  可为字典，即将series的元素当作key入参，得到返回值  可为f’it is {}’,即将series的元素放到字符串中  课文其他函数  na\_action:  na\_action= ‘ignore’,则忽略缺失值 |

## Pandas画图

|  |
| --- |
| DataFrame.plot(kind='line', figsize=(20,8),fontsize=20,colormap="hot")  kind : str，需要绘制图形的种类  ‘line’ : 折线图 (default)  ‘bar’ : 柱状图  ‘barh’ : 水平条形图  ‘hist’ :直方图  ‘pie’ : 饼图  ‘scatter’ : 散点图  stacked：bool，是否堆积，默认False，适用于bar  colormap:调色板 |

## 文件读取与存储

|  |
| --- |
| C:\Users\liusx\AppData\Local\Temp\1649141807(1).png |

### CSV

|  |
| --- |
| =======================读取===============================  pandas.read\_csv(filepath\_or\_buffer, sep =',', usecols )  filepath\_or\_buffer:文件路径  sep :分隔符，默认用","隔开  usecols:指定读取的列名，列表形式  eg:  data = pd.read\_csv("./data/stock\_day.csv", usecols=['open', 'close'])  =======================储存===============================  data.to\_csv(path\_or\_buf=None, sep=', ’, columns=None, header=True, index=True, mode='w', encoding=None)  path\_or\_buf :文件路径  sep :分隔符，默认用","隔开  columns :选择需要的列索引  header :是否写入表头栏，默认True  index:是否写进行索引  mode:'w'：重写, 'a' 追加  eg:  data[:10].to\_csv("./data/test.csv", columns=['open']) |

### HDF5

|  |
| --- |
| **hdf中的key相当于是excel中的工作表，必须保存有工作表才能读取对应的工作表**  ==========================读取============================  pandas.read\_hdf(path\_or\_buf，key =None，\*\* kwargs)  path\_or\_buffer:文件路径  key:读取的键  return:Theselected object  eg:  data\_hdf = pd.read\_hdf("./data/day\_close.h5")  ========================写入==============================  DataFrame.to\_hdf(path\_or\_buf, key, \*\kwargs\*)  eg:  data\_hdf.to\_hdf("./data/test.h5", key="day\_close")  ======================= 优点==============================  注意：优先选择使用HDF5文件存储   * HDF5在存储的时候支持压缩，使用的方式是blosc，这个是速度最快的也是pandas默认支持的 * 使用压缩可以提磁盘利用率，节省空间 * HDF5还是跨平台的，可以轻松迁移到hadoop 上面 |

### JSON

|  |
| --- |
| ==========================读取============================  pandas.read\_json(path\_or\_buf=None, orient=None, typ='frame', lines=False)  将JSON格式准换成默认的Pandas DataFrame格式  orient : string,Indication of expected JSON string format.  'split' : dict like {index -> [index], columns -> [columns], data -> [values]}  split 将索引总结到索引，列名到列名，数据到数据。将三部分都分开了  'records' : list like [{column -> value}, ... , {column -> value}]  records 以columns：values的形式输出  'index' : dict like {index -> {column -> value}}  index 以index：{columns：values}...的形式输出  'columns' : dict like {column -> {index -> value}},默认该格式  colums 以columns:{index:values}的形式输出  'values' : just the values array  values 直接输出值  lines : boolean, default False，按照每行读取json对象  typ : default ‘frame’， 指定转换成的对象类型series或者dataframe  eg:  json\_read = pd.read\_json("./data/Sarcasm\_Headlines\_Dataset.json", orient="records", lines=True)  ========================存储==============================  json\_read.to\_json("./data/test.json", orient='records', lines=True) |

### EXCEL

|  |
| --- |
| sheet\_name：str、int、list 或 None，默认为 0。定义要从 excel 文件中读取的工作表。  header：int 或 int 的列表，默认为 0。如果未指定此参数，它将默认为 0。excel 文件中的第一行将被视为 Headers。  names：array-like，默认为 None。这个参数用于结果的列名列表。如果文件中没有列名，则应该显式传递列名。  index\_col：int 或 int 的列表，默认为 None。这个参数用于定义一个或多个列号（0 是第一列）、列名，作为返回的 DataFrame 的索引。  usecols：int、str、类似列表的或可调用的，默认为 None。返回一个数据子集，即索引值或列标签。  dtype：类型名称或列 -> 类型的 dict，默认为 None。指定返回 DataFrame 的数据类型。  engine：None 或 str，默认为 None。如果 io 不是缓冲区或路径，则必须设置此选项来识别 io。接受的值包括：None, ‘xlrd’, ‘openpyxl’, ‘odf’, ‘pyxlsb’。  converters：dict，默认为 None。将函数应用于值序列以生成转换后的值序列，而不是转换整个序列一次。  true\_values：list，默认为 None。这些值将被解释为 True。  false\_values：list，默认为 None。这些值将被解释为 False。  skiprows：list-like 或 int 或可调用的，默认为 None。需要跳过的行号（从 0 开始）或需要跳过的行数。  nrows：int，默认为 None。需要读取的行数（从文件头开始）。  na\_values：scalar, str, list-like, or dict, 默认为 None。需要替换成 NaN 的附加值。  keep\_default\_na：bool，默认为 True。无论 na\_filter 如何设置，都保留默认的 NaN 值。  na\_filter：bool，默认为 True。检测丢失的值标记（空字符串和 sentinel 值）。在数据很大时禁用可以提高性能  verbose：bool，默认为 False。打印各种解析器输出信息，如“正在处理 x 行”等。  parse\_dates：bool 或 list of ints or names or list of lists or dict, 默认为 False。尝试将数据解析为日期。  date\_parser：function, optional。用于解析日期的函数。  thousands：str, 默认为 None。千位分隔符，例如 ‘,’ 或 ‘.’。  comment：str, 默认为 None。注释开始标记，所有在其后面且在行尾的数据都会被忽略。  skipfooter：int, 默认为 0。需要忽略的行尾行数（从文件尾开始计数）。  convert\_float：bool, 默认为 True。将整数浮点值转换回浮点表示。 |

|  |
| --- |
| # 单sheet写入  import pandas as pd import random import numpy as np from openpyxl import load\_workbook  data = np.random.randint(10, 30, 100).reshape(10, 10) pd\_data\_3 = pd.DataFrame(data, index=pd.date\_range('2022-03-01', periods=10).date,  columns=[f'{i}时' for i in range(10)]) pd\_data\_3.to\_excel('test.xlsx', sheet\_name='3月', header=False, index=False, na\_rep='NA', startcol=1, startrow=1) # header:是否写入列索引 # index:是否写入行索引 # na\_rep:缺失值写入为 # startcol:开始列 # startrow:开始行  # dtype:读取的数据类型。| str | dtype | Type[str] | Type[float] | Type[int] | Type[complex] | Type[bool] | Type[object] | dict[Hasha]  # engine=’xlsxwriter’/’openpxl’ 前者适用于大文件  # 多sheet写入  with pd.ExcelWriter('test.xlsx') as writer:  pd\_data\_3.to\_excel(writer, sheet\_name='Sheet1', index=False)  pd\_data\_3.to\_excel(writer, sheet\_name='Sheet2', index=False)  # 新增sheet，不覆盖已存在sheet  with pd.ExcelWriter('test.xlsx', mode='a') as writer:  pd\_data\_3.to\_excel(writer, sheet\_name='Sheet3', index=False)  # 修改sheet内容，不覆盖已存在sheet  # 根据DataFrame的入参类型修改行列，startrow确定位置  book = load\_workbook('test.xlsx') pd\_data\_2 = pd.DataFrame([[99, 99, 99]]) with pd.ExcelWriter('test.xlsx') as writer:  writer.book = book # 读取excel  writer.sheets = dict((ws.title, ws) for ws in book.worksheets) # 复制excel的所有表  pd\_data\_2.to\_excel(writer, sheet\_name='Sheet1', header=False, index=False,startrow=4) |

## 缺失值处理

|  |
| --- |
| ======================判断是否有缺失值=======================  pd.isnull(data),  pd.notnull(data)  # np.all(pd.notnull(movie\_var))  # any判断整个数据有没有缺失值. 有一个缺失值就有一个True，有True就返回True  data\_x.isnull().any()  # any判断某列有没有缺失值  data\_x.isnull().any()  # any判断某行有没有缺失值  data\_x.isnull().any(axis=1)  # 查看缺失值的位置  print(data\_y[data\_y.isnull()].index) # 查看第一个缺失值  print(list(data\_y.isnull().idxmax())) # 查看所有缺失值的位置  ========================删除缺失值==========================  data = movie\_var.dropna()  # 找出不是缺失值的索引重新赋值  # 删除缺失值  mask = y.notnull().all(axis=1)  x = x[mask]  y = y[mask]  ========================替换缺失值==========================  # 将dataframe变量的'Revenue (Millions)'列的缺失值替换为列的平均值  movie\_var['Revenue (Millions)'].fillna(movie['Revenue (Millions)'].mean(), inplace=True)  for i in movie.columns:  if np.all(pd.notnull(movie[i])) == False:  print(i)  movie[i].fillna(movie[i].mean(), inplace=True)  ========================缺失值是‘？‘=========================  data = data.replace(to\_replace='?', value=np.nan) #先替换成NAN  data = data.dropna()  ===================fillna()方法===============================  fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=None, \*\*kwargs)  value：用于填充的空值的值。  method： {'backfill', 'bfill', 'pad', 'ffill', None}, default None。定义了填充空值的方法， pad / ffill表示用前面行/列的值，填充当前行/列的空值， backfill / bfill表示用后面行/列的值，填充当前行/列的空值。  axis：轴。0或'index'，表示按行删除；1或'columns'，表示按列删除。  inplace：是否原地替换。布尔值，默认为False。如果为True，则在原DataFrame上进行操作，返回值为None。  limit：int， default None。如果method被指定，对于连续的空值，这段连续区域，最多填充前 limit 个空值（如果存在多段连续区域，每段最多填充前 limit 个空值）。如果method未被指定， 在该axis下，最多填充前 limit 个空值（不论空值连续区间是否间断）  downcast：dict, default is None，字典中的项为，为类型向下转换规则。或者为字符串“infer”，此时会在合适的等价类型之间进行向下转换，比如float64 to int64 if possible。  实例：  # 填补读取合并单元格的时候的空值  df.fillna(method=’pad’,inplace=True)  df ['age'].fillna(df ['age'].mean(), inplace=True) |

## 数据离散化

### 什么是数据离散化

将一堆数据分区间，同时可以统计每个区间的数据个数，最终转换成计算机容易识别的哑变量格式

### one-hot编码

把每个类别生成一个布尔列，这些列中只有一列可以为这个样本取值为1.其又被称为热编码。也是哑变量矩阵。

### API接口

|  |
| --- |
| import pandas as pd  data = pd.read\_csv('./date/stock\_day.csv')  p\_change= data['p\_change']  #将数据分成5个区间  qcut = pd.qcut(p\_change,5)  #显示每个区间的数据个数  qcut.value\_counts()  #将数据分成自定义的区间  cut = pd.cut(p\_change,bins=[-11,-2,0,1,5])  cut.value\_counts()  #将数据转成one-hot编码  dummies = pd.get\_dummies(qcut, prefix="test") |

## 数据合并

### pd.concat

|  |
| --- |
| pd.concat([data1, data2], axis=1)  按照行或列进行合并,axis=0为列索引，axis=1为行索引。将data2的数据根据行或列索引添加到data1houm |

### pd.merge

|  |
| --- |
| pd.merge(left, right, how='inner', on=None)  -  left：左侧的DataFrame，必需参数。 -  right：右侧的DataFrame，必需参数。 -  how：合并方式，可以取值为'left'、'right'、'outer'、'inner'，默认为'inner'。其中：         -  'left'：以左侧DataFrame为基准，按照key进行左连接；         -  'right'：以右侧DataFrame为基准，按照key进行右连接；         -  'outer'：按照key进行外连接，可以把所有的DataFrame都合并在一起；         -  'inner'：按照key进行内连接，只合并两个DataFrame中都存在的key所在的行； -  on：合并的key列，可以是一个列表或单个列名，必需参数。 -  left\_on：左侧DataFrame中用于合并的key列，默认与on参数一致，可选。 -  right\_on：右侧DataFrame中用于合并的key列，默认与on参数一致，可选。 -  suffixes：字符串列表，用于区分重名列的后缀，默认为('\_x',  '\_y')，可选。 -  sort：是否按照key进行排序，默认为True，可选。  eg:  left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],  'key2': ['K0', 'K1', 'K0', 'K1'],  'A': ['A0', 'A1', 'A2', 'A3'],  'B': ['B0', 'B1', 'B2', 'B3']})  right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],  'key2': ['K0', 'K0', 'K0', 'K0'],  'C': ['C0', 'C1', 'C2', 'C3'],  'D': ['D0', 'D1', 'D2', 'D3']})  C:\Users\liusx\AppData\Local\Temp\1649673211(1).png  C:\Users\liusx\AppData\Local\Temp\1649673237(1).png  C:\Users\liusx\AppData\Local\Temp\1649673258(1).png  C:\Users\liusx\AppData\Local\Temp\1649673276(1).png |

## 交叉表和透视表

### 介绍

交叉表：交叉表用于计算**一列数据**对于**另外一列**数据的**不同分组**的**个数**(用于统计分组频率的特殊透视表) -- pd.crosstab(value1, value2)

透视表：透视表是将原有的DataFrame的列分别作为行索引和列索引，然后对指定的列应用聚集函数.**透视表在交叉表的基础上再进行数据处理**,比如求平均值等 -- data.pivot\_table(['posi\_neg'], index='week'）

### API接口

|  |
| --- |
| ===================**交叉表（涨跌与周几的关系）**===================  #数据准备  import pandas as pd  data = pd.read\_csv('./date/stock\_day.csv')  #获取索引日期的datatime格式  data\_index = pd.to\_datetime(data.index)  #获取周几  data\_week = data\_index.weekday  #数据添加  data['week'] = data\_week  #数据添加  data['posi\_neg'] = np.where(data['p\_change'] > 0, 1, 0)  #通过交叉表找寻两列数据的关系  count = pd.crosstab(data['week'], data['posi\_neg'])  #算数运算，先求和  sum = count.sum(axis=1).astype(np.float32)  #进行相除操作，得出比例  pro = count.div(sum, axis=0)  #绘图  pro.plot(kind='bar', stacked=True)  plt.show()  =====================接口参数==============  pandas里crosstab接口参数的说明如下：  - index：定义每个交叉表的行索引  - columns：定义每个交叉表的列索引  - values：可选，指定了计算交叉表的值列  - aggfunc：可选，指定了聚合函数，缺省为计数  - rownames：可选，指定了每行的名称  - colnames：可选，指定了每列的名称  - margins：可选，默认为False，若为True，则在交叉表中添加边际和列  - normalize：可选，默认为False，如果为True，则计算每个分组的百分比  ========================**透视表**==========================  #通过透视表找关系  pro2 = data.pivot\_table(index='week', values=[ 'posi\_neg'], aggfunc='mean')  #绘图  pro2.plot(kind='bar', stacked=True)  plt.show()  #################### pd.pivot\_table参数###################  1. data：指定数据集。  2. values：指定需要聚合的列。  3. index：指定行标签。  4. columns：指定列标签。  5. aggfunc：指定聚合方法。默认是平均值mean。  6. fill\_value：指定空值填充的值。  7. margins：是否计算行列的总和。默认为False。  8. dropna：是否删除空值行。默认为True。  9. margins\_name：指定总和列的列名。默认为All。  10. observed：是否只计算观察到的数据。默认为False。  11. margins\_name：指定总和列的列名。默认为All。  12. sort：是否对行列标签进行排序。默认为True。  13. col\_order，row\_order：手动指定列标签或行标签的排序顺序。  14. ffill：透视表填充时的向前填充方法。  15. drop\_duplicates：是否删除重复项。  16. aggfunc\_kwargs：参数传递aggfunc的关键字参数 |

