大数据技术

说明

本提纲由 智造钱2201 李祎航 编写，内容适用于智造钱课程《大数据技术》。

CH1 概论

1.1 大数据发展历史

1.2 大数据特征与概念

1. 特征：Volume(大量) Variety(多样性) Velocity(快速性) Veracity(真实性)
2. 分类：结构化、非结构化、半结构化
3. **概念**：一种规模大到在获取、存储、管理、分析方面大大超出了传统数据库软件工具能力范围的**数据集合**，需要新处理模式才能具有更强的决策力、洞察发现力和流程优化能力来适应海量、高增长率和多样化的**信息资产**
4. 核心：大数据的核心不仅在于掌握庞大的数据信息，更在于对这些含有意义的数据进行专业化处理，以提升数据的价值

1.3 大数据技术

1. 谷歌：分布式文件系统、分布式计算框架、分布式数据存储系统

1.4 大数据技术应用

1. 搜索引擎
2. 大数据分析
3. 机器学习与人工智能
4. 行业：交通、金融、教育、健康、制造、网络

1.5 大数据安全

1. 风险：泄露用户资料、数据造假、垄断、监听
2. 威胁因素：硬件损坏、人为错误、电源故障、自然灾害、黑客、病毒、窃取
3. 保护方法：数据资产管理、数据库安全、数据库加密、数据脱敏、数据库漏洞扫除

1.6 工业大数据

1.6.1 分类

1. 制造过程数据

2. 制造执行系统数据
3. 企业运营管理相关的业务数据：企业资源计划、产品生命周期、供应链、客户关系、能耗管理
4. 企业外部数据：产品使用、客户名单、供应商、运营情况

1.6.2 属性

1. 属性：价值属性、产权属性

1.6.3 特点

1. 针对特定工业场景
2. 以大数据集为基础
3. 集成工业大数据技术与方法
4. 获得有价值的信息
5. 提高产品竞争力
6. 实现运营提质和管理增效

1.7 智能制造

1.7.1 定义

智能制造是一种由智能机器和人类专家共同组成的人机 一体化智能系统，它在制造过程中能进行智能活动，诸如分析、推理、判断、构思和决策等。通过人与智能机器的合作共事，去扩大、延伸和部分地取代人类专家在制造过程中的脑力劳动。

1.7.2 案例

1. 日本智能制造系统
2. 欧洲信息技术研究发展战略计划
3. 加拿大发展战略计划
4. 美国先进制造业国家战略计划
5. 德国工业4.0
6. 中国制造2025

1.7.3 特点

1. 自动化：制造执行系统、人机料法环
2. 数字化：设计、仿真、实验；供应链、渠道、客户；产品能耗、效能、寿命
3. 智能化：控制与工艺；生产计划；远程监控与健康管理；节能管理

1.7.4 工业大数据与智能制造

1. 工业大数据描述了智能制造各生产阶段的真实情况，为人类读懂、分析和优化制造提供了宝贵的数据资源，是实现智能制造的智能来源。工业大数据、人工智能模型和机理模型的结合，可有效提升数据的利用价值，是实现更高阶的智能制造的关键技术之一。
2. 智能制造是工业大数据的载体和产生来源，其各环节信息化、自动化系统所产生的数据构成了工业大数据的主体。另一方面，智能制造又是工业大数据形成的数据产品最终的应用场景和目标。

1.8 工业大数据关键技术

1.8.1 数据采集与传输：

1. 技术：PLC；监测控制和数据采集；数据传输
2. 挑战：工业环境中数据获取手段受限；时序数据质量问题

1.8.2 数据存储与管理

1. 多源异构数据
2. 数据管理引擎

1.8.3 数据分析技术

1. 时序模式分析技术
2. 工业知识图谱技术
3. 多源数据融合分析技术

1.9 工业互联网

1. 定义：工业互联网是新一代信息通信技术与工业经济深度融合的新型基础设施、应用模式和工业生态，通过对人、机、物、系统等的全面连接，构建起覆盖全产业链、全价值链的全新制造和服务体系，为工业乃至产业数字化、网络化、智能化发展提供了实现途径，是第四次工业革命的重要基石。
2. 内涵：网络为基础、平台为中枢、数据为要素、安全为保障
3. 外延：是工业数字化、网络化、智能化转型的基础设施，也是互联网、大数据、人工智能与实体经济深度融合的应用模式。

CH2 Python基础

2.1 引言与环境

2.1.1 为什么学编程

1. 计算机多
2. 未来语言
3. 创新时代

4. 就业机遇

2.1.2 程序

1. 程序：做一件事或解决一个问题所采取的一系列时序步骤
2. 程序是人与计算机对话的语言

2.1.3 程序设计语言的发展

1. 机械编程——机器语言——汇编语言——高级语言

2.1.4 程序思维

2.1.5 Python

1. 为什么用py处理大数据：社区、软件包、不需要底层细节

2.2 Python语言基础

2.2.1 变量和简单数据类型

1. 变量：编程中基础的储存单位
2. 字符串
3. 数字
4. 原地运算：

```
b = 2
b += 2
b *= 2
b -= 2
```

5. 基本运算符

```
+ - * /
% 取模
** 幂运算
// 取余
```

6. Python赋值机制：Python并没有使用新的内存来储存变量 y 的值，而是在命名空间中，让变量 y 与变量 x 指向了同一块内存空间

```
x = [1,2,3]
y = x
```

```
x[0] = 10

print(y)
[10, 2, 3]

# 可以使用 is 来判断变量是不是指向同一事务
x is y

True
```

7. print
8. python数据类型：
    1. 常用：float、int、string、列表、字典、numpy数组
    2. 其他：长整型、bool、元组、集合、df

2.2.2 列表

1. 列表 `k = [1, 2.0, 'hello']`
    1. 是一个有序序列
    2. 用一对`[]`生成，用`,`隔开
    3. 元素不需要是同一类型
    4. 长度不固定
    5. 空列表生成`empty_list = []; empty_list = list()`
    6. 查看长度`len(k)`
    7. 列表加法与乘法

```
a = [1, 2, 3]
b = [3.2, 'hello']
a + b => [1, 2, 3, 3.2, 'hello']

b * 2 => [3.2, 'hello', 3.2, 'hello']
```

2. 修改、添加、删除元素

```
a = [1002, 'a', 'b', 'c']
del a[0]    ===>  ['a', 'b', 'c']
del a[1:]   ===>  [1002]
del a[::2]  ===>  ['a', 'c']  #删除第1个及第1+ni个元素

b = [10,11,12,11]
b.extend([1, 2])   ===>   [10,11,12,11,1,2]
```

```
b.insert(3, 'a')   ===>   [10,11,12,'a',11]

b.remove(11)       ===>   [10,12,11]   # 移除第一个
```

3. 列表方法

```
a = [11,12,13,12,11]

a.count(11)        ===>   2    #某元素的个数
a.index(12)        ===>   1    #某元素第一次出现的位置
a.append(1)        ===>   [11,12,13,12,11,1]
a.append([11,12])  ===>   [11,12,13,12,11,[11,12]]

a.sort()     #排序
a.sorted()   #还原
```

2.2.3 操作列表

1. 从0开始索引
2. 负向索引：-2为倒数第2个
3. 分片：

```
var[lower:upper:step]   ===>  [lower, upper)

s = "hello world"
s[1:3]   ===> 'el'
s[1:-2]  ===> 'ello wor'

# lower和upper可以省略，分别意味着从头开始和一直到结尾
s[:3]   ===>  'hel'
s[-3:]  ===>  'rld'
s[:]    ===>  'hello world'
s[::2]  ===>  'hlowrd'

#step为负数，省略lower意味着从结尾开始分片，省略upper意味着分片到开头
s[::-1]  ===>  'dlrow olleh'

$当upper超出长度，不会报错，会计算到结尾
s[:100]  ===>  'hello world'
```

4. 修改

```
a = [10,11,12,13,14]

a[0] = 100          ===>     [100,11,12,13,14]
a[1:3] = [1,2]      ===>     [10,1,2,13,14]
a[1:3] = [1,2,3,4]  ===>     [10,1,2,3,4,13,14] #长度可以不一致
```

5. 元组（tuple） `t = (1,2,3,4)` 有序序列，有分片和索引，但不可变

2.2.4 判断语句

```
if condition:
    do something
elif condition:
    do
else:
    do

可以用and or not进行连接
```

2.2.5 循环语句

1. while

```
while condition:
    do
```

2. for

```
for item in iterable:
    do

range(n) ===> 0,1,2,3,...,n-1
range(lower,upper,step)
```

