# 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。 |

## 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

# 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

过拟合：判断错误

# 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>

# Matplotlib

## 基础绘图

* =======================绘图流程==========================
* 导入 import matplotlib.pyplot as plt
* 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 = '-.')
* 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) 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>
* =====================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)  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 |

### 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 |

### 数组去重

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

### 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.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 |

### 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 |

### 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 |

### 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。 |

## 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

|  |
| --- |
| # 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]  # 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))  ========================删除缺失值==========================  data = movie\_var.dropna()  ========================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) |

## 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)  可以指定按照两组数据的共同键值对合并或者左右各自  left: DataFrame  right: 另一个DataFrame  on: 指定的共同键  how:按照什么方式连接，可选‘left’,’right’,’outer’,或者默认  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()  ======================**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()  #################### pivot\_table（）###################  index：索引，可以设置多层，传参列表，相当于excel的行筛选  values：列对应的值，可以设置多个列的值，相当于excel的列筛选  aggfunc：对数据聚合时进行的函数操作，默认mean，可以设置多种 |

## 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() |

# 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)  # 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。

# 线性回归

## 简介

### 应用场景

* 房价预测

### 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 |

## api

### api

|  |
| --- |
| sklearn.linear\_model.LinearRegression()  LinearRegression.coef\_：回归系数 |

### 实例

|  |
| --- |
| 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

1. 损失函数

|  |
| --- |
|  |

1. 正规方程

|  |
| --- |
|  |