## 分组与聚合

|  |
| --- |
| 简介：**分组**：将数据按**行索引分类**；**聚合**：**再选择列,筛选此列不同类的数量或均值.**等对同类的数据操作，如求均值  API：  DataFrame.groupby(by, as\_index=False)[‘某列’].mean()  #不进行聚合则无意义；by可以是列表，进行多重分类再聚合  常用参数:  by，分组字段，可以是列名/series/字典/函数，常用为列名  axis，指定切分方向，默认为0，表示沿着行切  as\_index，是否将分组**列名作为**输出的**索引**，默认为True；当设置为False时相当于加了reset\_index功能  sort，与SQL中groupby操作会默认执行排序一致，该groupby也可通过sort参数指定是否**对**输出结果(**分组**)按索引**排序**  eg：  col=pd.DataFrame({'color':['white','red','green','red','green'],'object':['pen','pencil','pencil','ashtray','pen'],'price1':[5.56,4.20,1.30,0.56,2.75],'price2':[4.75,4.12,1.60,0.75,3.15]})  col.groupby(['color'])['price1'].mean()  data.groupby('Director').all() # 查看所有分组结果 |

## 操作数据库

### Mongodb

|  |
| --- |
| pip install pymongo  import pymongo  # 连接 MongoDB 数据库  client = pymongo.MongoClient("mongodb://localhost:27017/")  # 选择数据库和集合  db = client["mydatabase"]  collection = db["mycollection"]  # 查询数据  results = collection.find({"name": "John"})  for result in results:  print(result) |

# K-邻近算法

## 简介

|  |
| --- |
| 定义：  如果一个样本在特征空间中的k个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。  C:\Users\liusx\AppData\Local\Temp\1650093350(1).png |

## API

|  |
| --- |
| 1. **Scikit-learn包含的内容：**   Classification：分类  Regression：回归  Clustering：聚类  Dimensionality reduction:降维  Model selection:  Preprocessing:预处理   1. **K-近邻算法API**   sklearn.neighbors.KNeighborsClassifier(n\_neighbors=5, algorithm='auto')  n\_neighbors：int,可选（默认= 5），k\_neighbors查询默认使用的邻居数  algorithm：{‘auto’，‘ball\_tree’，‘kd\_tree’，‘brute’}  快速k近邻搜索算法，默认auto，维度大于20用ball\_tree,维度小用brute,中间用ke\_tree.  brute是蛮力搜索，也就是线性扫描，当训练集很大时，计算非常耗时。  kd\_tree，构造kd树存储数据以便对其进行快速检索的树形数据结构，kd树也就是数据结构中的二叉树。以中值切分构造的树，每个结点是一个超矩形，在维数小于20时效率高。  ball tree是为了克服kd树高纬失效而发明的，其构造过程是以质心C和半径r分割样本空间，每个节点是一个超球体。   1. **案例**   **3.1 步骤分析**  1.获取数据集  2.数据基本处理（该案例中省略）  3.特征工程（该案例中省略）  4.机器学习  5.模型评估（该案例中省略）  **3.2 代码过程**  from sklearn.neighbors import KneighborsClassifier  #准备数据  x = [[-2],[-1],[1],[2]]  y = [0,0,1,1]  # 实例化API  estimator = KNeighborsClassifier(n\_neighbors=3)  # 使用fit方法进行训练  estimator.fit(x, y)  #预测新数据  ret = estimator.predict([[-10]])  print(ret) |

## 距离度量

### 欧氏距离

|  |
| --- |
|  |

### 曼哈顿距离

|  |
| --- |
|  |

### 切比雪夫距离

|  |
| --- |
|  |

### 闵可夫斯基距离

|  |
| --- |
|  |

### 标准化欧氏距离

|  |
| --- |
| C:\Users\liusx\AppData\Local\Temp\1650186287(1).png |

### 余弦距离

|  |
| --- |
| C:\Users\liusx\AppData\Local\Temp\1650190296(1).png |

### 汉明距离（了解）

|  |
| --- |
| 定义：两个等长字符串s1与s2的汉明距离为：将其中一个变为另外一个所需要作的最小字符替换次数。  eg：  C:\Users\liusx\AppData\Local\Temp\1650201886(1).png  **汉明重量**：是字符串相对于同样长度的零字符串的汉明距离，也就是说，它是字符串中非零的元素个数：对于二进制字符串来说，就是 1 的个数，所以 11101 的汉明重量是 4。因此如果向量空间中元素a和b之间的汉明距离等于它们汉明重量的差a-b。  **应用**：汉明重量分析在包括信息论、编码理论、密码学等领域都有应用。比如在信息编码过程中，为了增强容错性，应使得编码间的最小汉明距离尽可能大。但是，如果要比较两个不同长度的字符串，不仅要进行替换，而且要进行插入与删除的运算，在这种场合下，通常使用更加复杂的编辑距离等算法。 |

### 杰卡德距离（了解）

|  |
| --- |
| 杰卡德相似系数(Jaccard similarity coefficient)：两个集合A和B的交集元素在A，B的并集中所占的比例，称为两个集合的杰卡德相似系数，用符号J(A,B)表示：  C:\Users\liusx\AppData\Local\Temp\1650202656(1).png  杰卡德距离(Jaccard Distance)：与杰卡德相似系数相反，用两个集合中不同元素占所有元素的比例来衡量两个集合的区分度：  C:\Users\liusx\AppData\Local\Temp\1650202700(1).png  例子：  C:\Users\liusx\AppData\Local\Temp\1650202731(1).png |

### 马氏距离（了解）

|  |
| --- |
| **马氏距离**是基于样本分布的一种距离。  马氏距离是由印度统计学家马哈拉诺比斯提出的，表示数据的协方差距离。它是一种有效的计算两个位置样本集的相似度的方法。  与欧式距离不同的是，它考虑到各种特性之间的联系，即独立于测量尺度。  **马氏距离定义**：设总体G为m维总体（考察m个指标），均值向量为μ=（μ1，μ2，… ...，μm，）`,协方差阵为∑=（σij）,  则样本X=（X1，X2，… …，Xm，）`与总体G的马氏距离定义为：  C:\Users\liusx\AppData\Local\Temp\1650202873(1).png  马氏距离也可以定义为两个服从同一分布并且其协方差矩阵为∑的随机变量的差异程度：如果协方差矩阵为单位矩阵，马氏距离就简化为欧式距离；如果协方差矩阵为对角矩阵，则其也可称为正规化的欧式距离。  **马氏距离特性：**  1.量纲无关，排除变量之间的相关性的干扰；  2.马氏距离的计算是建立在总体样本的基础上的，如果拿同样的两个样本，放入两个不同的总体中，最后计算得出的两个样本间的马氏距离通常是不相同的，除非这两个总体的协方差矩阵碰巧相同；  3 .计算马氏距离过程中，要求总体样本数大于样本的维数，否则得到的总体样本协方差矩阵逆矩阵不存在，这种情况下，用欧式距离计算即可。  4.还有一种情况，满足了条件总体样本数大于样本的维数，但是协方差矩阵的逆矩阵仍然不存在，比如三个样本点（3，4），（5，6），（7，8），这种情况是因为这三个样本在其所处的二维空间平面内共线。这种情况下，也采用欧式距离计算。 |

## K值的选择

**K过小：**

如果刚好是异常点就预测错误，所以容易受到异常点影响

**K过大：**

代表匹配的距离越远，越不准确，容易受到样本均衡性的影响

**近似误差**：对现有训练集的训练误差，关注训练集的误差：

误差大则测不准，欠拟合

误差小则特征精确，测试集稍有不慎就判断错误，过拟合

**估计误差：**对测试集的测试误差

误差小，说明预测能力好

选择小K值，近似误差会减小，估计误差会变大，容易发生判断出错的情况，过拟合

选择大K值，近似误差会变大，估计误差会变小，会使一些不正确的目标也进入K的范围，使预测发送错误，模型变简单。

## kd树

### 概念和意义

|  |
| --- |
| 意义：降低距离计算时的计算量，避免不必要的计算  原理：如果A和B距离很远，B和C距离很近，那么A和C的距离也很远。有了这个信息，就可以在合适的时候跳过距离远的点。  本质：按距离排序的平衡二叉树，树的深度越深距离越远 |

### 树的建立

|  |
| --- |
| 1. 选取数据所有维度中最分散的那一维开始，即方差最大的 2. 按选取维度的数据小到大排序，取中位数平分为两个子树 3. 取另一维排序，再取中位数，依次往后 |

### 最近领域搜索

|  |
| --- |
| 原理：找到最近的点，  步骤：  1、先从最远的点开始比较，树的根节点是按哪一维往下分，就按哪一维比较。小于进入左子树，大于进入右子树。  2、到达最后一个子树时计算和最后一个点的距离，以此距离画圆，得到候选超球。再回溯到上一个节点，看和这节点的维度轴是否有交集（上一个节点划分依据的维度），有交集则代表轴的上分可能有点在候选超球内，则最近距离需要和这个节点的另一个子树相比较。 |

## 案例：鸢尾花种类预测(sklearn库)

### sklearn数据集

#### 接口API

|  |
| --- |
| sklearn.datasets.  加载获取流行数据集  datasets.load\_\*()  获取小规模数据集，数据包含在datasets里  datasets.fetch\_\*(data\_home=None)  获取大规模数据集，需要从网络上下载，函数的第一个参数是data\_home，表示数据集下载的目录,默认是 ~/scikit\_learn\_data/  sklearn.datasets.fetch\_20newsgroups(data\_home=None,subset=‘train’)  subset：'train'或者'test'，'all'，可选，选择要加载的数据集。  训练集的“训练”，测试集的“测试”，两者的“全部” |

#### 返回值介绍

|  |
| --- |
| from sklearn.datasets import load\_iris, fetch\_20newsgroups  iris = load\_iris() # 返回值是一个继承自字典的Bench print("鸢尾花数据集的返回值：\n", iris) print("鸢尾花的特征值:\n", iris["data"]) # 特征值数据，是 [n\_samples \* n\_features] 的二维 numpy.ndarray 数组 print("鸢尾花的目标值：\n", iris.target) # 目标值数据，是 [n\_samples \* n\_features] 的二维 numpy.ndarray 数组 print("鸢尾花特征的名字：\n", iris.feature\_names) # 特征名,新闻数据，手写数字、回归数据集没有 print("鸢尾花目标值的名字：\n", iris.target\_names) # 目标值名 print("鸢尾花的描述：\n", iris.DESCR) # 数据描述 |

### 查看数据分布(seaborn)

|  |
| --- |
| from sklearn.datasets import load\_iris import matplotlib.pyplot as plt import pandas as pd import seaborn as sn   def polt\_iris(pd\_df, df\_col1\_name, df\_col2\_name):  sn.lmplot(x=df\_col1\_name, y=df\_col2\_name, data=pd\_df, fit\_reg=True, hue='鸢尾花种类')  # data- df数据类型  # x - 图坐标x的数据来源df的key  # y - 图坐标x的数据来源df的key  # hue - 图坐标的数据按df的此key分类为不同颜色  # fit\_reg - 是否进行线性拟合  plt.xlabel(df\_col1\_name)  plt.ylabel(df\_col2\_name)  plt.title(f'鸢尾花的{df\_col1\_name}和{df\_col2\_name}坐标图')  # 标题  plt.show()   iris = load\_iris() iris\_pd = pd.DataFrame(iris.data, columns=iris.feature\_names) print(iris.target\_names) iris\_pd['鸢尾花种类'] = iris.target  polt\_iris(iris\_pd, iris.feature\_names[0], iris.feature\_names[1]) |

### 数据划分

|  |
| --- |
| feature\_train\_data, feature\_test\_data, target\_train\_data, target\_test\_data = train\_test\_split(iris.data, iris.target,test\_size=0.2, random\_state=2) # 入参特征值、目标值 # test\_size:测试集占比 # random\_state:随机数种子，入参则每次返回一致，不入参则每次返回随机 print('特征值-训练集 ', feature\_train\_data) print('特征值-测试集 ', feature\_test\_data) print('目标值-训练集 ', target\_train\_data) print('目标值-测试集 ', target\_test\_data) |

### 特征预处理

#### 定义

将各个特征值转换成相同的量级便于算法模型使用

#### 归一化

|  |
| --- |
| 受异常点影响大，适用于传统规范的数据，一般不用  from sklearn.datasets import load\_iris, fetch\_20newsgroups import pandas as pd from sklearn.preprocessing import MinMaxScaler  iris = load\_iris() iris\_pd = pd.DataFrame(iris["data"], columns=iris.feature\_names) # 归一化 transfer = MinMaxScaler(feature\_range=(0, 1)) iris\_pd\_processed = transfer.fit\_transform(iris\_pd[['sepal length (cm)', 'sepal width (cm)']]) print(iris\_pd\_processed) |

#### 标准化

|  |
| --- |
| # 标准化 from sklearn.preprocessing import StandardScaler import pandas as pd from sklearn.datasets import load\_iris  iris = load\_iris() iris\_pd = pd.DataFrame(iris["data"], columns=iris.feature\_names) # 实例化转换器 transfer = StandardScaler() iris\_pd\_processed = transfer.fit\_transform(iris\_pd[['sepal length (cm)', 'sepal width (cm)']]) print(iris\_pd\_processed) |

### 训练数据

|  |
| --- |
| 实例化训练器接口查看 八.2.2)章节  # 实例化训练器 estimator = KNeighborsClassifier(n\_neighbors=5, algorithm='auto') estimator.fit(feature\_data\_train, target\_data\_train) |

### 预测数据

|  |
| --- |
| # 预测计算-测试集特征值的目标值  target\_data\_predict = estimator.predict(feature\_data\_test)  # 获取当前估计器测试集准确率，此时训练集的准确率是未知的  score = estimator.score(feature\_data\_test, target\_data\_test)  # score是模型测试集的预测目标值对比测试集的真实目标值的准确率。入参为测试集的特征值和测试集的目标值，调用score函数相当于对测试集做了一次预测然后对比。  # 使用交叉验证可得到平均值下的训练集准确率 |