3. continue/break：跳出本轮循环/跳出整个循环

```
values = [7,6,4,9,2]
for i in values:
    if i % 2 != 0:
        continue
    print(i/2)
```

```
3.0
2.0
1.0

num_list = [1,2,3,4,5,6,7,8,9]
for i in num_list:
    if i==5:
        break
    print(i)

1,2,3,4
```

4. else：要与break一起使用：循环被break结束时，循环条件仍满足，else不执行；循环正常结束，循环条件不足，else执行

```
values = [19,20,21,7,23]
for i in values:
    if i <=10:
        print(i)
        break
else:
    print("均大于10")

==========>
7

values = [19,20,21,17,23]
for i in values:
    if i <=10:
        print(i)
        break
else:
    print("均大于10")

==========>
均大于10
```

2.2.6 函数

1. 简介：通常接受输入参数，并有返回值。它负责完成某项特定任务，而且相较于其他代码，具备相对的独立性。使用函数时，只需要将参数换成特定的值传给函数。Python并没有限定参数的类型，因此可以使用不同的参数类型。

```
def function(arg1, agr2):
    ......
    return result

function(arg1=a, arg2=b)
funciton(a, b)
```

2. 传参：第一种是使用的按照位置传入参数，另一种则是使用关键词模式，显式地指定参数的值；可以混合这两种模式；设定参数默认值：可以在函数定义的时候给参数设定默认值；可以修改默认值。
3. 接受不定参数：
    1. `def add(x, *args):` * *args* 表示参数数目不定，相当于一个元组
    2. `def add(x, **kwargs):` ** *kwargs* 表述参数数目不定，相当于一个字典
4. lambda简化函数：
    1. lambda函数是匿名的： lambda函数没有名字。
    2. lambda函数有输入和输出：输入是传入到参数列表argument_list的值，输出是根据表达式expression计算得到的值。
    3. lambda函数一般功能简单：单行expression决定了lambda函数不可能完成复杂的逻辑，只能完成非常简单的功能。

```
lambda argument_list: experssion
    g = lambda x: x + 1
```

2.2.7 class

1. 创建和使用类：
    1. 类：具有相同或相似性质的对象的抽象
    2. 编写：定义一大类对象都有的通用行为。基于类创建对象时，每个对象都自动具备这种通用行为，然后可以根据需要赋予每个对象独特的个性。
    3. 基本语法：`class Name:`
    4. 继承：一个类继承另一个类时，将自动获得另一个类的所有属性和方法。原有的类称为父类，新的类称为子类。• 子类继承了父类的所有属性和方法，同时还可以定义自己的属性和方法。
    5. 导入：Python允许将类储存在模块中，然后在主程序中导入所需的模块，使文件尽可能简洁

2.2.8 字典

1. 定义：由键值对组成的数据结构
2. 创建：

```
a = {}
a = dict()
```

3. 插入、查看、更新键值

```
a["one"] = "number 1"
a["two"] = "number 2"

{'one':'number 1', 'two':'number 2'}

a['one']  ===>  'number 1'

a['one'] = 'number 3'
a ===> {'one':'number 3', 'two':'number 2'}
```

4. 初始化字典
5. 字典没有顺序：不能用数字索引查看键值
6. 键值的数据类型可以不同；int和string比较常用，不用float；也可使用元组=>有序的
7. 字典方法

```
update:
person = {}
person["name"] = "James"
person["sex"] = "male"
person["age"] = "18"

person_new = {'name':'Jim', 'last':'Maxwell'}
person.updata(person_new)
====> {'name':'Jim', 'male':'man', 'age':'18', 'last':'Maxwell'}

d.keys()     #返回一个由所有键组成的列表
d.values()   #返回一个由所有值组成的列表
d.items()    #返回一个由所有键值对元组组成的列表
```

8. 字典应用场景：集中管理数据信息

2.2.9 模块与工具包

1. 模块：.py文件
2. 包：foo文件夹，内有**init**.py（可为空）

2.2.10 文件与异常

1. 库
2. 读文件：open/file/read/readline；使用完需close
3. 写文件：open/write/close
4. 追加模式：open('file_name', 'a')

2.3 Python的集成包

2.3.1 Numpy

2.3.2 Pandas

2.3.3 Matplotlib

2.4 数据清洗与预处理

2.4.1 数据清洗

1. 定义：理顺杂乱的原始数据，并修正数据中的错误，这一步比较繁杂，但确是整个分析的基石；是从记录表、表格、数据库中检测、纠正或删除损坏或不准确记录的过程；数据清洗就是将“污染数据”变成“干净数据”过程
2. 污染数据：残缺数据、 错误数据、 重复数据、不符合规则的数据
3. 干净数据：可以直接带入模型的数据

2.4.2 数据预处理

读写——探索与描述——简单处理——重复值——缺失值——异常值——文本数据——时序数据

2.5 使用Scikit_learn构建模型

2.5.1 使用sklearn转换器进行数据预处理与降维

1. 数据集加载`load_diabetes`
2. 数据集划分：train、val、test：`train_test_split`
3. k折验证
4. 转换器三个方法：*fit(), transform(), fit_transform()*
5. 对数据的处理：标准化、归一化、二值化、独热变量、PCA降维

2.5.2 用sklearn估计器构建聚类模型

1. 聚类的输入是一组未被标记的样本，聚类根据数据自身的距离或相似度将它们划分为若干组，划分的原则是组内（内部）距离最小化，而组间（外部）距离最大化。

2. 常见聚类算法：

算法类别	包括的主要算法
划分（分裂）方法	K-Means算法（K-平均）、K-MEDOIDS算法（K-中心点）和CLARANS算法（基于选择的算法）
层次分析方法	BIRCH算法（平衡迭代规约和聚类）、CURE算法（代表点聚类）和CHAMELEON算法（动态模型）
基于密度的方法	DBSCAN算法（基于高密度连接区域）、DENCLUE算法（密度分布函数）和OPTICS算法（对象排序识别）
基于网格的方法	STING算法（统计信息网络）、CLIOUE算法（聚类高维空间）和WAVE-CLUSTER算法（小波变换）

3. 实现：
    1. 需要sklearn估计器：estimator
    2. 用有 fit() 和 predict() 两个方法
4. 评价模型：

评价指标名称	真实值	最佳值	sklearn函数
ARI评价法（兰德系数）	需要	1.0	adjusted_rand_score
AMI评价法（互信息）	需要	1.0	adjusted_mutual_info_score
V-measure评分	需要	1.0	completeness_score
FMI评价法	需要	1.0	fowlkes_mallows_score
轮廓系数评价法	不需要	畸变程度最大	silhouette_score
Calinski-Harabasz指数评价法	不需要	相较最大	calinski_harabaz_score

    1. 除了轮廓系数评价法以外的评价方法，在不考虑业务场景的情况下都是得分越高，其效果越好，最高分值为1。
    2. 而轮廓系数评价法则需要判断不同类别数目情况下的轮廓系数的走势，寻找最优的聚类数目。
    3. 综合以上聚类评价方法，在真实值作为参考的情况下，几种方法均可以很好地评估聚类模型。
    4. 在没有真实值作为参考的时候，轮廓系数评价法和Calinski-Harabasz指数评价法可以结合使用。

2.5.3 使用skleran估计器构建分类模型

1. 在数据分析领域，分类算法很多，其原理千差万别，常用算法如下

模块名称	函数名称	算法名称
linear_model	LogisticRegression	逻辑斯蒂回归
svm	SVC	支持向量机
neighbors	KNeighborsClassifier	K最近邻分类
naive_bayes	GaussianNB	高斯朴素贝叶斯
tree	DecisionTreeClassifier	分类决策树
ensemble	RandomForestClassifier	随机森林分类
ensemble	GradientBoostingClassifier	梯度提升分类树