### 交叉验证和网格搜索

|  |
| --- |
| **交叉验证：**  作用：  为了让被评估的模型更加准确可信，模型说预测准确率是多少，测试集预测的准确率就是多少，因为测试集只使用1次。  做法：  将训练集分为N折(一折就是一份)，其中1折为验证集，其他为训练集。不断更改验证集的位置，得到不同的准确率，最后取平均值。  **网格搜索：**  作用：选出最好的估计器参数，比如K值，称为超参数。  做法：取不同的超参数进行交叉验证，得到每个超参数其平均准确率，再选出最优超参数。  **API:**  from sklearn.model\_selection import train\_test\_split, GridSearchCV  sklearn.model\_selection.GridSearchCV(estimator, param\_grid=None,cv=None)  入参：  estimator：估计器对象  param\_grid：估计器参数。为字典。key为估计器的超参数名称，value为超参数列表值。  cv：几折交叉验证  n\_jobs：用多少cpu的性能计算，-1为满负荷  返回：  新的估计器对象，fit()训练后有如下结果：  best\_score\_：在交叉验证中最好的结果  best\_estimator\_：最好的参数模型，是一个有参数的估计器对象  cv\_results\_：每次交叉验证后的验证集准确率结果和训练集准确率结果 |

### 综合实现

|  |
| --- |
| **流程：**   1. 获取数据 2. 划分训练集和测试集 3. 标准化训练集和测试集的特征值 4. 训练数据 5. 预测测试集的目标值 6. 测试集的准确度 7. 交叉验证和网格搜索 8. 交叉验证：分为n折，得到一个k下训练集平均准确度 9. 网格搜索：得到不同k的准确度，选取最高准确度的k 10. 用最好的k模型训练数据，看测试集的准确度   **代码：**  from sklearn.datasets import load\_iris from sklearn.model\_selection import train\_test\_split, GridSearchCV from sklearn.preprocessing import StandardScaler from sklearn.neighbors import KNeighborsClassifier  # 1、获取数据 iris = load\_iris() target\_data = iris.target feature\_data = iris.data # print('目标值数据\n', target\_data) # print('特征数据\n', feature\_data)  # 2、数据分割测试集和训练集 feature\_data\_train, feature\_data\_test, target\_data\_train, target\_data\_test = train\_test\_split(feature\_data, target\_data,  test\_size=0.2) # print('训练集特征数据：\n', feature\_data\_train) # print('训练集目标数据：\n', target\_data\_train) # print('测试集特征数据：\n', feature\_data\_test) # print('测试集目标数据：\n', target\_data\_test)  # 3、特征预处理，标准化训练集和测试集的特征值 # 实例化转换器 transfer = StandardScaler() feature\_data\_train = transfer.fit\_transform(feature\_data\_train) feature\_data\_test = transfer.fit\_transform(feature\_data\_test) # print('标准化后的特征值训练数据：\n', feature\_data\_train) # print('标准化后的特征值测试数据：\n', feature\_data\_test)  # 4、训练数据 # 实例化估计器 estimator = KNeighborsClassifier(n\_neighbors=5, algorithm='auto') estimator.fit(feature\_data\_train, target\_data\_train)  # 预测计算测试集的特征值的目标值 target\_data\_predict = estimator.predict(feature\_data\_test) # print('预测的目标值：\n', target\_data\_predict) # print('实际的目标值：\n', target\_data\_test) # print('对比的目标值：\n', target\_data\_predict == target\_data\_test)  # 预测测试集，并返回测试集的准确率 score = estimator.score(feature\_data\_test, target\_data\_test) print('k=5的测试集准确度:\n', score)  # 5、交叉验证和网格搜索 # 将估计器交叉验证实例化 k = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] cv = 20 estimator\_cv = GridSearchCV(estimator, param\_grid={'n\_neighbors': k}, cv=cv) estimator\_cv.fit(feature\_data\_train, target\_data\_train)  # cv估计器的结果 print(f'k={k},cv={cv}时交叉验证的训练集最高准确度:\n', estimator\_cv.best\_score\_) # print(f'k={k},cv={cv}时交叉验证的最好的参数模型:\n', estimator\_cv.best\_estimator\_) print(f'k={k},cv={cv}时交叉验证的最好的参数模型的k值:\n', k\_best := estimator\_cv.best\_estimator\_.n\_neighbors) # print(f'每次交叉验证后的准确率结果:\n', estimator\_cv.cv\_results\_)  # 6、用最好的模型训练数据 # 实例化估计器 estimator\_best = KNeighborsClassifier(n\_neighbors=k\_best) estimator\_best.fit(feature\_data\_train, target\_data\_train) # 预测测试集，并返回测试集的准确率 score\_best = estimator\_best.score(feature\_data\_test, target\_data\_test) print('最好的模型测试集的准确度:\n', score\_best) |

## KNN算法优缺点

### 优点

1. **简单有效**
2. **重新训练的代价低**
3. **适合类域交叉样本**

KNN方法主要靠周围有限的邻近的样本,而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN方法较其他方法更为适合。

1. **适合大样本自动分类**

该算法比较适用于样本容量比较大的类域的自动分类，而那些样本容量较小的类域采用这种算法比较容易产生误分。

### 缺点

1. **惰性学习**

KNN算法是懒散学习方法（lazy learning,基本上不学习），一些积极学习的算法快很多

1. **类别评分不是规格化**

不像一些通过概率评分的分类

1. **输出可解释性不强**

例如决策树的输出可解释性就较强

1. **对不均衡的样本不擅长**

当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的K个邻居中大容量类的样本占多数。该算法只计算“最近的”邻居样本，某一类的样本数量很大，那么或者这类样本并不接近目标样本，或者这类样本很靠近目标样本。无论怎样，数量并不能影响运行结果。可以采用权值的方法（和该样本距离小的邻居权值大）来改进。

1. **计算量较大**

目前常用的解决方法是事先对已知样本点进行剪辑，事先去除对分类作用不大的样本。

## 个人总结

|  |
| --- |
| 通过求解预测值最近的点的距离，最近的K个点为某个类别，则预测为某个类别。  过程：  将数据集二分为kd树，时间复杂度n。  （到左侧结点近，则到左侧所有子节点都近。）  通过最近邻域搜索，每次只判断一维的距离大小，直到找到最小距离点，再返回找上一结点。以最小节点到上一结点距离做超平面，看是否和当前节点划分维度的轴相交。不相交则找到了，相交则要去另一子节点找。 |

# 线性回归

## 简介

### 应用场景理解

* 房价预测：假设有N个房屋的信息（售价、面积等），一个房屋信息就是一个**样本。**一个房屋的一个信息为一个**变量（特征值）。**线性回归为求这些变量和售价直接的**关系**。这个关系不是固定的，比如3个样本3个变量就能得到固定的关系。它是需要从成千上万的样本中统计出来，这个过程叫**线性回归**。
* 损失函数：假设线性回归的函数为h(x)=w1x1+w2x2+...，通过那么房价预测的**正确率**可以列为每个样本的代入到预测函数h(x)中得到**预测值h(x1)**，预测值减去真实值y1，如此累加得到了**损失函数**。它的形式为(h(x1)-y1)2+(h(x1)-y1)2+...，简化后为wi和xi的一个函数，此时只需求这个函数的最小值。
* 梯度下降：先对这个损失函数的所有参数w求偏导，首先参数定义一个默认值(**起点**)，代入一个样本后更新参数。如此循环。最后参数后收敛。

### 定义与公式

|  |
| --- |
| 线性回归(Linear regression)是利用回归方程(函数)对一个或多个自变量(特征值)和因变量(目标值)之间关系进行建模的一种分析方式。  特点：只有一个自变量的情况称为单变量回归，多于一个自变量情况的叫做多元回归    线性关系多元一次方程  非线性关系多元多次方程 |

### 梯度的概念

梯度是一个向量，它指向函数增长最快的方向。在数学中，如果一个多元函数具有一阶连续偏导数，那么这个函数的梯度就是由它的所有偏导数组成的向量。

如：

函数 f(x,y)=x^2 +y^3+xy，我们可以分别计算它的偏导数。函数 f 对 x 的偏导数为 2x+y，对 y 的偏导数为 3y^2+x。因此，函数 f 的梯度为 (2x+y)i + (3y^2+x)j。【i 和 j 是单位向量。在二维空间中，i 和 j 分别表示 x 轴和 y 轴的方向。例如，向量 (2x+y)i 表示在 x 轴方向上的分量为 2x+y，而 (3y^2+x)j 表示在 y 轴方向上的分量为 3y^2+x。】

### 偏置

在机器学习中，模型的偏置 (Bias) 是指模型对真实数据的拟合程度。如果模型的偏置很高，那么它可能无法很好地拟合数据，即使在训练集上也表现不佳。这种情况通常称为欠拟合 (Underfitting)。相反，如果模型的偏置很低，那么它可能会过度拟合数据，在训练集上表现得非常好，但在测试集上表现不佳。这种情况通常称为过拟合 (Overfitting)。

## api

### api

|  |
| --- |
| 正规方程和随机梯度下降： |

### 实例

|  |
| --- |
| 正规方程：  1.获取数据集  2.数据基本处理（该案例中省略）  3.特征工程（该案例中省略）  4.机器学习  5.模型评估（该案例中省略）  from sklearn.linear\_model import LinearRegression  # 构造数据集 x = [[80, 86],  [82, 80],  [85, 78],  [90, 90],  [86, 82],  [82, 90],  [78, 80],  [92, 94]] y = [84.2, 80.6, 80.1, 90, 83.2, 87.6, 79.4, 93.4]  # 实例化估计器 estimator = LinearRegression() # fit方法训练 estimator.fit(x, y) # 查看系数 coef = estimator.coef\_ print('线性回归系数', coef) # 预测 y\_1 = estimator.predict([[100, 40]]) print(y\_1) |

## 求导

#### 常见函数导数

|  |
| --- |
|  |

#### 导数的四则运算

|  |
| --- |
|  |

#### 矩阵求导

<https://en.wikipedia.org/wiki/Matrix_calculus#Scalar-by-vector_identities>

## 线性回归的损失和优化

#### 损失-函数-最小二乘法

|  |
| --- |
| 损失:预测值和真实值的差的平方和 |

#### 优化-正规方程

|  |
| --- |
| 正规方程: 求损失最小值,即损失的导数为0时的方程 |

#### 优化-梯度下降

|  |
| --- |
| 含义: 梯度下降,求损失的最小值,当特征值很多时,正规方程运算量大,采用梯度下降.每次求一个特征值的微分,以这个特征值的微分学习率(斜率)α做切线,以此来跳过部分特征值的求导操作,直到求导到最后.  举例:    公式:     1. α的含义   α在梯度下降算法中被称作为学习率或者步长，意味着我们可以通过α来控制每一步走的距离，以保证不要步子跨的太大扯着蛋，哈哈，其实就是不要走太快，错过了最低点。同时也要保证不要走的太慢，导致太阳下山了，还没有走到山下。 |

#### 对比

|  |
| --- |
|  |

## 梯度下降法

#### 全梯度下降算法(FG)

|  |
| --- |
| 每次对所有特征值求微分, 找到其中梯度最大的点.不过有可能进入局部最优解. |

#### 随机梯度下降算法(SG)

|  |
| --- |
| 每次随机对一个特征值求微分,再取下一个样本, 直到损失函数值停止下降或损失函数值小于某个可以容忍的阈值。 |

#### 小批量梯度下降算法(min-bantch)

|  |
| --- |
| 每次随机取部分(batch\_size个)特征值,再求这部分特征值的所有特征值的微分(即FG),选取梯度最大的. 当batch\_size个为1时,为SG. |

#### 随机平均梯度下降算法(SAG)

|  |
| --- |
| 在SG的基础上(随机取一个样本特征值求微分,得梯度), |

## 波士顿房价预测案例

|  |
| --- |
| SGDRegressor API:  SGDRegressor是scikit-learn库中的一个类，它通过最小化正则化的经验损失来拟合线性模型。SGD代表随机梯度下降：损失的梯度每次估计一个样本，模型沿着递减强度计划（即学习率）更新1。  SGDRegressor有许多参数，包括：  loss：要使用的损失函数，默认为squared\_error。可能的值为squared\_error、huber、epsilon\_insensitive或squared\_epsilon\_insensitive1。  penalty：要使用的惩罚（也称为正则化项），默认为l2。当设置为None时，不添加惩罚1。  alpha：乘以正则化项的常数。值越高，正则化越强。当learning\_rate设置为‘optimal’时，也用于计算学习率，默认值为0.00011。  l1\_ratio：Elastic Net混合参数，0 <= l1\_ratio <= 1。l1\_ratio=0对应L2惩罚，l1\_ratio=1对应L1。仅在惩罚为‘elasticnet’时使用，默认值为0.151。  fit\_intercept：是否应该估计截距。如果为False，则假定数据已经居中，默认值为True1。  max\_iter：训练数据（即时期）的最大传递次数。它只影响fit方法中的行为，而不影响partial\_fit方法，默认值为10001。  tol：停止标准。如果在max\_iter次迭代后，(loss > best\_loss - tol) 仍然成立，则优化停止，默认值为0.0011。 |

|  |
| --- |
| **from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.preprocessing **import** StandardScaler **from** sklearn.linear\_model **import** LinearRegression, **from** sklearn.metrics **import** mean\_squared\_error from sklearn.linear\_model import SGDRegressor  **def** normal\_equation():  *"""  线性回归正规方程* **:return***:  """  # 1 获取数据* **import** pandas **as** pd  **import** numpy **as** np   data\_url = **"http://lib.stat.cmu.edu/datasets/boston"** raw\_df = pd.read\_csv(data\_url, sep=**"\s+"**, skiprows=22, header=**None**)  boston\_data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])  target = raw\_df.values[1::2, 2]   *# 2 数据集划分* x\_feature\_train, x\_feature\_test, y\_target\_train, y\_target\_test = train\_test\_split(boston\_data,  target,  random\_state=22)  *# 3 特征工程-标准化* transfer = StandardScaler()  x\_feature\_train = transfer.fit\_transform(x\_feature\_train)  x\_feature\_test = transfer.fit\_transform(x\_feature\_test)   *# 4 机器学习-线性回归（正规方程）* estimator = LinearRegression()  estimator.fit(x\_feature\_train, y\_target\_train)   *# 5 模型评估  # 5.1 获取系数* y\_target\_predict = estimator.predict(x\_feature\_test)  print(**f'预测测试集目标值为\n {**y\_target\_predict**}'**)  print(**f'测试集的目标值为\n {**y\_target\_test**}'**)  print(**f'模型中的系数为\n：{**estimator.coef\_**}'**)  print(**f'模型中的偏置为\n：{**estimator.intercept\_**}'**)   *# 5.2 评价  # 均方误差* error\_mean\_square = mean\_squared\_error(y\_target\_test, y\_target\_predict)  print(**f'测试机的预测值和测试机的目标值的均方误差为:\n {**error\_mean\_square**}'**)  **def** sgd():  *"""  线性回归正规方程* **:return***:  """  # 1 获取数据* **import** pandas **as** pd  **import** numpy **as** np   data\_url = **"http://lib.stat.cmu.edu/datasets/boston"** raw\_df = pd.read\_csv(data\_url, sep=**"\s+"**, skiprows=22, header=**None**)  boston\_data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])  target = raw\_df.values[1::2, 2]   *# 2 数据集划分* x\_feature\_train, x\_feature\_test, y\_target\_train, y\_target\_test = train\_test\_split(boston\_data,  target,  random\_state=22)  *# 3 特征工程-标准化* transfer = StandardScaler()  x\_feature\_train = transfer.fit\_transform(x\_feature\_train)  x\_feature\_test = transfer.fit\_transform(x\_feature\_test)   *# 4 机器学习-线性回归（随机梯度下降）* estimator = SGDRegressor(loss=**"squared\_error"**,learning\_rate=**"constant"**,eta0=0.00003)  estimator.fit(x\_feature\_train, y\_target\_train)   *# 5 模型评估  # 5.1 获取系数* y\_target\_predict = estimator.predict(x\_feature\_test)  print(**f'预测测试集目标值为\n {**y\_target\_predict**}'**)  print(**f'测试集的目标值为\n {**y\_target\_test**}'**)  print(**f'模型中的系数为\n：{**estimator.coef\_**}'**)  print(**f'模型中的偏置为\n：{**estimator.intercept\_**}'**)   *# 5.2 评价  # 均方误差* error\_mean\_square = mean\_squared\_error(y\_target\_test, y\_target\_predict)  print(**f'测试机的预测值和测试机的目标值的均方误差为:\n {**error\_mean\_square**}'**)  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  normal\_equation()  sgd() |