2. 评价分类模型

分值越高越好，其使用方法基本相同

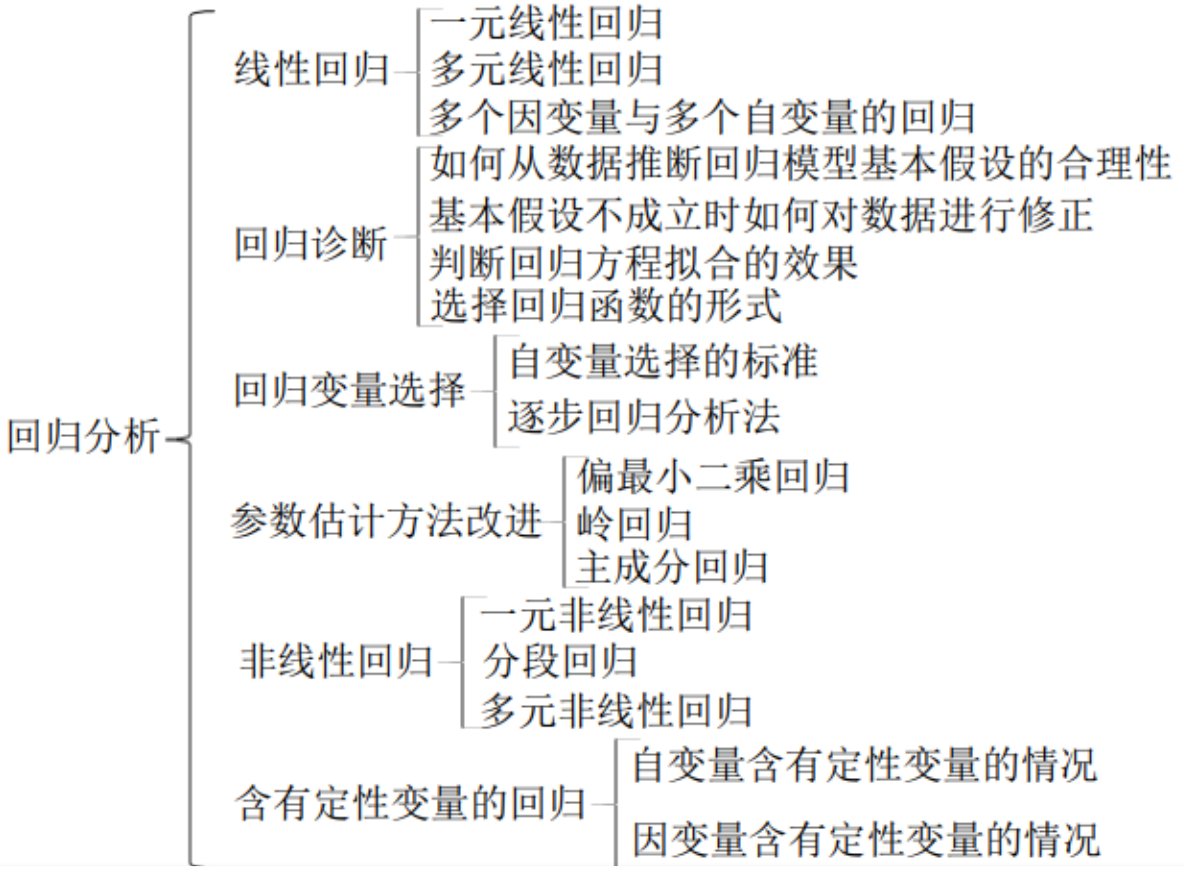
方法名称	最佳值	sklearn函数
Precision（精确率）	1.0	metrics.precision_score
Recall（召回率）	1.0	metrics.recall_score
F1值	1.0	metrics.f1_score
Cohen's Kappa系数	1.0	metrics.cohen_kappa_score
ROC曲线	最靠近y轴	metrics. roc_curve

通常情况下，ROC曲线与x轴形成的面积越大，表示模型性能越好

2.5.4 使用sklearn估计器构建回归模型

1. 回归模型：从19世纪初高斯提出最小二乘估计法算起，回归分析的历史已有200多年。从经典的回归分析方法到近代的回归分析方法，按照研究方法划分，回归分析研究的范围大

致如图所示：



2. 回归模型算法：

模块名称	函数名称	算法名称
linear_model	LinearRegression	线性回归
svm	SVR	支持向量回归
neighbors	KNeighborsRegressor	最近邻回归
tree	DecisionTreeRegressor	回归决策树
ensemble	RandomForestRegressor	随机森林回归
ensemble	GradientBoostingRegressor	梯度提升回归树

    1. 在回归模型中，自变量与因变量具有相关关系，自变量的值是已知的，因变量是要预测的
    2. 回归算法的实现步骤和分类算法基本相同，分为学习和预测两个步骤
    3. 学习是通过训练样本数据来拟合回归方程的
    4. 预测则是利用学习过程中拟合出的回归方程，将测试数据放入方程中求出预测值

3. 回归模型评价：回归模型的性能评价不同于分类模型，虽然都是对照真实值进行评价，但由于回归模型的预测结果和真实值都是连续的，所以不能够求取Precision、 Recall和F1值

等评价指标。回归模型拥有一套独立的评价指标。

方法名称	最优值	sklearn函数
平均绝对误差	0.0	metrics. mean_absolute_error
均方误差	0.0	metrics. mean_squared_error
中值绝对误差	0.0	metrics. median_absolute_error
可解释方差值	1.0	metrics. explained_variance_score
R方值	1.0	metrics. r2_score

值越靠近1，模型性能越好

值越靠近0，模型性能越好

CH3 统计分析

3.1 概述

3.1.1 数据的重要角色

1. 利用大数据驱动的方法解决智能问题和决策问题已经成为多个领域的共识，数据通常并不能直接被人们利用。 如何从大量看似杂乱无章的数据中，发掘有用的知识、 揭示其中隐含的内在规律， 指导人们进行科学的推断与决策。
2. 数据分析可赋予新的方法论：数据科学中兴起另一种方法论——“问题→数据→问题”，根据“问题”找“数据”，并直接用“数据”（在不需要把“数据”转换成“知识”的前提下）解决“问题”
3. 数据分析可赋予我们洞察未来的能力

3.1.2 数据分析框架

问题分析——数据理解——数据准备——建立模型——模型评估——应用

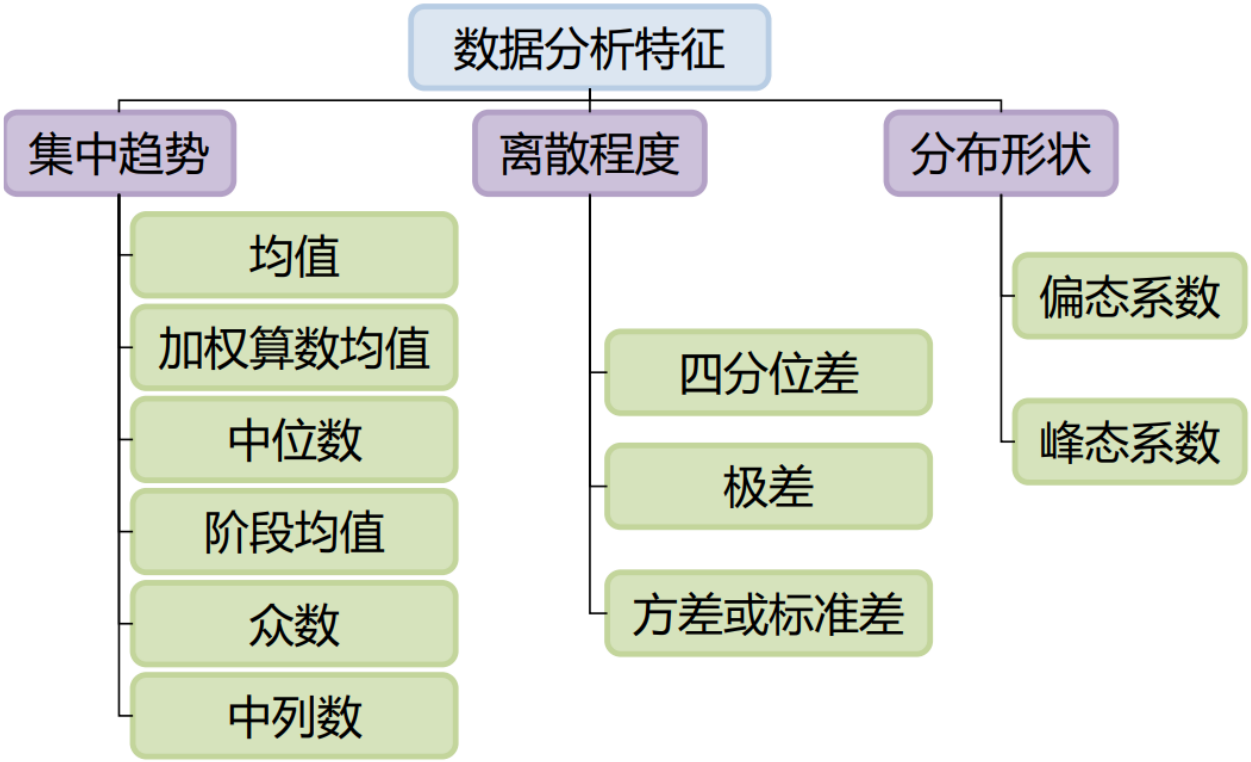
3.1.3 数据分析方法

1. 小数据分析方法：
    1. 分析数据和建模：
    2. 统计数据分析方法：
        1. 数据描述性分析
        2. 数据推理性分析
        3. 数据探索性分析
    3. 基于机器学习的数据分析
2. 大数据的新挑战：
    1. 数据量大并不意味着数据价值的增加，相反往往意味着数据噪音的增多
    2. 大数据时代的算法需要进行调整
    3. 数据结果的衡量标准
3. 新认识：从统计学到数据科学：