## 欠拟合和过拟合

### 定义

欠拟合：训练集不好，测试集不好（模型过于简单）

过拟合：训练集很好，测试集不好（模型过于复杂）

### 原因和解决办法

* 欠拟合：

原因：学习到的数据的特征太少

解决办法：（1）添加其他特征项。（2）添加多项式特征让模型更复杂和灵活丰富。

* 过拟合：

原因：原始特征过多，存在嘈杂特征

解决办法：（1）重新清洗数据。（2）增大数据训练量。（3）正则化，惩罚模型参数的大小迫使模型变简单。（4）建减少特征维度，防止维灾难。

## 正则化和维灾难

### 正则化定义

在学习的时候，数据提供的特征有些影响模型复杂度或者这个特征的数据点异常较多，所以算法在学习的时候尽量减少这个特征的影响（甚至删除某个特征的影响），这就是正则化。

正则化技术通过向模型的损失函数中添加一个正则化项来实现这一目的。这个正则化项一般是模型参数的函数，并且惩罚模型参数的大小，逼迫模型更为简单，使得模型更加健壮。

### 类别

* L2正则化

作用：可以使得其中一些W的都很小，都接近于0，削弱某个特征的影响

优点：越小的参数说明模型越简单，越简单的模型则越不容易产生过拟合现象

Ridge回归（岭回归）

* L1正则化

作用：可以使得其中一些W的值直接为0，删除这个特征的影响

LASSO回归

### 维灾难

* 定义

随着维度的增加，分类器性能逐步上升，到达某点后，其性能逐渐下降

* 原因

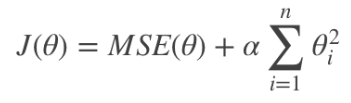
维度太多，训练数据无法覆盖特征空间。

维度太多，测试数据的判断愈发复杂，导致误判过拟合。

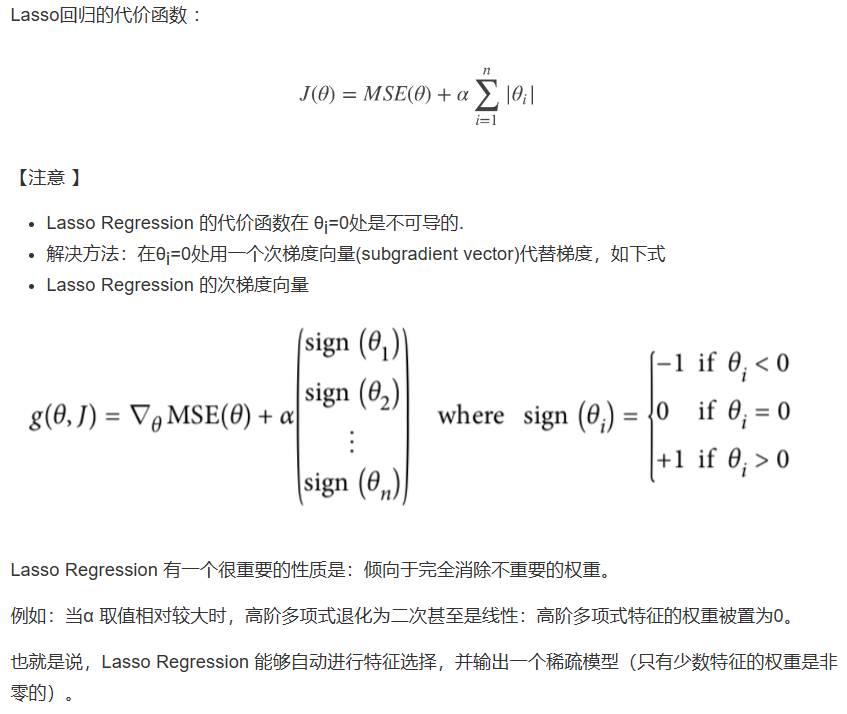
### 正则化线性模型

* Ridge Regression（岭回归）

岭回归是线性回归的正则化版本，即在原来的线性回归的损失函数里添加正则化项，以达到在拟合数据的同时，使模型权重尽可能的小。

（岭回归函数为损失函数均方误差加正则化项）

* Lasso Regression (Lasso回归)



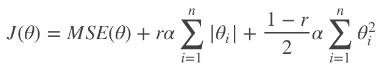
* Elastic Net（弹性网络）

弹性网络在岭回归和Lasso回归中进行了折中，通过 混合比(mix ratio) r 进行控制：

r=0：弹性网络变为岭回归

r=1：弹性网络便为Lasso回归

弹性网络的代价函数：



* 小结

常用：岭回归

假设只有少部分特征是有用的：

* + - 弹性网络
    - Lasso
    - 一般来说，弹性网络的使用更为广泛。因为在特征维度高于训练样本数，或者特征是强相关的情况下，Lasso回归的表现不太稳定。

api:

* + - from sklearn.linear\_model import Ridge, ElasticNet, Lasso

## 岭回归

### 简介

SGDRegressor接口对应的是随机梯度下降算法，岭回归对应的手动平均随机梯度下降算法。

### api

sklearn.linear\_model.Ridge是一个线性回归模型，它使用岭回归算法来拟合数据。Ridge回归通过对系数的大小施加惩罚来解决多重共线性问题。这个模型包含了一个正则化参数alpha，它控制着惩罚的强度。当alpha=0时，Ridge回归退化为普通最小二乘法；当alpha趋近于无穷大时，所有的系数都趋近于0。

入参：

* alpha：惩罚力度。
* fit\_intercept：是否拟合截距。默认为True。
* normalize：是否对自变量进行标准化。默认为False。
* copy\_X：是否复制自变量。默认为True。
* max\_iter：最大迭代次数。默认为None，表示没有限制。
* tol：收敛容差。默认为1e-3。
* solver:

会根据数据自动选择优化方法，如果数据集、特征都比较大，选择该随机梯度下降优化。

auto：自动选择求解器。

svd：使用奇异值分解方法求解。

cholesky：使用Cholesky分解方法求解。

lsqr：使用最小二乘法的迭代优化算法求解。

sparse\_cg：使用共轭梯度法求解。

属性：

* coef\_：回归系数。
* intercept\_：回归截距。
* alpha\_：实际使用的正则化参数值。
* n\_iter\_：实际迭代次数。
* rank\_：自变量矩阵的秩。
* singular\_：自变量矩阵的奇异值。

实例：

|  |
| --- |
| **def** ridge():  *"""  岭回归* **:return***:  """  # 1 获取数据* **import** pandas **as** pd  **import** numpy **as** np   data\_url = **"http://lib.stat.cmu.edu/datasets/boston"** raw\_df = pd.read\_csv(data\_url, sep=**"\s+"**, skiprows=22, header=**None**)  boston\_data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])  target = raw\_df.values[1::2, 2]   *# 2 数据集划分* x\_feature\_train, x\_feature\_test, y\_target\_train, y\_target\_test = train\_test\_split(boston\_data,  target,  random\_state=22)  *# 3 特征工程-标准化* transfer = StandardScaler()  x\_feature\_train = transfer.fit\_transform(x\_feature\_train)  x\_feature\_test = transfer.fit\_transform(x\_feature\_test)   *# 4 机器学习-线性回归（随机梯度下降）* estimator = Ridge(alpha=0.1)  *#estimator = RidgeCV(alphas=(0.1, 1, 10))* estimator.fit(x\_feature\_train, y\_target\_train)   *# 5 模型评估  # 5.1 获取系数* y\_target\_predict = estimator.predict(x\_feature\_test)  print(**f'预测测试集目标值为\n {**y\_target\_predict**}'**)  print(**f'测试集的目标值为\n {**y\_target\_test**}'**)  print(**f'模型中的系数为\n：{**estimator.coef\_**}'**)  print(**f'模型中的偏置为\n：{**estimator.intercept\_**}'**)   *# 5.2 评价  # 均方误差* error\_mean\_square = mean\_squared\_error(y\_target\_test, y\_target\_predict)  print(**f'测试机的预测值和测试机的目标值的均方误差为:\n {**error\_mean\_square**}'**) |

sklearn.linear\_model.RidgeCV是一个带有交叉验证的岭回归模型。它可以自动选择最优的正则化参数alpha，并使用交叉验证来评估模型的性能。

入参：

* alphas：正则化参数的候选值。默认为None，表示自动选择。
* fit\_intercept：是否拟合截距。默认为True。
* normalize：是否对自变量进行标准化。默认为False。
* scoring：用于评估模型性能的指标。默认为None，表示使用默认指标。
* cv：交叉验证的折数。默认为None，表示使用5折交叉验证。
* gcv\_mode：广义交叉验证模式。默认为None，表示使用默认模式。
* store\_cv\_values：是否存储每个alpha值对应的交叉验证得分。默认为False。

属性：

* alpha\_：实际使用的正则化参数值。
* coef\_：回归系数。
* intercept\_：回归截距。
* cv\_values\_：每个alpha值对应的交叉验证得分。
* best\_score\_：最佳交叉验证得分。

## 模型保存和加载

### api

from sklearn.externals import joblib # 新版本无此接口

import joblib

* 保存：joblib.dump(estimator, 'test.pkl')
* 加载：estimator = joblib.load('test.pkl')

注：划分训练集和测试集的时候随机数种子保持一致

### 使用

|  |
| --- |
| **from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.preprocessing **import** StandardScaler **from** sklearn.linear\_model **import** LinearRegression **from** sklearn.metrics **import** mean\_squared\_error **from** sklearn.linear\_model **import** SGDRegressor **from** sklearn.linear\_model **import** Ridge,RidgeCV **import** joblib   **def** ridge\_jump\_load(mode: str):  *"""  模型的保存和加载* **:arg** *mode:jump/load 。对应保存和加载* **:return***:  """  # 1 获取数据* **import** pandas **as** pd  **import** numpy **as** np   data\_url = **"http://lib.stat.cmu.edu/datasets/boston"** raw\_df = pd.read\_csv(data\_url, sep=**"\s+"**, skiprows=22, header=**None**)  boston\_data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])  target = raw\_df.values[1::2, 2]   *# 2 数据集划分* x\_feature\_train, x\_feature\_test, y\_target\_train, y\_target\_test = train\_test\_split(boston\_data, target,  random\_state=22)  *# 3 特征工程-标准化* transfer = StandardScaler()  x\_feature\_train = transfer.fit\_transform(x\_feature\_train)  x\_feature\_test = transfer.fit\_transform(x\_feature\_test)   **if** mode == **'jump'**:  *# 4 机器学习-线性回归（随机梯度下降）  #estimator = Ridge(alpha=0.1)* estimator = RidgeCV(alphas=(0.1, 1, 10))  estimator.fit(x\_feature\_train, y\_target\_train)  joblib.dump(estimator, **"./14.模型保存和加载/test.pkl"**)  **elif** mode == **'load'**:  estimator = joblib.load(**"./14.模型保存和加载/test.pkl"**)  **else**:  **raise '入参错误'** *# 5 模型评估  # 5.1 获取系数* y\_target\_predict = estimator.predict(x\_feature\_test)  print(**f'预测测试集目标值为\n {**y\_target\_predict**}'**)  print(**f'测试集的目标值为\n {**y\_target\_test**}'**)  print(**f'模型中的系数为\n：{**estimator.coef\_**}'**)  print(**f'模型中的偏置为\n：{**estimator.intercept\_**}'**)   *# 5.2 评价  # 均方误差* error\_mean\_square = mean\_squared\_error(y\_target\_test, y\_target\_predict)  print(**f'测试机的预测值和测试机的目标值的均方误差为:\n {**error\_mean\_square**}'**)  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  ridge\_jump\_load(**'load'**) |

## 个人总结

|  |
| --- |
| 线性回归是求解线性方程的过程，不断梯度下降求w特征值。  梯度下降的方法是求解最小二乘法的损失函数。  根据每次梯度下降使用的样本数量分为了很多种梯度下降算法。  通过正则化减小某个特征值的影响（防止噪声） |

# 逻辑回归

## 简介

逻辑回归是先将样本的特征值和目标值数据进行线性回归，得到模型。将线性回归的结果y当作输入到逻辑回归的sigmoid函数，此函数将y映射到0到1之间。求损失时，将真实值为0和1的区分开，用-log(y)代表真实值为1的损失，这样y越小(即偏离1越多)时损失将变大，用-log(1-y)作为真实值为0的损失。。

## 原理

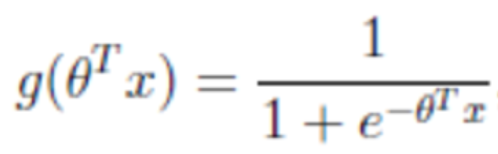
### 输入

逻辑回归的输入就是一个线性回归的结果h(w)。

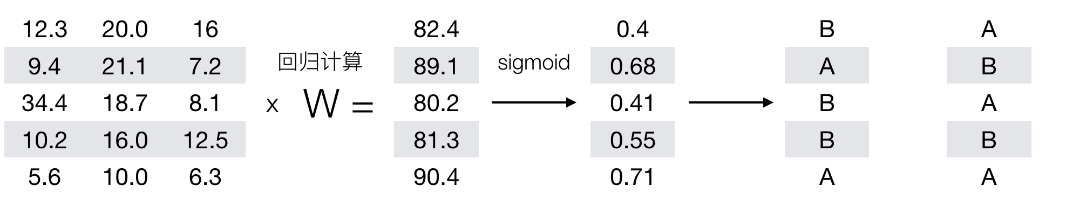
### 激活函数

此函数把线性回归的结果h(w)作为输入，输出[0,1]区间的一个概率值。（可以理解为标准化，便于和真实值0或者1对比）

激活函数sigmoid:



流程演示：

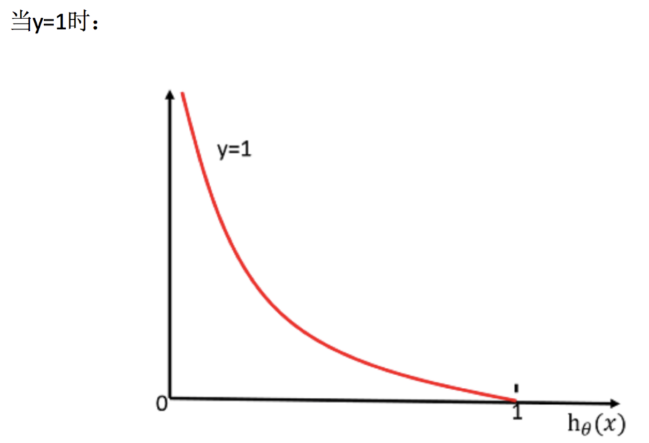


## 损失和优化

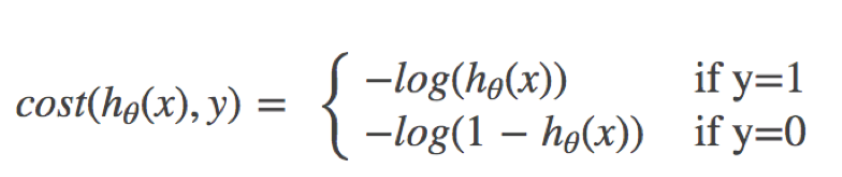
### 损失

损失即为经过sigmoid函数得出的概率值hθ(x)和真实值0或者1的相对距离。

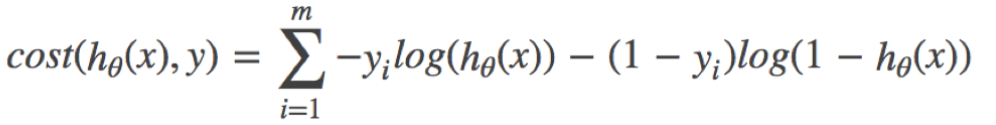
* 当真实值为1时，损失为1- hθ(x)。但是为了给与更大更合理的“惩罚“，将损失定义为-log(hθ(x))，这样当hθ(x)越远离1，损失函数将越大；当hθ(x)越接近1时，损失函数将越小；它不再是单纯的线性关系。
* 当真实值为0时反之，为-log(1- hθ(x))。



对数似然损失：



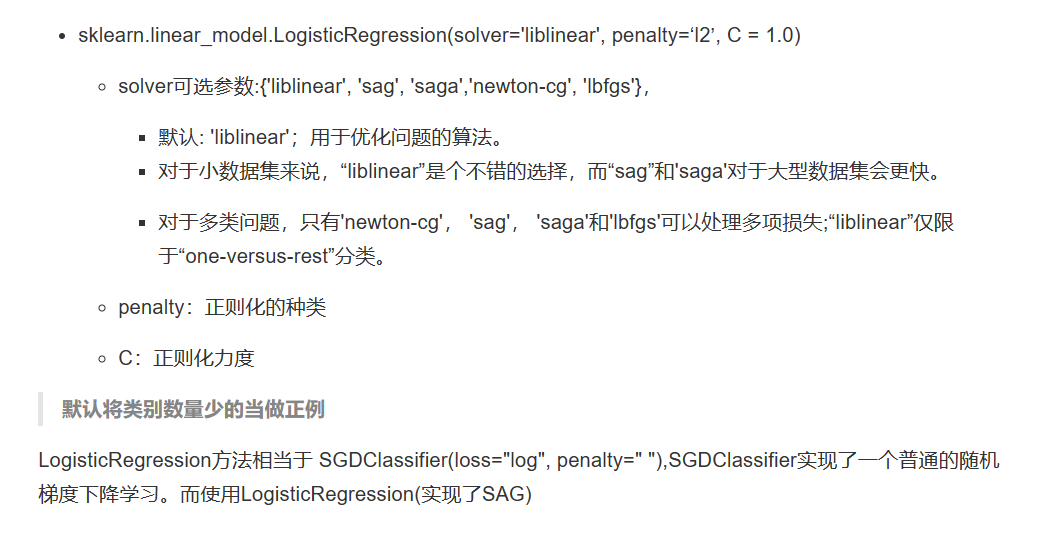
综合完整的损失函数:



### 优化

同样使用梯度下降优化算法，去减少损失函数的值。梯度下降求损失函数最小值，每次求损失函数的偏导，更新参数。 这样去更新逻辑回归前面对应算法的权重参数，**提升原本属于1类别的概率，降低原本是0类别的概率。**

## API



## 案例

|  |
| --- |
| *#!/usr/bin/env python # -\*- coding: utf-8 -\*- # @Time : 2023/3/31 21:07 # @Author : 刘双喜 # @File : 16.癌症分类预测.py # @Description : 添加描述* **import** pandas **as** pd **import** numpy **as** np **from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.preprocessing **import** StandardScaler **from** sklearn.linear\_model **import** LogisticRegression *# 1.获取数据 # 2.基本数据处理 # 2.1 缺失值处理 # 2.2 确定特征值,目标值 # 2.3 分割数据 # 3.特征工程(标准化) # 4.机器学习(逻辑回归) # 5.模型评估  # 获取数据* name = [**'示例编号'**, **'肿块厚度'**, **'细胞大小均匀性'**, **'细胞形状均匀性'**,  **'边际附着力'**, **'单个上皮细胞大小'**, **'裸核'**, **'乏味的染色质'**,  **'正常的核仁'**, **'有丝分裂'**, **'类别'**] data = pd.read\_csv(**'16.breast-cancer-wisconsin.data'**, names=name)  *# 缺失值处理* data = data.replace(**'?'**, np.nan) data = data.dropna()  *# 确定特征值和目标值* x, y = data.iloc[:, 1:-1], data.iloc[:, -1]  *# 分割数据* x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,random\_state=22)  *# 标准化* transfer = StandardScaler() x\_train = transfer.fit\_transform(x\_train) x\_test = transfer.fit\_transform(x\_test)  *# 机器学习* estimator = LogisticRegression() estimator.fit(x\_train, y\_train)  *# 模型评估* y\_predict = estimator.predict(x\_test) print(**f'特征值的测试集的预测结果为：{**y\_predict**}'**) score = estimator.score(x\_test, y\_test) print(**f'分数：{**score**}'**) is\_equal = np.equal(y\_predict, y\_test) true\_count = np.count\_nonzero(is\_equal == **True**) print(**f'准确率:{**true\_count/len(y\_predict)**}'**) |

## 分类评估方法

### 混淆矩阵

真实结果和预测结果的正例和反例组合成混淆矩阵。



### 精确率(precision)和召回率(Recall)

* 精确率：预测结果为正例中真实为正例的比例（预测正例对的比例）(TP/TP+FP)

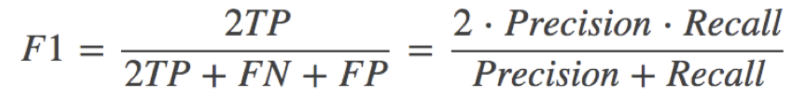


* 召回率：真实结果为正例中预测为正例的比例（真实正例对的比例）（TP/TP+FN）

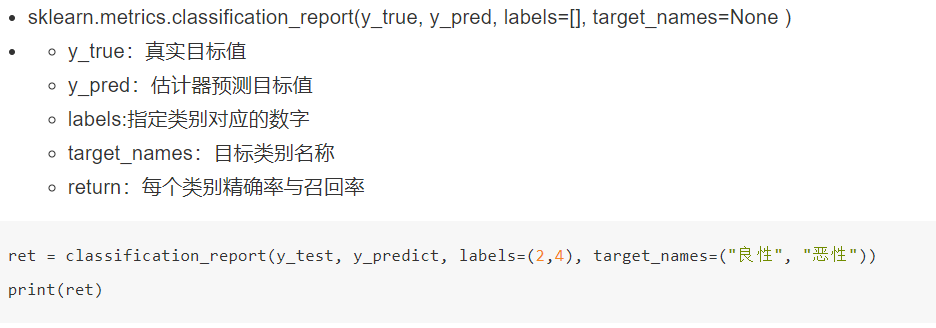


### F1-score

* 一种评估标准，反映了模型的稳健性



### API



## ROC曲线和AUG指标

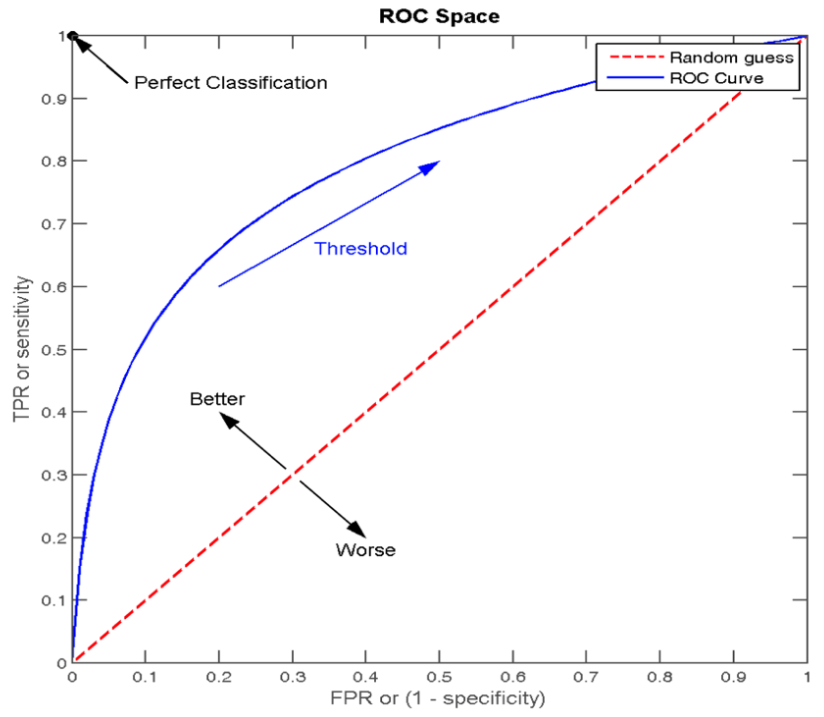
假设一种情况，如果99个样本癌症，1个样本非癌症，不管怎样我全都预测正例(默认癌症为正例),准确率就为99%但是这样效果并不好，这就是样本不均衡下的评估问题

### TPR和FPR

* TPR=TP/(TP+FN) (所有真实类别为1的样本中，预测类别为1的比例) (相当于召回率)
* FPR=FP/(FP+TN) (所有真实类别为0的样本中，预测类别为1的比例)

### ROC曲线

ROC曲线的横轴就是FPRate，纵轴就是TPRate，当二者相等时，表示的意义则是：对于不论真实类别是1还是0的样本，分类器预测为1的概率是相等的，此时AUC为0.5



### AUC指标

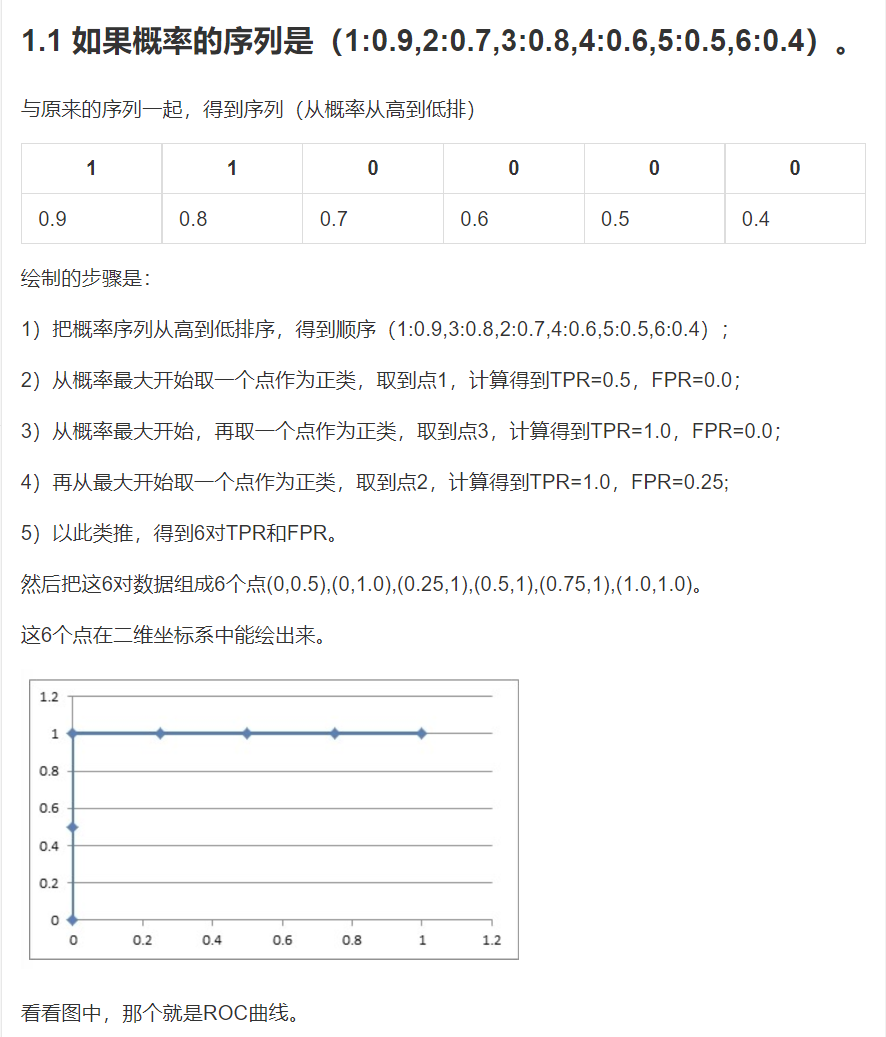
* AUC指标就是ROC曲线的面积(Area Under roc Curve)
* AUC的概率意义是随机取一对正负样本，正样本得分大于负样本的概率
* AUC的最小值为0.5，最大值为1，取值越高越好
* AUC=1，完美分类器，采用这个预测模型时，不管设定什么阈值都能得出完美预测。绝大多数预测的场合，不存在完美分类器。
* 0.5<AUC<1，优于随机猜测。这个分类器（模型）妥善设定阈值的话，能有预测价值。
* 最终AUC的范围在[0.5, 1]之间，并且越接近1越好