1. 数据结果的衡量标准
2. 不是纯净性，而是混杂性：这就是从”小数据“到”大数据“的改变
3. 不是精确性，而是趋势
4. 不是因果关系，而是相关关系
5. 不是采样的，而是全量的

3.2 数据描述性分析

1. 获取到数据后， 第一时间往往是需要从宏观角度来观察数据，也就是分析数据的特征。能够概括数据位置特性、 分散性、关联性等数字特征，以及能够反映数据整体分布特征的分析方法，称为数据描述性分析。
2. 数据分布的特征：

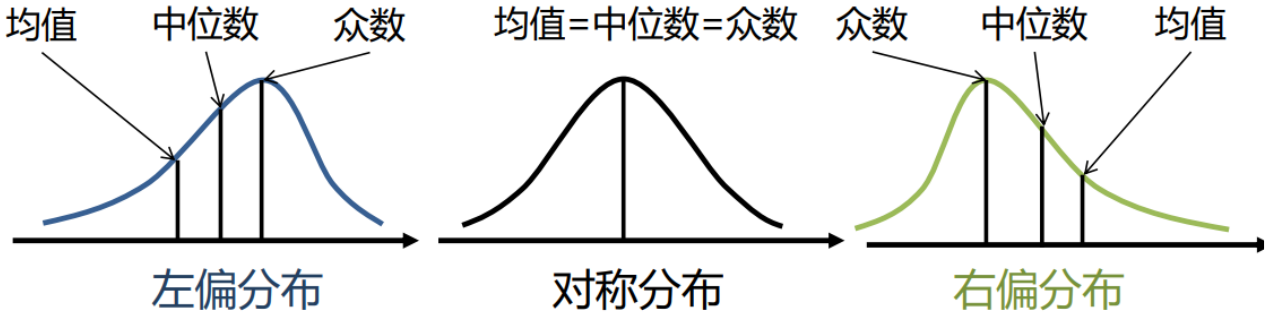


3.2.1 中心趋势度量 / 集中趋势测度

1. 定义：一组数据向其中心值靠拢的倾向和程度；中心趋势度量就是寻找数据水平的代表值或中心值；对于不同类型的数据, 可以采用不同的集中趋势测度值
2. 均值（mean）：均值是度量数据集中心的最常用、最有效的数值度量方法
3. 加权算术均值：集合中每个值与一个权值相关联，权值反映对应值的显著性、重要性或出现频率。在这种情况下，使用加权算数均值（weighted arithmetic mean）
4. 中位数：一组数据按从小到大(或从大到小)的顺序依次排列， 处在中间位置的一个数(或最中间两个数据的平均数)
    1. 排序后，处于中间位置上的值
    2. 不受极端值的影响
    3. 主要用于顺序数据，也可用数值型数据，但不能用于分类数据
    4. 相比均值， 中位数有着更好的抗干扰性

5. P分位数 M_p ： 表示排在序列长度p（0 ≤ p ≤ 1） 位置的数；其中四分位数（p=0.25和p=0.75 ） 最为常用
    1. 排序后，处于25%和75%位置上的值
    2. 不受极端值的影响
    3. 主要用于顺序数据， 也可用于数值型数据， 但不能用于分类数据
6. 截断均值：指定0和100间的百分数p， 丢弃高低两端各（p/2） 百分个数， 然后用常规方法计算均值， 所得到的结果即是截断均值。
    1. 目的： 可以抵消少数极端值的影响， 去掉最高和最低值影响
    2. 中位数是p=100%时的截断均值
    3. 均值对应于p=0%时的截断均值
7. 众数：一组数据中出现次数最多的数值
    1. 若一个样品中只有一个众数/峰就叫单峰；若有两个或两个以上的峰就叫双峰或多峰；其中最高的一个叫主峰,次高的叫次峰
    2. 众数是在统计分布上具有明显集中趋势点的数值， 代表数据的一般水平。
    3. 适合于数据量较多时使用， 不受极端值的影响。
    4. 不唯一：一组数据可能没有众数或有几个众数。
    5. 主要用于分类数据， 也可用于顺序数据和数值型数据。
8. 中列数：在统计中指的是数据集里最大值和最小值的算术平均
9. 众数、中位数、均值的特点：
    1. 众数：
        1. 不受极端值影响
        2. 缺点具有不唯一性
        3. 数据量较少时，不宜使用
        4. 主要适合作为 **分类数据** 的集中趋势测度值
    2. 中位数：
        1. 不受极端值影响
        2. 数据分布偏斜程度较大时应用
        3. 主要适合作为 **顺序数据** 的集中趋势测度值
    3. 均值：
        1. 易受极端值影响
        2. 数据对称分布或接近对称分布时应用
        3. 数据为偏态分布，特别是偏态程度较大时，中位数或众数代表性好

10. 分布：



左偏分布：说明数据存在极小值点，拉动平均值向极小值一方靠，而众数和中位数是位置代表值，不受极值的影响，三者的关系表现为：

$$\overline{x} < M < M_0$$

右偏分布：说明数据存在极大值点，拉动平均值向极大值一方靠，而众数和中位数是位置代表值，不受极值的影响，三者的关系表现为：

$$\overline{x} > M > M_0$$

3.2.2 数据的离散程度度量

反映各变量值远离其中心值的程度(离散程度)；从另一个侧面说明了集中趋势测度值的代表程度；不同类型的数据有不同的离散程度测度值

1. 四分位差：是对顺序数据离散程度的测度，也称为四分位距（interquartile range, IQR）
    1. 四分位差定义为上四分位数与下四分位数之差， 反映了中间50%数据的离散程度。
    2. 四分位差不受极端值的影响， 四分位差的数值越小，说明中间的数据越集中，其数值越大，说明中间的数据越分散
2. 极差：一组数据的最大值与最小值之差
    1. 极差是离散程度的最简单测度值
    2. 易受极端值影响
    3. 未考虑数据的分布
    4. 特点：计算简便，直观易于理解
3. 方差与标准差：方差 S^2 是用来反映这种数据分散程度的最常用的一种指标，反映了各变量值与均值的平均差异

$$S^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

3.2.3 数据分布度量

1. 偏度：称为偏态、偏态系数；是统计数据分布偏斜方向和程度的度量；是统计数据分布非对称程度的数字特征。

$$\text{根据原始数据计算：} SK = \frac{n\sum(x-\overline{x})^3}{(n-1)(n-2)s^3}$$

$$根据分组数据计算：SK=\frac{\sum_{i=1}^{k}(M_i-\overline{x})^3 f_i}{ns^3}$$

1. 偏度是数据分布偏斜程度的测度：偏度系数=0为对称分布；偏度系数> 0为右偏分布；偏度系数< 0为左偏分布
2. 偏度系数大于1或小于-1，被称为高度偏态分布
3. 偏度系数在0.5～ 1或-1～ -0.5之间，被认为是中等偏态分布
4. 偏度越接近0，分布更对称，偏斜程度低
5. 偏度大于零，比均值小的数据更多；

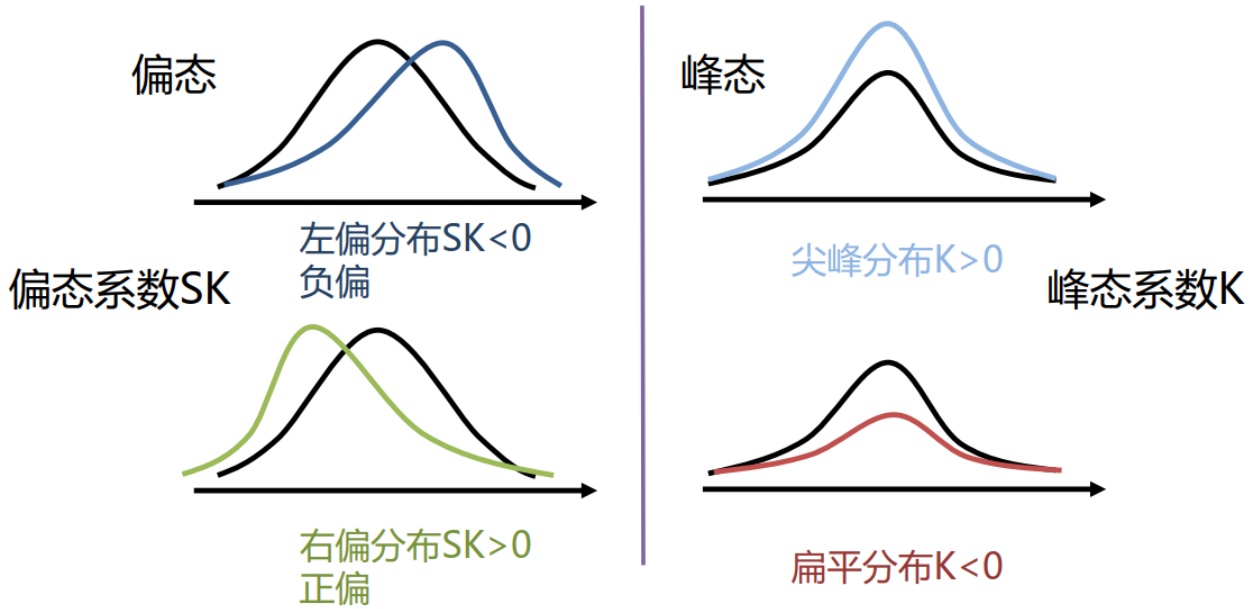
2. 峰度：也称为峰态， 用于数据分布扁平程度的测度

$$根据原始数据计算：K=\frac{n(n+1)\sum(x_i-\overline{x})^4-3[\sum(x_i-\overline{x})^2]^2(n-1)}{(n-1)(n-2)(n-3)s^4}$$

$$根据分组数据计算：K=\frac{\sum_{i=1}^{k}(M_i-\bar{x})^4 f_i}{ns^4}-3$$

1. 峰态通常是与标准正态分布相比较而言的;如果以正态分布作为标准， 不同分布的数据在均值附近的集中程度也不同;有的分布可能会显得“平坦” 一些， 有更多的数据分布在两侧;有的分布则看起来比较“尖锐” ， 数据更多地集中在均值附近
2. 峰态系数=0扁平峰度适中；峰态系数<0为扁平分布或者平峰分布；峰态系数>0为尖峰分布

3. 偏态与峰态分布的形状



3.2.4 图形化分析方法

1. 直方图：对于有限的数据，通过频率分布直方图来观察数据的分布， 直方图是频数直方图的简称。
   1. 用一系列宽度相等、高度不等的长方形表示数据的图。 长方形的宽度表示数据范围的间隔，长方形的高度表示在 给定间隔内的数据数。
   2. 适用范围：数据是数值型时；想弄清楚数据分布的形状；确定一个过程的输出是否近乎符合正态分布

2. 箱线图：由五个数值点组成：最小值 (min)，下四分位数(Q1)，中位数(median)，上四分位数(Q3)，最大值(max)

3.3 数据推理性分析

1. 描述统计是用整体的数据来描述整体特征， 而推理统计是用部分数据来推理整体特征。
2. 假设检验、 相关分析、 回归预测模型、 *贝叶斯模型* 都是推理性统计。

3.3.1 假设检验

1. 什么时候用假设检验：
    1. 当错误很明显的时候，不需要做假设检验。比如销售、运营的表现不好，可以直接取过往X天的销售业绩、运营指标来看。
    2. 有些问题不那么明显，比如上边吐槽的产品寿命问题。除非产品用到烂，否则不知道寿命是多少， 此时必须做抽样检测，就得用到假设检验方法。
    3. 在论证新点子很好的时候，也需要先把新点子做出来，再做小范围测试，此时也要用假设检验方法。
    4. ***假设检验方法适合于抽样检验/小范围测试的场景***
    5. 小概率反证法思想： 假设检验的核心思想是小概率反证法，在假设的前提下，估算某事件发生的可能性，如果该事件是小概率事件，在一次研究中本来是不可能发生的，现在发生了，这时候就可以推翻之前的假设，接受备择假设。如果该事件不是小概率事件，就找不到理由来推翻之前的假设，实际中可引申为接受所做的无效假设
2. 假设检验的步骤
    1. 提出原假设H0与备择假设H1
    2. 给定显著性水平 α，通常α=0.05, 0.01, 0.1等计算在此α 下H0成立条件下的临界值
    3. 确定拒绝域
    4. 由样本的观测值计算出检验统计量的具体值
    5. 计算统计量的样本观测值，与计算的临界值进行比较，如果计算结果落在拒绝域内，则拒绝原假设 H0，否则接受原假设

3.3.2 相关分析

在数据处理中， 一般将描述和分析两个或两个以上变量之间相关的性质及其相关程度的过程， 称为相关分析。

1. 解决问题：
    1. 确定变量之间是否存在相关关系， 如果存在一定关系， 找出它们依存关系的适当数学模型。
    2. 根据一个或几个变量值， 预测或控制另一个变量的取值。 并估计预测或控制达到的精确程度。
    3. 进行因素分析， 若有多个变量(因素)影响一个变量(因素)， 找出因素的主次，分析因素间的关系。

2. 图表相关分析：将数据点绘制成图表后更好的显示趋势和联系；对于有明显时间维度的数据，选择使用折线图
3. 协方差与协方差矩阵：协方差用来衡量两个变量的总体误差，如果两个变量的变化趋势一致，协方差就是正值，说明两个变量正相关。如果两个变量的变化趋势相反，协方差就是负值，说明两个变量负相关。如果两个变量相互独立，那么协方差就是0，说明两个变量不相关。

$$cov(X,Y)=\frac{\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y})}{n-1}=S_{xy}$$

4. 相关系数：相关系数是反应变量之间关系密切程度的统计指标，相关系数的取值区间在1到-1之间。 1表示两个变量完全线性相关， -1表示两个变量完全负相关， 0表示两个变量不相关。数据越趋近于0表示相关关系越弱。

$$r_{xy}=\frac{S_{xy}}{S_xS_y}$$

5. 一元回归和多元回归：回归分析是确定两组或两组以上变量间关系的统计方法， 按照变量的数量分为一元回归和多元回归。两个变量使用一元回归，两个以上变量使用多元回归。
    1. 一元回归方程： $y=b_0+b_1x$
    2. 最小二乘法计算b_1： $b_1=\frac{\sum(x_i-\overline{x})(y_i-\overline{y})}{\sum(x_i-\overline{x})^2}$

CH4 机器学习

4.1 机器学习算法

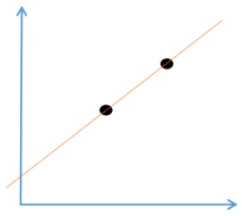
1. 定义
    1. 人工智能：机器展现的人类智能
    2. 机器学习： 计算机利用已有的数据(经验)，得出了某种模型，并利用此模型预测未来的一种方法。
    3. 深度学习： 实现机器学习的一种技术
2. 分类：
    1. 监督学习：表示机器学习的数据是带标记的，这些标记可以包括数据类别、数据属性及特征点位置等。这些标记作为预期效果，不断修正机器的预测结果
        1. 分类
        2. 回归
    2. 无监督学习：输入数据没有被标记， 也没有确定的结果。 样本数据类别未知， 需要根据样本间的相似性对样本集进行分类试图使类内差距最小化， 类间差距最大化。
        1. 聚类
        2. 降维
        3. 关联规则
        4. 推荐系统

    3. 半监督学习：半监督学习使用大量的未标记数据， 以及同时使用标记数据， 来进行模式识别工作。当使用半监督学习时， 将会要求尽量少的人员来从事工作， 同时又能够带来比较高的准确性。
    4. 强化学习：强化学习是带有激励机制的，如果机器行动正确，将施予一定的“正激励”；如果行动错误，同样会给出一个惩罚（也可称为“负激励”）。因此在这种情况下，机器将会考虑如何在一个环境中行动才能达到激励的最大化，具有一定的动态规划思想。
    5. 深度学习：神经网络算法的衍生， 是新兴的机器学习研究领域，旨在研究如何从数据中自动地提取多层特征表示，其核心思想是通过数据驱动的方式，采用一系列的非线性变换，从原始数据中提取由低层到高层、由具体到抽象、由一般到特定语义的特征。
3. 一般步骤：
    1. 数据搜集
    2. 数据清洗
    3. 特征工程
    4. 数据建模
4. 特征工程：
    1. 定义：把原始数据转变为模型的训练数据的过程
    2. 目的作用：
        1. 好的特征即使使用一般的模型， 也能得到很好的效果。
        2. 使模型的性能得到提升
    3. 构成：
        1. 特征构建：首先需要从现有数据中挑选或将现有数据进行变形，组合形成新特征
        2. 特征提取：当特征维度比较高，通过映射或变化的方式，用低维空间样本来表示样本
        3. 特征选择：从一组特征中挑选出一些最有效的特征，以达到降低维度和降低过拟合风险的目的