### API

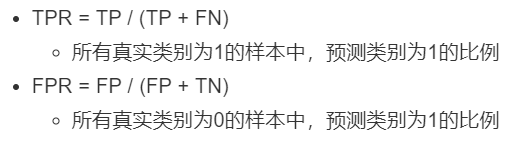


* AUC只能用来评价二分类
* AUC非常适合评价样本不平衡中的分类器性能

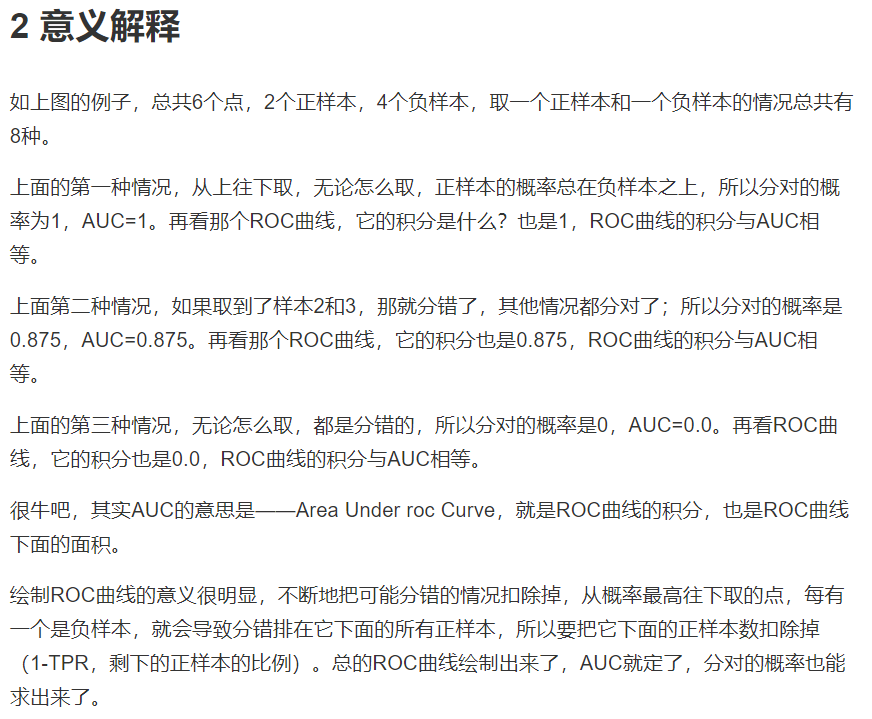
## ROC曲线的绘制



如表所示：



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 概率 | 1:0.9 | 3:0.8 | 2:0.7 | 4:0.6 | 5:0.5 | 6:0.4 | TRR | FRR |
| 真实值 | 1 | 1 | 0 | 0 | 0 | 0 |  |  |
| 步骤2）预测值 | 1 | 0 | 0 | 0 | 0 | 0 | 1/2 | 0/4 |
| 步骤3）预测值 | 1 | 1 | 0 | 0 | 0 | 0 | 2/2 | 0/4 |
| 步骤4）预测值 | 1 | 1 | 1 | 0 | 0 | 0 | 2/2 | 1/4 |



## 个人总结

|  |
| --- |
| 逻辑回归解决分类问题，回归问题用于预测连续的值。  逻辑回归将线性回归的输出值经过sigmoid函数转换到0到1之间。  通过增大对错误选择的惩罚和减小对正确选择的惩罚，列出对数释然损失。  用梯度下降计算损失函数。 |

# 决策树

## 简介

通过计算每个决策熵值的变化大小，来进行决策。

## 熵

### 概念

熵Entropy:混乱程度的量度。

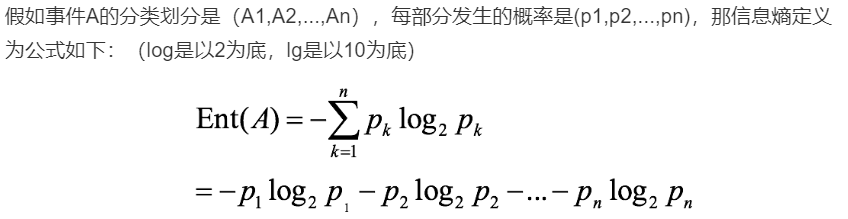
1、从信息的完整性上进行的描述:

当系统的有序状态一致时，\*\*数据越集中的地方熵值越小，数据越分散的地方熵值越大。

2、从信息的有序性上进行的描述:

当数据量一致时，系统越有序，熵值越低；系统越混乱或者分散，熵值越高。

### 信息熵



## 决策树划分依据一：信息增益

### 概念

* 信息增益=熵（前）-熵（后）
* 信息增益：以某特征划分数据集前后的熵的差值。熵可以表示样本集合的不确定性，熵越大，样本的不确定性就越大。因此可以使用划分前后集合熵的差值来衡量使用当前特征对于样本集合D划分效果的好坏。
* 定义：特征A对训练数据集D的信息增益g(D,A),定义为集合D的信息熵H(D)与特征A给定条件下D的信息条件熵H(D|A)之差，即公式为：g(D,A)= H(D)- H(D|A)。

即：整体的信息熵-某个类别下所有条件的每个信息熵（如整体-性别类熵=整体-男性熵-女性熵）

### 案例

|  |
| --- |
|  |

## 决策树划分依据二：信息增益率

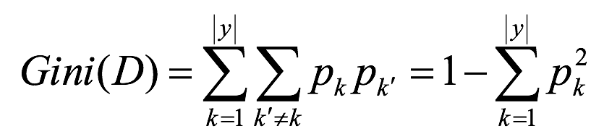
对应类型的信息增益和它的信息熵的比值。

## 决策树划分依据三：基尼值和基尼指数

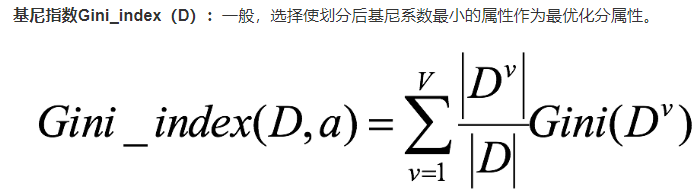
### 概念

基尼值：从数据集D中随机抽取两个样本，其类别标记不一致的概率。基尼值越小，数据纯度越高。决策时先决策数据纯度高的分类。

Gini(D) = 1 - 一致的概率和 = 1-Σp\*p



基尼指数：各部分基尼值，以概率为权重，加权的基尼值之和。



基尼增益：根节点基尼值-分类A基尼系数=根节点基尼值-分类A各类别基尼值和概率

（基尼

### 案例

|  |
| --- |
| 1. 根节点基尼值      1. 是否有房基尼值   是：  否：  {是否有房基尼增益} =   1. 是否结婚基尼增益      1. 年收入基尼增益   取每两个年收入之间的中间值来划分数据，求每部分数据的基尼增益。    本轮基尼增益计算结束，分组为是否结婚的基尼增益最高，第一轮决策按是否结婚决策。  第二轮决策去掉第一轮决策的是否结婚后，剩下的数据继续计算各分类的基尼增益。  ... |

## 总结

### ID3算法

* 信息增益为属性
* 缺点是倾向于选择取值较多的属性
* 只能对属性为离散型的数据构造决策树

### C4.5算法

* 对ID3算法进行的改进
* 信息增益率为属性
* 可以处理连续数值型属性
* 后剪枝方法
* 对缺失值进行处理
* 优点：产生的分类规则易于理解，准确率较高
* 缺点：构造树的过程中，需要对数据集进行多次顺序扫描和排序，因此算法低效

C4.5只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时程序无法运行。

### CART算法

* 简化为二叉树模型
* 属性为简化的信息增益，即基尼系数

同时，无论是ID3, C4.5还是CART,在做特征选择的时候都是选择最优的一个特征来做分类决策，但是大多数，**分类决策不应该是由某一个特征决定的，而是应该由一组特征决定的。**这样决策得到的决策树更加准确。这个决策树叫做多变量决策树(multi-variate decision tree)。在选择最优特征的时候，多变量决策树不是选择某一个最优特征，而是选择最优的一个特征线性组合来做决策。这个算法的代表是OC1.

## CART剪枝

### 为什么要剪枝

随着决策的节点变多，数据出现过拟合，在训练集表现好，测试集表现差。

原因：噪声、样本冲突；特征值属性不能完全做为分类标准；巧合的规律性，数据量不够大。

### 预剪枝

* 1、节点下最小样本数目规定 达到某种不再划分
* 2、指定树的深度 达到某种不再划分
* 3、指定节点的熵 小于某值不再划分

### 后剪枝

在已生成过拟合决策树上进行剪枝，可以得到简化版的剪枝决策树。

## 特征工程

### 特征提取

* 定义：将任意数据（如文本或图像）转换为可用于机器学习的数字特征
* 分类：字典特征提取。文本特性提取。图像特征提取。

### 字典特征提取

|  |
| --- |
| **from** sklearn.feature\_extraction **import** DictVectorizer  **def** dict\_feature\_extra():  *"""  对字典进行特征提取* **:return***:  """* data = [{**'city'**: **'北京'**, **'temperature'**: 100}, {**'city'**: **'上海'**, **'temperature'**: 60}, {**'city'**: **'深圳'**, **'temperature'**: 30}]  *# 实例化转换器* transfer = DictVectorizer(sparse=**False**) *# sparse默认True  # 训练* data = transfer.fit\_transform(data)  print(**f'结果：{**data**}'**)  print(**f'特征名字:\n:{**transfer.get\_feature\_names\_out()**}'**) |

### 文本特征提取（英文）

|  |
| --- |
| **from** sklearn.feature\_extraction.text **import** CountVectorizer  **def** txt\_english\_feature\_extra():  *"""  对文本进行特征提取, 默认以空格分隔* **:return***:  """* data = [**"life is short,i like python"**, **"life is too long,i dislike python"**]  *# 实例化转换器* transfer = CountVectorizer()  *# 训练* data = transfer.fit\_transform(data)  print(**f'结果：{**data.toarray()**}'**)  print(**f'特征名字:\n:{**transfer.get\_feature\_names\_out()**}'**) |

### 文本特征提取（中文）

|  |
| --- |
| **from** sklearn.feature\_extraction.text **import** CountVectorizer  **def** txt\_chinese\_feature\_extra():  *"""  对中文文本进行特征提取, 先转换成空格的形式* **:return***:  """* **def** word\_cut(txt):  *"""  将一句话按照中文习惯以空格断句* **:return***:  """* **import** jieba  ls\_txt = list(jieba.cut(txt))  res = **' '**.join(ls\_txt)  **return** res   data = [**"《三体》，豆瓣9分，，主演：张鲁一 于和伟 陈瑾 王子文 林永健 李小冉 王传君 张帆 白客 涂松岩 刘敏 孔琳 张峻."**,  **"2007年，地球基础科学出现了异常的扰动，一时间科学界风雨飘飘，人心惶惶。离奇自杀的科学家，近乎神迹的倒计时，行事隐秘的科学边，神秘莫测的《三体》游戏……纳米科学家汪淼被警官史强带到联合作战中心，并潜入名为“科学边界”的组织协助调查。"**,  **"迷雾之中，汪淼接触到一个名为ETO的 组织，发现其幕后统帅竟是自杀身亡的科学家杨冬的母亲——叶文洁。随着ETO与作战中心你来我往的不断博弈，汪淼和史强逐渐确定《三体》游戏中的世界真实存在。而所有事件的源起，是两个文明为了生存空间，孤注一掷的生死相逐。在联合作战中心及科学家们的共同努力下，汪淼、史强等人坚定信念、重燃希望，带领大家继续准备着在今后与即将入侵的三体人展开殊死斗争"**]  data\_cut = [word\_cut(txt) **for** txt **in** data]  *# 实例化转换器* transfer = CountVectorizer()  *# 训练* data = transfer.fit\_transform(data\_cut)  print(**f'结果：{**data.toarray()**}'**)  print(**f'特征名字:\n:{**transfer.get\_feature\_names\_out()**}'**) |

### Tf-idf文本特征提取

* TF-IDF的主要思想是：如果某个词或短语在一篇文章中出现的概率高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。
* TF-IDF作用：用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。
* 词频（term frequency,TF）：某一个给定的词语在该文件中出现的频率。
* 逆向文档频率（inverse document frequency）：衡量一个词语普遍性的度量。
* IDF计算：总文件数目/包含该词语的文件数据，再将结果取以10为底的对数。

## 泰坦尼克号生存预测

|  |
| --- |
| *#!/usr/bin/env python # -\*- coding: utf-8 -\*- # @Time : 2023/4/21 21:23 # @Author : 刘双喜 # @File : 25.决策树算法鸢尾花数据演示.py # @Description : 添加描述* **import** pandas **as** pd **from** sklearn.feature\_extraction **import** DictVectorizer **from** sklearn.tree **import** DecisionTreeClassifier, export\_graphviz   *# 1、获取数据* data\_train = pd.read\_csv(**'25.titanic\\train.csv'**)  *# 2、数据基本处理 ## 确定目标值、特征值* x\_train = data\_train[[**"Pclass"**, **"Age"**, **"Sex"**]] y\_train = data\_train[[**'Survived'**]] *## 处理缺失值* x\_train[**'Age'**].fillna(x\_train[**'Age'**].mean(), inplace=**True**) *## 数据集划分* x\_test = pd.read\_csv(**'25.titanic\\test.csv'**)[[**"Pclass"**, **"Age"**, **"Sex"**]] y\_test = pd.read\_csv(**'25.titanic\\gender\_submission.csv'**)[[**'Survived'**]] *## 处理缺失值* x\_test[**'Age'**].fillna(x\_train[**'Age'**].mean(), inplace=**True**) *# 特征工程* transfer = DictVectorizer(sparse=**False**) x\_train = transfer.fit\_transform(x\_train.to\_dict(**'records'**)) x\_test = transfer.fit\_transform(x\_test.to\_dict(**'records'**)) print(transfer.get\_feature\_names\_out())  *# 3、机器学习 ## 实例化估计器* estimator = DecisionTreeClassifier(criterion=**'entropy'**, max\_depth=5) *## 训练* estimator.fit(x\_train,y\_train)  *# 4、模型评估* socre = estimator.score(x\_test,y\_test) print(**'分数：'**, socre) y\_predict = estimator.predict(x\_test) print(**'预测值：'**, y\_predict)  *# 5、保存树的结构到dot文件* export\_graphviz(estimator, out\_file=**'25.titanic\_tree.dot'**, feature\_names=transfer.get\_feature\_names\_out()) *# 查看dot文件http://webgraphviz.com/* |

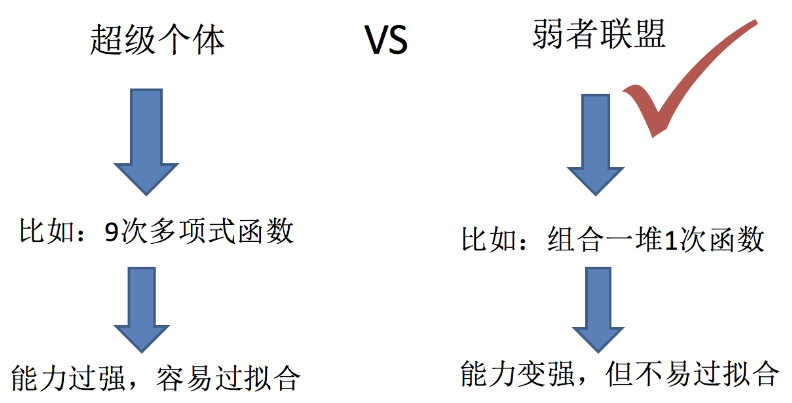
## 个人总结

|  |
| --- |
| 决策树本质就是基于已有数据算各个分类数据的概念差。  用log计算的是信息增益，直接用分数计算概率的是基尼增益。  ID3-信息增益；  C4.5-信息增益率；  CART-基尼增益； |