4.2 线性回归

1. 简介：种根据属性的线性组合来进行预测的线性模型， 其目的是找到一条直线或者一个平面或者更高维的超平面， 使得预测值与真实值之间的误差最小化。
2. 方法：线性回归方程是利用最小二乘函数对一个或多个自变量和因变量之间关系进行建模的一种回归分析。 这种函数是一个或多个称为回归系数的模型参数的线性组合。 只有一个自变量的情况称为简单回归， 大于一个自变量情况的叫做多元回归。
3. 算法流程：回归是一种不断迭代的算法，通过不断的训练数据进行迭代优化，而优化的目的就是降低损失，使损失函数的值尽可能小，而试图降低损失的过程，就是一个优化过程

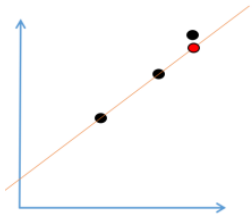
➢算法流程



x和y的关系

$$h(x)=w_0+w_1x_1+w_2x_2+\ldots+w_n x_n$$

设$x_0=1$，



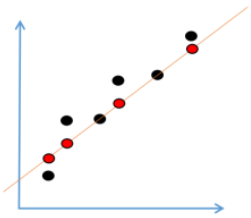
则：$\hat{y}=h(x)=w_0x_0+w_1x_1+w_2x_2+\ldots w_n x_n=w^T X$

残差平方：$l\left(x^{(i)}\right)=\frac{1}{2}(h(x^{(i)})-y^{(i)})^2$

要找到一组$w(w_0,w_1,w_2,\ldots,w_n)$，使得

$$J(w)=\frac{1}{2}\sum_{i=1}^{m}\left(h\left(x^{(i)}\right)-y^{(i)}\right)^2$$

(残差平方和)最小



回归是一种不断迭代的算法，通过不断的训练数据进行迭代优化，而优化的目的就是降低损失，使损失函数的值尽可能小，而试图降低损失的过程，就是一个优化过程

4. 最小二乘法（LSM）：

要找到一组$w(w_0,w_1,w_2,\ldots,w_n)$，使得$J(w)=\frac{1}{2}\sum_{i=1}^{m}\left(h\left(x^{(i)}\right)-y^{(i)}\right)^2$（残差平方和）最小，即最小化：$\frac{\partial J(w)}{\partial w}$

将向量表达形式转为矩阵表达形式，则有：$J(w)=\frac{1}{2}(Xw-Y)^2$

其中X为m行n+1列的矩阵（m为样本个数，n为特征个数），w为n+1行1列的矩阵（包含了w_0），Y为m行1列的矩阵，则：

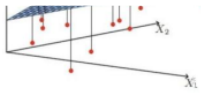
$$J(w)=\frac{1}{2}(Xw-Y)^2=\frac{1}{2}(Xw-Y)^T(Xw-Y)$$

向量平方的性质：

$$\sum_i z_i^2=z^Tz$$

$$X=\begin{bmatrix}1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \ldots & x_n^{(1)}\\ 1 & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \ldots & x_n^{(2)}\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ 1 & x_1^{(m)} & x_2^{(m)} & x_3^{(m)} & \ldots & x_n^{(m)}\end{bmatrix}\quad Y=\begin{bmatrix}y^{(1)}\\ y^{(2)}\\ \ldots\\ y^{(m)}\end{bmatrix}$$

为了最小化，对J(w)求偏导：



$$\frac{\partial J(w)}{\partial w}=\frac{1}{2}\frac{\partial}{\partial w}(Xw-Y)^T(Xw-Y)\ =\frac{1}{2}\frac{\partial}{\partial w}(w^TX^TXw-Y^TXw-w^TX^TY+Y^TY)$$

由于中间两项互为转置：

$$\frac{\partial J(w)}{\partial w}=\frac{1}{2}\frac{\partial}{\partial w}(w^TX^TXw-2w^TX^TY+Y^TY)\ =\frac{1}{2}(2X^TXw-2X^TY+0)$$

$$=X^TXw-X^TY$$

令：$\frac{\partial J(w)}{\partial w}=0$

则有：$w=(X^TX)^{-1}X^TY$

需要用到一下几个矩阵的求导法则：

$\frac{dX^TX}{dX}=2X$ $\frac{dAX}{dX}=A^T$

$\frac{\partial X^TAX}{\partial X}=(A+A^T)X$

若A为对称阵，$\frac{\partial X^TAX}{\partial X}=2AX$

X 是列满秩的，求矩阵的逆比较慢

$$w=(X^TX)^{-1}X^TY$$

5. 梯度下降法：
    1. 思想：在直线方程中， 导数代表斜率， 在曲线方程中， 导数代表切线的斜率。 导数代表着参数、 w单位变化时， 损失函数J相应的的变化；
    2. 学习率：如果最小化一个函数，就需要得到导数再取个负数，并乘以一个系数，系数通常叫做步长或者叫学习率(Learning rate, Lr)。α的取值影响获得求最优解的速度，取值不合适的话甚至得不到最优解，它是梯度下降的一个超参数。α太小，减慢收敛速度效率，α太大，甚至会导致不收敛。
    3. 缺点：对于梯度下降算法而言， 最不友好的就是并不是所有的函数都有唯一的极值点。很大概率就是局部最优解， 并不是真正的全局最优解。 这还只是针对二维平面来说， 如果对于高维空间更加相对复杂的环境， 就更复杂。解决方案： 多次运行，随机化初始点， 初始点也是梯度下降算法的一个超参数
    4. 收敛时间：需要保证所有特征值的大小比例都差不多，否则收敛的时间会长很多；损失函数虽然是碗状的， 但如果不同特征的尺寸差别巨大， 那它可能是一个非常细长的碗。
    5. 三种形式：
        1. 批量梯度下降(Batch Gradient Descent, BGD): 梯度下降的每一步中，都用到了所有的训练样本。
        2. 随机梯度下降(Stochastic Gradient Descent, SGD): 梯度下降的每一步中，用到一个样本， 在每一次计算之后便更新参数，而不需要首先将所有的训练集求和。
        3. 小批量梯度下降(Mini-Batch Gradient Descent, MBGD): 梯度下降的每一步中，用到了一定批量的训练样本。
6. LSM和梯度下降对比：
    1. 最小二乘法：不需要选择学习率α， 一次计算得出， 需要计算(XTX)-1， 如果特征数量n较大则运算代价大， 通常来说当n小于10000时还是可以接受的， 只适用于线性模型， 不适合逻辑回归模型等其他模型。

    2. 梯度下降：需要选择学习率α， 需要多次迭代， 当特征数量n大时也能较好适用， 适用于各种类型的模型。
7. 过拟合的处理：
    1. 获得更多数据
    2. 降维、PCA
8. 欠拟合处理：
    1. 添加新特质
    2. 复杂模型
    3. 减小正则化稀疏
9. 常用损失函数：
    1. MSE：$\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}-\hat{y}^{(i)})^2$
    2. MAE：$\frac{1}{m}\sum_{i=1}^{n}|(y^{(i)}-\hat{y}^{(i)}|$
    3. R2：

$$R^2(y,\hat{y})=1-\frac{\sum_{i=0}^{m}(y^{(i)}-\hat{y}^{(i)})^2}{\sum_{i=0}^{m}(y^{(i)}-\overline{y})^2}=\frac{\text{SSR}}{\text{SST}}$$
$$=1-\frac{\text{SSE}}{\text{SST}}$$
$$R^2(y,\hat{y})=1-\frac{\sum_{i=0}^{m}(y^{(i)}-\hat{y}^{(i)})^2/m}{\sum_{i=0}^{m}(y^{(i)}-\overline{y})^2/m}$$
$$=1-\frac{\text{MSE}}{\text{Var}}$$

$$SSR=\sum_{i=0}^{m}(\hat{y}^{(i)}-\overline{y})^2$$
$$SSE=\sum_{i=0}^{m}(y^{(i)}-\hat{y}^{(i)})^2$$
$$SST=\sum_{i=0}^{m}(y^{(i)}-\overline{y})^2$$

4.3 逻辑回归 *分类算法*

1. 回归和分类的区别：回归模型的输出时连续的， 分类模型的输出时离散的
2. Sigmoid：$\sigma(z)=g(z)=\frac{1}{1+e^{-z}}$
3. 多分类问题：先定义其中一类为类型1（正类），其余数据为负类；接下来去掉类型1的数据， 剩余部分再次进行二分类， 分成类型2和负类；如果有n类，那就需要分类n-1次

4.4 KNN

1. K近邻法（k-Nearest Neighbor, KNN）：给定一个训练数据集，对于新的输入实例， 在训练数据集中找到与该实例最邻近的K个实例，这K个实例的多数属于某个类， 就把该输入实例分类到这个类中；
    1. K近邻算法是将数据集合中每一个数据进行分类的方法。
    2. 可用于基本的分类与回归方法， 比较适用于样本容量比较大的类域的自动分类。
2. 算法流程：
    1. 计算测试对象到训练集中每个对象的距离
    2. 按照距离的远近排序
    3. 选取与当前测试对象距离最近的k个训练对象，作为该测试对象的邻居
    4. 统计这k个邻居的类别频次
    5. K个邻居里频次最高的类别为测试对象的类别

3. 距离：
    1. 欧氏距离
    2. 曼哈顿距离
    3. 切比雪夫距离
    4. 闵可夫斯基距离（P范数）
4. 缺陷：
    1. 需要存储全部的训练样本
    2. 计算量大：因为对每一个待分类的样本都要计算它到全体已知样本的距离， 以求得它的K个最近邻点
5. 改进：分组快速搜索近邻法：将样本集按近邻关系分解成组，给出每组质心的位置，以质心作为代表点，和未知样本计算距离，选出距离最近的一个或若干个组，再在组的范围内应用常规的KNN算法。由于并不是将未知样本与所有样本计算距离，故该改进算法可以有效减少计算量。

4.5 朴素贝叶斯

1. 贝叶斯分类：贝叶斯分类是一类分类算法的总成， 这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。
2. 概论：
    1. 先验概率：根据以往经验和分析得到的概率。 用$P(Y)$来代表在没有训练数据前假设Y拥有的初始概率
    2. 后验概率：根据已经发生的事件来分析得到的概率。 以$P(Y|X)$代表假设X成立的情下观察到Y数据的概率， 因为它反映了在看到训练数据X后Y成立的置信度。
    3. 联合概率：联合概率是指在多元的概率分布中多个随机变量分别满足各自条件的概率。 X与Y的联合概率表示为$P(X,Y)$、 $P(XY)$或$P(X\cap Y)$。 假设X和Y都服从正态分布，那么$P(X<5,Y<0)$就是一个联合概率， 表示$X<5,Y<0$两个条件同时成立的概率。 表示两个事件共同发生的概率。
3. 朴素贝叶斯：朴素贝叶斯法是典型的生成学习方法。生成方法由训练数据学习联合概率分布$P(X,Y)$，然后求得后验概率分布$P(Y|X)$。具体来说，利用训练数据学习$P(X|Y)$和$P(Y)$的估计，得到联合概率分布：$P(X,Y) = P(X|Y)P(Y)$
4. 贝叶斯公式：P(B|A) = P(A|B) P(B) / P(A) P(类别|特征)=P(特征|类别)P(类别)/P(特征)

4.6 决策树

4.6.1 决策树基本概念

1. 定义
    1. 决策树是从训练数据中学习到的一个树状结构的模型。决策树属于监督学习。
    2. 决策树是一种树状结构，通过做出一系列决策（选择）来对数据进行划分，这类似于针对一系列问题进行选择。
2. 决策过程：

    1. 决策树的决策过程就是从根节点开始，测试待分类项中对应的特征属性，并按照其值选择输出分支，直到叶子节点，将叶子节点的存放的类别作为决策结果。
    2. 决策树算法是一种归纳分类算法，通过对训练集的学习，挖掘出有用的规则，用于对新数据进行预测。
    3. 决策树归纳的基本算法是贪心算法，自顶向下来构建决策树。贪心算法：在每一步选择中都采取当前状态下最好/优的选择。
3. 基本类型：建立决策树的关键，在于当前状态下选择哪个属性作为分类依据。根据不同的目标函数，建立决策树主要有以下三种算法：ID3；C4.5；CART

4.6.2 ID3算法

1. 定义：
    1. ID3算法最早是由J.RossQuinlan于1975年提出的一种决策树构建算法，算法的核心是“信息熵”， 期望信息越小， 信息熵越大， 样本纯度越低。
    2. ID3算法是以信息论为基础， 以信息增益为衡量标准， 从而实现对数据的归纳分类。
    3. ID3算法计算每个属性的信息增益， 并选取当前具有最高增益的属性作为给定的测试属性。
2. 算法步骤：
    1. 初始化特征及数据集合；
    2. 计算数据集合的信息熵和所有特征的条件熵， 选择信息增益最大的特征作为当前决策节点；
    3. 更新数据集合和特征集合（删除上一步使用的特征， 并按照特征值划分出不同分支的数据集合）；
    4. 重复2， 3两步， 若子集值包含单一特征， 则为分支叶子节点。
3. 信息熵：
    1. 信息熵：

$$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}log_2\frac{|C_k|}{|D|}$$

$$H(D_i)=H(D|A=A_i)=-\sum_{k=1}^{K}\frac{|C_{kA_i}|}{|D_{A_i}|}log_2\frac{|C_{kA_i}|}{|D_{A_i}|}$$

    2. 条件熵：

$$H(D|A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)$$

    3. 信息增益：

$$g(D,A)=H(D)-H(D|A)$$

$$g(D,A=A_i)=H(D)-H(D_i)$$

4.6.3 C4.5算法

1. 定义：
    1. C4.5算法是Ross对ID3算法的改进。
    2. 用信息增益率来选择属性。 ID3选择属性用的是子树的信息增益， 而C4.5用的是信息增益率， 一般选择信息增益率最大的属性。
    3. 在决策树构造过程中进行剪枝。
    4. 对非离散数据、 不完整数据也能处理。
2. 信息增益率：

$$g_R(D,A)=\frac{g(D,A)}{H(D)}$$

3. 剪枝：
    1. 为了尽可能正确分类训练样本， 结点的划分过程会不断重复直到不能再分，这样就可能对训练样本学习的“太好” 了， 把训练样本的一些特点当作所有数据都具有的一般性质， 从而导致过拟合。
    2. 通过剪枝操作去掉一些分支来降低过拟合的风险。
    3. 剪枝的基本策略有“预剪枝” (prepruning)和“后剪枝” (post-pruning)。
4. 预剪枝：
    1. 预剪枝是要对属性选择前后模型的泛化性能及逆行评估，对比决策树某节点生成前与生成后的泛化性能。
    2. 策略：在节点划分前来确定是否继续增长， 及早停止增长主要方法有：
        1. 节点内数据样本低于某一阈值；
        2. 所有节点特征都已分裂；
        3. 节点划分前准确率比划分后准确率高。
    3. 步骤：
        1. 在未划分前，根据训练集，选择标记类别为好瓜（选择数目最多的类别，此训练集好瓜、坏瓜一样多）
        2. 计算训练集的信息增益，得知脐部的信息增益最大，按照脐部进行划分
        3. 在脐部划分的基础上，进一步计算凹陷、根蒂特征下，其他属性的信息增益，根据计算结果可知，在凹陷的情况下，色泽的信息增益最大，因此对于凹陷的西瓜，进一步确定按照色泽进行划分
    4. 特点：预剪枝使得很多分支没有展开，这不仅降低了过拟合的风险，还显著减少了决策树的训练时间和测试时间。但是，有些分支虽当前不能提升泛化性，甚至可能导致泛化性暂时降低，但在其基础上进行后续划分却有可能导致显著提高，因此预剪枝的这种贪心本质，给决策树带来了欠拟合的风险。
5. 后剪枝：
    1. 定义：
        1. 在已经生成的决策树上进行剪枝， 从而得到简化版的剪枝决策树。
        2. 后剪枝决策树通常比预剪枝决策树保留了更多的分支。一般情况下， 后剪枝的欠拟合风险更小， 泛化性能往往优于预剪枝决策树。

2. 策略：
    1. 在已经生成的决策树上进行剪枝，从而得到简化版的剪枝决策树。
    2. C4.5用递归的方式从低往上针对每一个非叶子节点， 评估用一个最佳叶子节点去代替这颗子树是否有益。 如果剪枝后与剪枝前相比其错误率是保持或者下降， 则这颗子树就可以被替换掉。
    3. C4.5通过训练数据集上的错误分类数量来估算位置样本上的错误率。
    4. 后剪枝决策树的欠拟合风险很小，泛化性能往往优于预剪枝决策树。
3. 步骤：
    1. 后剪枝将从决策树的底部往上进行剪枝， 先看最底部的纹理， 将其所在的分支剪掉， 即将其替换成叶子节点。
    2. 接着往上裁剪， 此时应该是色泽部分， 由于在训练集上， 替换后，包含的样本号为{6， 7， 15}， 好瓜（2个） 多于坏瓜（1个） ， 因此选择好瓜进行标记。
    3. 接下来， 看脐部凹陷分支。 由于在训练集上， 将色泽替换为叶节点后， 包含的样本号为{1， 2， 3， 14}， 好瓜（3个） 多于坏瓜（1）个， 因此选择好瓜进行标记。