# 集成学习

## 简介

集成学习通过建立几个模型来解决单一预测问题。它的工作原理是**生成多个分类器/模型**，各自独立地学习和作出预测。这些预测最后**结合成组合预测**，因此优于任何一个单分类的做出预测。



## 集成学习的boosting和bagging

机器学习的核心任务：

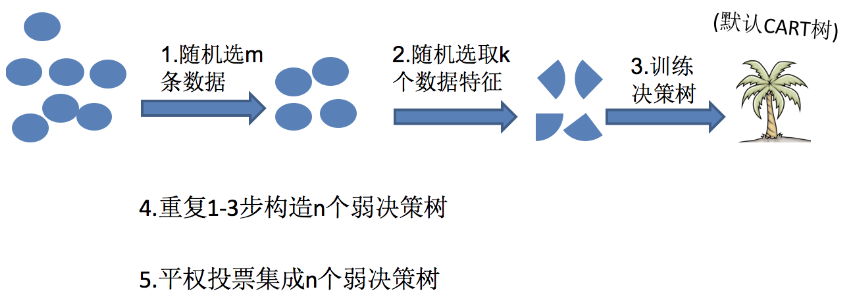
* 优化训练数据 —— 解决欠拟合问题 —— boosting逐步增强学习 — 弱弱组合变**强**
* 提高泛化性能 —— 解决过拟合问题 —— bagging采样集成学习 — 相互扼制变**稳**

bagging：采样 —— 学习 —— 集成

## 随机森林构造

随机森林是一个包含多个决策树的分类器，其输出类别由个别树输出类别的众数而定。

随机森林 = bagging + 决策树



随机森林够造过程中的关键步骤（用N来表示训练用例（样本）的个数，M表示特征数目）：

​1）一次随机选出一个样本，有放回的抽样，重复N次（有可能出现重复的样本）

​2）随机去选出m个特征, m <<M，建立决策树

1.为什么要随机抽样训练集？

如果不进行随机抽样，每棵树的训练集都一样，那么最终训练出的树分类结果也是完全一样的

2.为什么要有放回地抽样？

如果不是有放回的抽样，那么每棵树的训练样本都是不同的，都是没有交集的，这样每棵树都是“有偏的”，都是绝对“片面的”（当然这样说可能不对），也就是说每棵树训练出来都是有很大的差异的；而随机森林最后分类取决于多棵树（弱分类器）的投票表决。

## 随机森林API

|  |
| --- |
|  |

## 泰坦尼克号案例

|  |
| --- |
| **import** pandas **as** pd **from** sklearn.feature\_extraction **import** DictVectorizer **from** sklearn.ensemble **import** RandomForestClassifier **from** sklearn.model\_selection **import** GridSearchCV  *# 1、获取数据* data\_train = pd.read\_csv(**'25.titanic\\train.csv'**)  *# 2、数据基本处理 ## 确定目标值、特征值* x\_train = data\_train[[**"Pclass"**, **"Age"**, **"Sex"**]] y\_train = data\_train[**'Survived'**] *## 处理缺失值* x\_train[**'Age'**].fillna(x\_train[**'Age'**].mean(), inplace=**True**) *## 数据集划分* x\_test = pd.read\_csv(**'25.titanic\\test.csv'**)[[**"Pclass"**, **"Age"**, **"Sex"**]] y\_test = pd.read\_csv(**'25.titanic\\gender\_submission.csv'**)[**'Survived'**] *## 处理缺失值* x\_test[**'Age'**].fillna(x\_train[**'Age'**].mean(), inplace=**True**) *# 特征工程* transfer = DictVectorizer(sparse=**False**) x\_train = transfer.fit\_transform(x\_train.to\_dict(**'records'**)) x\_test = transfer.fit\_transform(x\_test.to\_dict(**'records'**)) print(**'提取特征后的特征名：'**, transfer.get\_feature\_names\_out())  *# 3、机器学习 ## 实例化估计器* estimator = RandomForestClassifier() *# 实例化网格搜索对象，训练* param\_grid = {**"n\_estimators"**: [120, 200, 300, 500, 800, 1200], **"max\_depth"**: [5, 8, 15, 25, 30]} estimator = GridSearchCV(estimator, param\_grid=param\_grid, cv=5) *## 训练* estimator.fit(x\_train, y\_train)  *# 4、模型评估 # 最好的模型* best\_estimator = estimator.best\_estimator\_ print(**'最好的模型: '**, best\_estimator) *# 分数* socre = estimator.score(x\_test, y\_test) print(**'分数：'**, socre) y\_predict = estimator.predict(x\_test) print(**'预测值：'**, y\_predict) |

## Boosting和GBDT

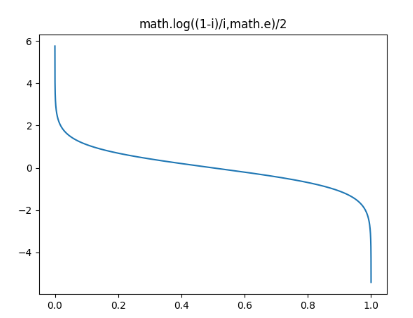
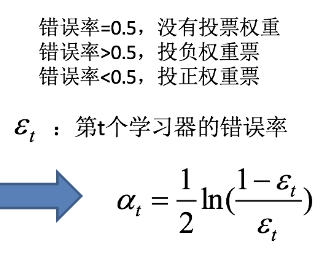
### Boosting

#### 简介

* 每加入一个弱学习器，整体能力都得到提升
* 代表算法Adaboost、GBDT、XGboosting

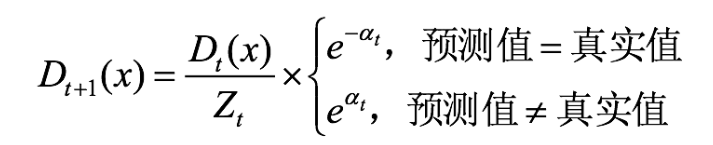
#### Adaboost实现过程

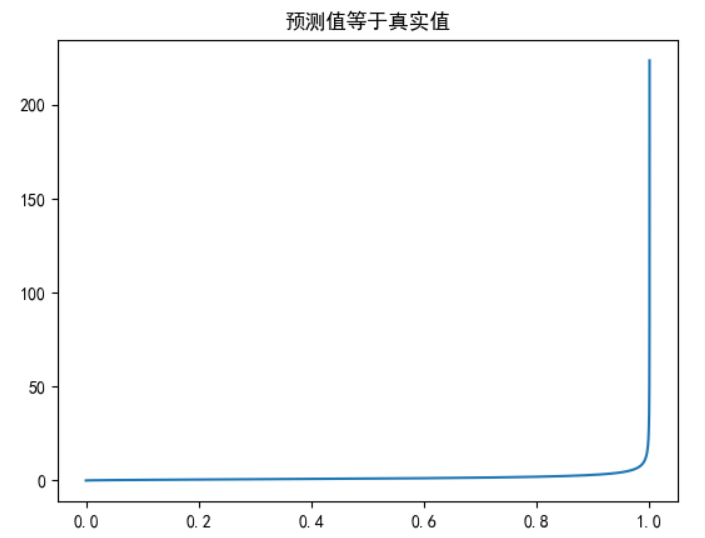
* 每次学习错误的数据放大，正确的数据放小
* 根据错误率确定投票权重。下公式错误率为0.5时权重为0，错误率越大，负权重越大。



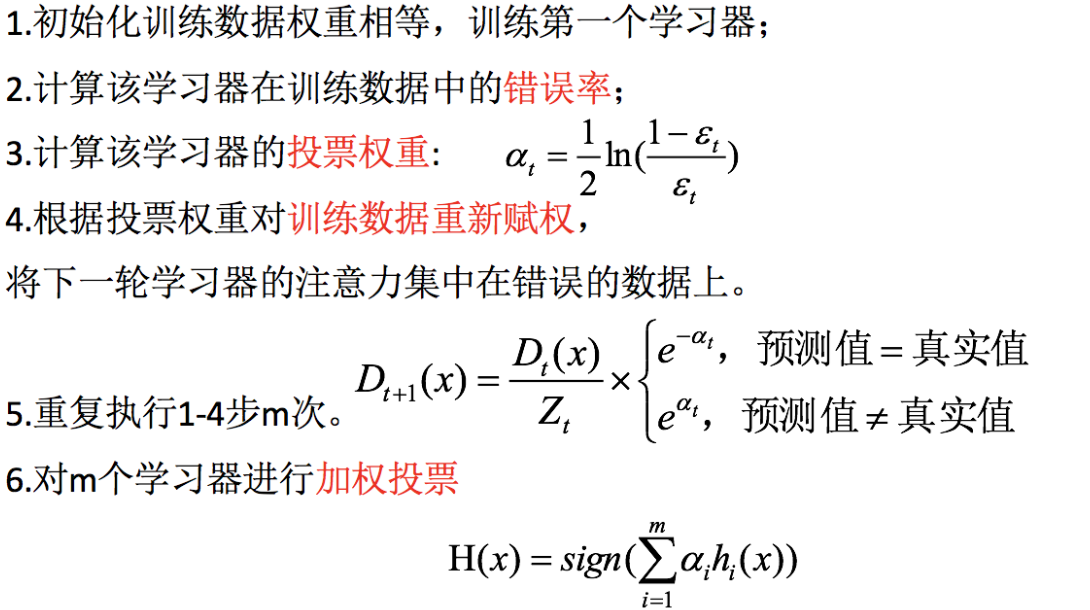
* 调整数据分布。核心原则：前一轮学习正确的数据忽视，错误的数据重视。

如下等于真实值时，错误率越小分布越小（预测对时忽视）。预测值不等于真实值时，错误率越小分布越大（错误的数据重视）



AdaBoost小结



#### bagging集成和boosting集成的区别

* 数据方面

bagging对数据进行采样训练

boosting根据前一轮学习结果调整数据的重要性

* 投票方面

bagging对所有学习器平权投票

boosting对学习器进行加权投票

* 学习顺序

bagging的学习是并行的，每个学习器没有依赖关系

boosting的学习是串行，学习有先后

* 主要作用

bagging主要用于提高泛化性能（解决过拟合，减低方差）

boosting主要用于提高训练精度（解决欠拟合，减低偏差）

#### API介绍

from sklearn.ensemble import AdaBoostClassifier

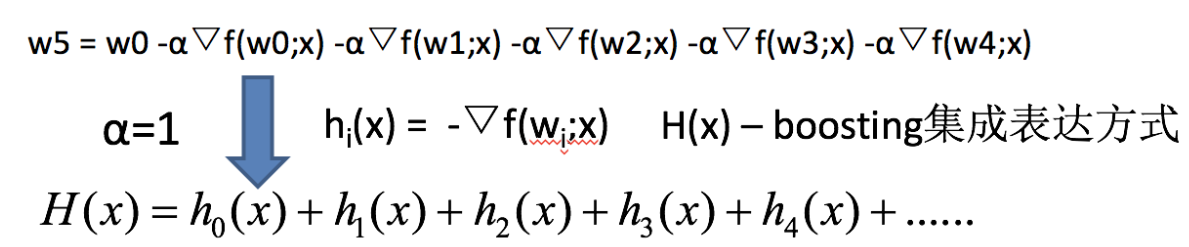
api链接:

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier>

### GBDT

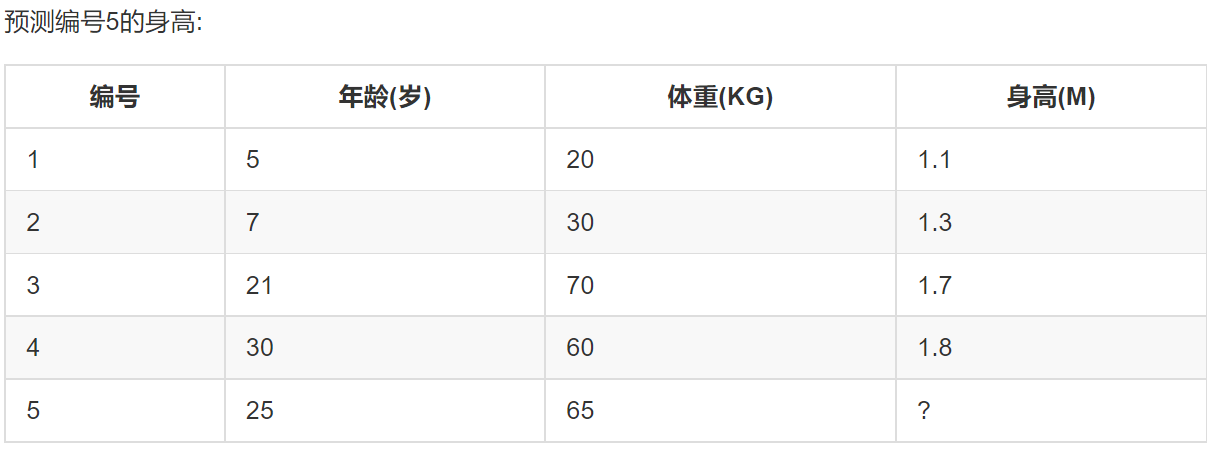
梯度提升决策树(GBDT Gradient Boosting Decision Tree) 是一种迭代的决策树算法，该算法由多棵决策树组成，所有树的结论累加起来做最终答案。它在被提出之初就被认为是泛化能力（generalization)较强的算法。近些年更因为被用于搜索排序的机器学习模型而引起大家关注。

GBDT = 梯度下降 + Boosting + 决策树



H(X)是预测值，每一个h(x)是梯度下降最快的方向。所有梯度下降最快的方向之和就是损失函数最小点，也就是预测值。

案例：



第一步：根据身高的损失函数求出损失最小的预测身高

第二步：用第一步的预测身高，求出年龄的对它的误差的方差。

第三步：加上上一步的预测值之后，以体重为划分，再求体重的误差的方差。

### XGBoost

XGBoost= 二阶泰勒展开+boosting+决策树+正则化

面试题：了解XGBoost么，请详细说说它的原理

回答要点：二阶泰勒展开，boosting，决策树，正则化

Boosting：XGBoost使用Boosting提升思想对多个弱学习器进行迭代式学习

二阶泰勒展开：每一轮学习中，XGBoost对损失函数进行二阶泰勒展开，使用一阶和二阶梯度进行优化。

决策树：在每一轮学习中，XGBoost使用决策树算法作为弱学习进行优化。

正则化：在优化过程中XGBoost为防止过拟合，在损失函数中加入惩罚项，限制决策树的叶子节点个数以及决策树叶子节点的值。

## 个人总结

|  |
| --- |
| 集成学习是多个模型一起预测。  boosting：随机抽取数据，组成多个模型，加权投票，变强，解决欠拟合。  bagging：随机抽取数据，组成多个模型，平权投票，变稳，解决过拟合。 |

# 聚类算法

## 简介

### 应用

* 用户画像，广告推荐，Data Segmentation，搜索引擎的流量推荐，恶意流量识别
* 基于位置信息的商业推送，新闻聚类，筛选排序
* 图像分割，降维，识别；离群点检测；信用卡异常消费；发掘相同功能的基因片段

### 概念

聚类算法是典型的无监督学习算法，主要作用将相似的样本自动归到一个类别中。

有监督学习 —— 特征值离散 —— 分类 —— K邻近算法、逻辑回归、决策树

* 特征值连续 —— 回归 —— 线性回归

无监督学习 —— 聚类算法

## api介绍

### api

sklearn.cluster.KMeans(n\_clusters=8)

参数:

n\_clusters:开始的聚类中心数量

整型，缺省值=8，生成的聚类数，即产生的质心（centroids）数。

方法:

estimator.fit(x)

estimator.predict(x)

estimator.fit\_predict(x)

计算聚类中心并预测每个样本属于哪个类别,相当于先调用fit(x),然后再调用predict(x)

### 案例

|  |
| --- |
| **import** matplotlib.pyplot **as** plt **from** sklearn.datasets.\_samples\_generator **import** make\_blobs **from** sklearn.cluster **import** KMeans **from** sklearn.metrics **import** calinski\_harabasz\_score  *# 创建数据集 # x为样本特征值，y为样本簇类别* x, y = make\_blobs(n\_samples=1000, *# 样本数量* n\_features=2, *# 特征类别数量* centers=[[-1, -1], [0, 0], [1, 1], [2, 2]], *# 围绕的簇中心数量和位置* cluster\_std=[0.4, 0.2, 0.2, 0.2], *# 簇方差* random\_state=9) plt.scatter(x[:,0], x[:,1], marker=**'+'**) *#* plt.show()  y\_pred = KMeans(n\_clusters=2,random\_state=9).fit\_predict(x) plt.scatter(x[:,0], x[:,1],c=y\_pred) plt.show()  *# CH评估聚类分数* print(calinski\_harabasz\_score(x, y\_pred)) |

## 实现流程

### 流程

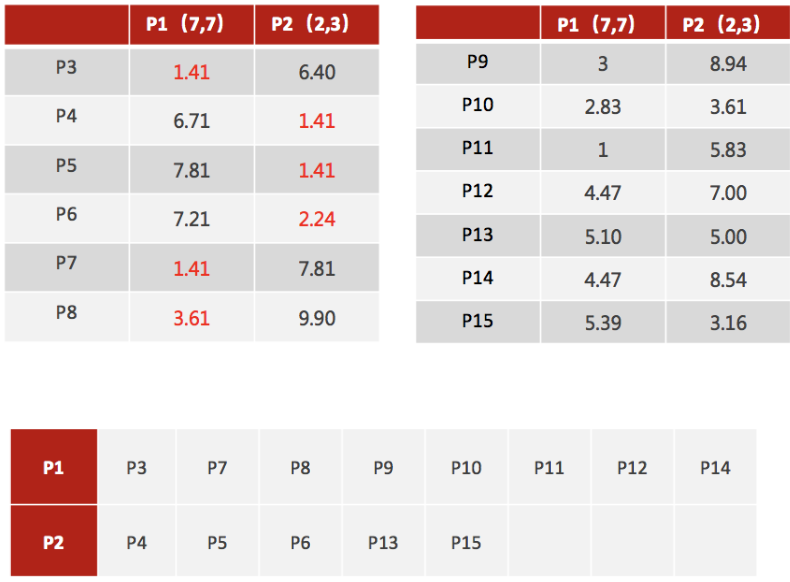
* 1、随机设置K个点做为初始的聚类中心
* 2、计算其他点到聚类中心距离。其他点到哪个聚类中心近则分类给哪个聚类中心。
* 3、对计算后的聚类计算平均值，得到下一轮的聚类中心
* 4、继续循环步骤2到3，直到新的聚类中心和上次的一致

### 案例

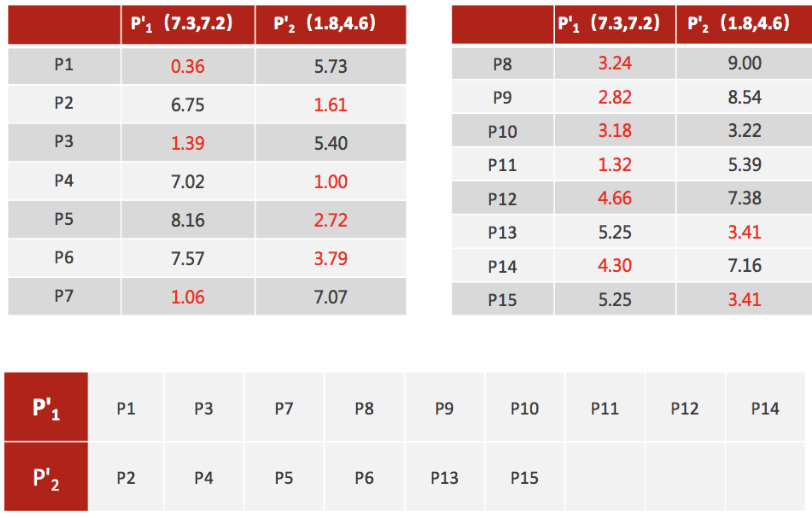


解：

随机取P1和P2做为聚类中心，求距离，分类，如图。



求平均值得新的聚类中心。重新计算距离，再分类



## 模型评估

### 误差平方和

误差平方和(SSE \The sum of squares due to error)



* p为特征点，m为聚类中心。C为类别集合。
* k为类别数。
* 如果质心的初始值选择不好,SSE只会达到一个不怎么好的局部最优解

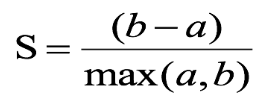
### 肘方法—确定k值（样本中心）

（1）对于n个点的数据集，迭代计算k from 1 to n，每次聚类完成后计算每个点到其所属的簇中心的距离的平方和；

（2）平方和是会逐渐变小的，直到k==n时平方和为0，因为每个点都是它所在的簇中心本身。

（3）在这个平方和变化过程中，会出现一个拐点也即“肘”点，下降率突然变缓时即认为是最佳的k值。

### 轮廓系数法(Silhouette Coefficient)



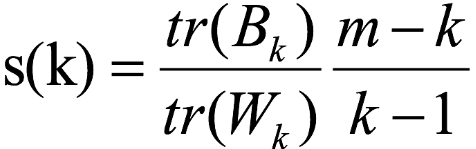
a：样本i到同一个聚类中心的其他点的距离的平均值

b：样本i到另外所有的聚类中心的簇内点的平均值里最小值（离样本i最近的其他簇）

同样可求出平均轮廓系数S，它取值[-1,1]，越大越好

**目的：内部距离最小化，外部距离最大化。如果两个K值的S相差不大，优先K大的。**

### CH系数法(Calinski-Harabasz Index)



* tr为矩阵的迹, 定义为矩阵斜对角线的和，可以表示一个物体的相似性
* Bk为类别之间的协方差矩阵，Wk为类别内部数据的协方差矩阵;
* m为训练集样本数，k为类别数。
* 分母越大，分子越小，越好。类别之间差别大，类别之内差别小。

**目的：用尽量少的类别聚类尽量多的样本，同时获得较好的聚类效果。**

## 模型优化

### 算法小结

k-means算法小结

优点：

​ 1.原理简单（靠近中心点），实现容易

​ 2.聚类效果中上（依赖K的选择）

​ 3.空间复杂度o(N)，时间复杂度o(IKN)

（N为样本点个数，K为中心点个数，I为迭代次数）

缺点：

​ 1.对离群点，噪声敏感 （中心点易偏移）

​ 2.很难发现大小差别很大的簇及进行增量计算

​ 3.结果不一定是全局最优，只能保证局部最优（与K的个数及初值选取有关）

### canopy配合初始聚类

|  |
| --- |
| 作用：为聚类算法选K值。  Canopy算法的实现过程可以用以下流程图来表示：  1.给定样本列表 L=x\_1,x\_2,…,x\_m 以及初始距离阈值为 T1、T2，且（T1>T2） (T1、T2可自己定义)；  2.从列表L中任取一点P，计算P到所有聚簇中心点的距离 (如果不存在聚簇中心，那么.就把点P作为一个新的聚簇)，并选出与聚类中心最近的距离 D (P,a\_j) ；  3.如果距离D小于T1，表示该节点属于该聚簇，添加到该聚簇列表中。  4.如果距离D小于T2，表示该节点不仅仅属于该聚簇，还表示和当前聚簇中心点非常近，所以将P从列表L中删除。  5.如果距离D大于T1，那么节点P形成一个新的聚簇。  6.重复步骤2-5，直到列表L为空时结束循环。    优点：  ​ 1.Kmeans对噪声抗干扰较弱，通过Canopy对比，将较小的NumPoint的Cluster直接去掉有利于抗干扰。  ​ 2.Canopy选择出来的每个Canopy的centerPoint作为K会更精确。  ​ 3.只是针对每个Canopy的内做Kmeans聚类，减少相似计算的数量。  缺点：  ​ 1.算法中 T1、T2的确定问题 ，依旧可能落入局部最优解 |

### K-means++

|  |
| --- |
| 作用：为聚类算法选质心位置。  K-means++算法是一种为k-means聚类算法选择初始值（或“种子”）的算法。它是一种避免标准k-means算法有时发现的较弱聚类的方法1。  K-means++算法的步骤如下所示：  1、随机选取一个中心点 a\_1；  2、计算数据到之前 n 个聚类中心最远的距离D (x)，并以一定概率P选择新中心点 a\_i ；  这种方法可以使得距离当前n个聚类中心越远的点会有更高的概率被选为第n+1个聚类中心3。 |

### 二分k-means

|  |
| --- |
| 作用：二分k-means是对k-means算法的优化。它的主要优化之处在于选择质心时，二分k-means算法有效地避免了初始选择质心时的误差，可以有效地提高算法效率.  实现流程:  1.所有点作为一个簇  2.将该簇一分为二  3.选择能最大限度降低聚类代价函数（也就是误差平方和）的簇划分为两个簇。  4.以此进行下去，直到簇的数目等于用户给定的数目k为止。    隐含的一个原则:  因为聚类的误差平方和能够衡量聚类性能，该值越小表示数据点越接近于他们的质心，聚类效果就越好。所以需要对误差平方和最大的簇进行再一次划分，因为误差平方和越大，表示该簇聚类效果越不好，越有可能是多个簇被当成了一个簇，所以我们首先需要对这个簇进行划分。  二分K均值算法可以加速K-means算法的执行速度，因为它的相似度计算少了并且不受初始化问题的影响，因为这里不存在随机点的选取，且每一步都保证了误差最小 |

### k-medoids

|  |
| --- |
| 实现流程：  ( 1 )总体n个样本点中任意选取k个点作为medoids  ( 2 )按照与medoids最近的原则，将剩余的n-k个点分配到当前最佳的medoids代表的类中  ( 3 )对于第i个类中除对应medoids点外的所有其他点，按顺序计算当其为新的medoids时，代价函数的值，遍历所有可能，选取代价函数最小时对应的点作为新的medoids  ( 4 )重复2-3的过程，直到所有的medoids点不再发生变化或已达到设定的最大迭代次数  ( 5 )产出最终确定的k个类 |

### 总结



## 特征降维

### 简介

* 定义：降维是指在某些限定条件下，降低随机变量(**特征**)个数，得到一组“**不相关**”主变量的过程
* 方式：特征选择；主成分分析；

### 特征选择

* 目的：从原有特征中找出主要特征
* 方法：

Filter(过滤式)：主要探究特征本身特点、特征与特征和目标值之间关联

方差选择法：低方差特征过滤

相关系数

Embedded (嵌入式)：算法自动选择特征（特征与目标值之间的关联）

决策树:信息熵、信息增益

正则化：L1、L2

深度学习：卷积等

1. 低方差过滤

|  |
| --- |
| 删除方差低的特征值；  api:  sklearn.feature\_selection.VarianceThreshold(threshold = 0.0)  Variance.fit\_transform(X)  X:numpy array格式的数据[n\_samples,n\_features]  返回值：训练集差异低于threshold的特征将被删除。默认值是保留所有非零方差特征，即删除所有样本中具有相同值的特征。  def variance\_demo():  """  删除低方差特征——特征选择  :return: None  """  data = pd.read\_csv("factor\_returns.csv")  print(data)  # 1、实例化一个转换器类  transfer = VarianceThreshold(threshold=1)  # 2、调用fit\_transform  data = transfer.fit\_transform(data.iloc[:, 1:10])  print("删除低方差特征的结果：\n", data)  print("形状：\n", data.shape)  return None |

1. 皮尔逊相关系数

|  |
| --- |
| 相关系数的值介于–1与+1之间，即–1≤ r ≤+1。其性质如下：  当r>0时，表示两变量正相关，r<0时，两变量为负相关  当|r|=1时，表示两变量为完全相关，当r=0时，表示两变量间无相关关系  当0<|r|<1时，表示两变量存在一定程度的相关。且|r|越接近1，两变量间线性关系越密切；|r|越接近于0，表示两变量的线性相关越弱  一般可按三级划分：|r|<0.4为低度相关；0.4≤|r|<0.7为显著性相关；0.7≤|r|<1为高度线性相关  api：  from scipy.stats import pearsonr  x : (N,) array\_like  y : (N,) array\_like Returns: (Pearson’s correlation coefficient, p-value) |

1. 斯皮尔曼相关系数

|  |
| --- |
| 斯皮尔曼相关系数表明 X (自变量) 和 Y (因变量)的相关方向。 如果当X增加时， Y 趋向于增加, 斯皮尔曼相关系数则为正  与之前的皮尔逊相关系数大小性质一样，取值 [-1, 1]之间  斯皮尔曼相关系数比皮尔逊相关系数应用更加广泛  api：  from scipy.stats import spearmanr |

### 主成分分析

* 定义：高维数据转化为低维数据的过程，在此过程中可能会舍弃原有数据、创造新的变量
* 作用：是数据维数压缩，尽可能降低原数据的维数（复杂度），损失少量信息。
* 应用：回归分析或者聚类分析当中

|  |
| --- |
| sklearn.decomposition.PCA(n\_components=None)  将数据分解为较低维数空间  n\_components:  小数：表示保留百分之多少的信息  整数：减少到多少特征  PCA.fit\_transform(X) X:numpy array格式的数据[n\_samples,n\_features]  返回值：转换后指定维度的array |

### 个人**总结**

|  |
| --- |
| 流程：  1、随机设置K个点做为初始的聚类中心  2、计算其他点到聚类中心距离。其他点到哪个聚类中心近则分类给哪个聚类中心。  3、对计算后的聚类计算平均值，得到下一轮的聚类中心  4、继续循环步骤2到3，直到新的聚类中心和上次的一致 |