4.6.4 CART算法

1. Classification And Regression Tree， 即分类回归树算法， 简称CART算法， 它是决策树的三种实现之一。
2. CART算法是一种二分递归分割技术， 把当前样本划分为两个子样本， 使得生成的每个非叶子结点都有两个分支， 因此CART算法生成的决策树是结构简洁的二叉树。
3. 由于CART算法构成的是一个二叉树， 它在每一步的决策时只能是“是”或者“否” ， 即使一个feature有多个取值， 也是把数据分为两部分。 在CART算法中主要分为两个步骤（1） 将样本递归划分进行建树过程（2） 用验证数据进行剪枝。
4. CART算法用基尼指数来选择属性（分类），或用均方差来选择属性（回归）。既可以用于创建分类树，也可以用于创建回归树，两者在构建的过程中稍有差异。如果目标变量是离散的，称为分类树；如果目标变量是连续的， 称为回归树。

4.6.5 总结

1. **划分标准的差异** ID3使用信息增益偏向特征值多的特征， C4.5使用信息增益率克服信息增益的缺点， 偏向于特征值少的特征， CART使用基尼指数克服C4.5的巨大计算量， 偏向于特征值较多的特征。
2. **使用场景的差异** ID3和C4.5都只能用于分类问题， CART可以用于分类和回归问题； ID3和C4.5是多叉树， 速度较慢， CART是二叉树， 计算速度很快；
3. **样本数据的差异** ID3只能处理离散数据且缺失值敏感， C4.5和CART可以处理连续性数据且有多种方式处理缺失值；从样本量考虑的话， 小样本建议C4.5、 大样本建议CART。 C4.5处理过程中需对数据集进行多次扫描排序，处理成本耗时较高， 而CART本身是一种大样本的统计方法， 小样本处理下泛化误差较大，

4.7 K-means

4.7.1 无监督学习

1. 场景：
    1. 搜索引擎：百度是常用的搜索引擎之一。 当我们搜索一些信息， 如位于某地的超市， 百度将为我们提供不同的超市选择。 这些提供给我们的结果就是聚类的相似结果。
    2. 2.社交网络：通过已知的朋友信息， 比如经常发email的联系人， 或是你的微博好友、 微信朋友圈好友， 我们可以运用聚类方法自动地对朋友进行分组， 做到让每个组里的人们彼此都熟识。
2. 聚类算法：
    1. 聚类分析又称群分析， 是研究（样品或指标） 分类问题的一种统计分析方法， 同时也是数据挖掘的一个重要算法。
    2. 聚类（Cluster） 分析是由若干模式（Pattern） 组成的， 通常模式是一个度量（Measurement） 的向量， 或者是多维空间的一点。
    3. 聚类分析以相似性为基础， 在一个聚类的模式之间比不在同一聚类中的模式之间具有更多的相似性。
    4. 图中的数据可以分成三个分开的点集（称为簇） ， 一个能够分出这些点集的算法，被称为聚类算法。

K-means聚类

1. 距离度量：闵可夫斯基距离； p取1或2时的闵氏距离是最为常用的； p=2即为欧氏距离,p=1时则为曼哈顿距离； 当p取无穷时的极限情况下，可以得到切比雪夫距离
2. K-means算法流程：
    1. 选择K个点作为初始质心(聚类中心)。
    2. 计算每个样本到各个质心的距离并将其指派到最近的质心，形成K个簇。
    3. 对于上一步聚类的结果，将各质心到对应簇中每个样本的距离和进行平均，得出该簇的新的聚类中心。
    4. 重复上述两步/直到迭代结束：质心不发生变化。
3. 优化：
    1. 上述两个步骤是迭代进行的， 直到质心停止移动， 即质心不再改变， 并且成为静态的。 这种情况下， k-means算法被称为收敛。
    2. 代价函数：

$$J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_k)=\frac{1}{m}\sum_{i=1}^{m}||X^i-\mu_{c^{(i)}}||^2$$

    优化目标是找出使代价函数最小的 $c^{(1)},\dots c^{(m)},\mu_1,\dots\mu_k$
4. 数据标准化：在进行聚类之前我们需要对数据进行标准化处理。因为在实际数据中，不同维度的数值范围可能相差较大，在计算均值时，数值较大的维度会对结果产生主导作用，为了避免这种情况，需要进行标准化。
5. K值选择：通常通过“肘部法则” 进行计算。 随着K数量的增加，代价函数的值会迅速下降， 类似于人的肘部曲线（右图在K=3的时候达到一个肘点） 。 在此之后， 代价函数的

值会就下降得非常慢， 所以我们选择K=3。 这个方法叫"肘部法则"

6. 总结
    1. 优点：
        1. 鲁棒性高；
        2. 速度快、 易于理解、 效率高；
        3. 计算成本低、 灵活性高；
        4. 如果数据集是不通的， 则结果更好；
        5. 可以产生更紧密的簇；
        6. 重新计算质心时， 簇会发生变化
    2. 缺点：
        1. 需预先指定簇的数量；
        2. 如果由两个高度重叠的数据， 那么它就不能被区分， 也不能判断有两个簇；
        3. 有时随机选择质心并不能带来理想的结果；
        4. 无法处理异常值和噪声数据；
        5. 不适用于非线性数据集；
        6. 对特征尺度敏感；

4.8 Apriori算法

4.8.1 关联规则

1. 关联规则反应一个事物与其他事物之间的相互依存性和关联性。 如果两个或者多个事物之间存在一定的关联关系， 那么， 其中一个事物就能够通过其他事物预测到。
2. 关联规则可以看作是一种IF-THEN关系。 假设商品A被客户购买， 那么在相同的交易ID下， 商品B也被客户挑选的机会也会增大。

4.8.2 Apriori算法

1. 定义：
    1. Apriori算法利用频繁项集生成关联规则。
    2. 频繁项集是支持度满足最小支持度阈值
    3. Apriori算法就是基于一个先验：如果某个项集是频繁的， 那么它的所有非空子集也是频繁的，频繁项集的非空子集也必须是频繁项集。
2. 流程：
    1. 输入：数据集合D， 支持度阈值α
    2. 输出：最大的频繁k项集
        1. 扫描整个数据集， 得到所有出现过的数据， 作为候选频繁1项集。 k=1， 频繁0项集为空集
        2. 挖掘频繁k项集扫描数据计算候选频繁k项集的支持度；去除候选频繁k项集中支持度低于支持度阈值的数据集， 得到频繁k项集。 如果得到的频繁k项集为空，则直接返回频繁k-1项集的集合作为算法结果。 如果得到的频繁k项集只有一

项， 则直接返回频繁k项集的集合作为算法结果， 算法结束；基于频繁k项集，连接生成候选频繁k+1项集。

3. 令k=k+1， 转入步骤2.

4.9人工神经网络

4.9.1 感知机

1. 神经元
2. 感知机：
    1. 感知机接收多个输入信号， 输出一个信号。 如同电流流过导线， 向前方输送电子一样， 感知机的信号也会形成流， 向前方输送信息。 但是， 和实际的电流不同的是，感知机的信号只有“流/不流” 两种取值。
    2. 输入信号被送往神经元时， 会被分别乘以固定的权重。神经元计算传送过来的信号的综合， 只有当这个总和超过了某个界限值时， 才会输出1，被称为“神经元被激活”；将该界限值称为阈值， 用符合θ表示。
    3. 局限性：
        1. 感知机可以表示与门、 与非门、 或门的逻辑电路；重要的一点：与门、 与非门、 或门的感知机构造是一样的；实际上， 3个门电路只有参数的值(权重和阈值)不同， 即， 先沟通构造的感知机，只需要适当地调整参数的值， 就可以向“变色龙演员” 表演不同的角色一样， 变身为与门、 与非门、 或门。
        2. 感知机的局限性在于它只能表示由一条直线分割的空间。 下图这样的弯曲的曲线无法用感知机表示；无法表示异或门
3. 多层感知机：异或门是一种多层结构的神经网络。 将最左边的一列称为第0层， 中间的一列称为第1层， 最右边的一列称为第2层；叠加了多层的感知机也成为多层感知机。

4.9.2 神经网络

1. 定义：
    1. 在前一小节中， 感知机的参数由人为设定， 我们看着真值表这种“训练数据” ，人工考虑到参数的值；机器学习的将这个决定参数值的工作交由计算机自动进行；学习是确定合适参数的过程， 而人要做的是思考感知机的构造(模型)， 并把训练数据交给计算机；神经网络的一个重要形式是它可以自动地从数据中学习到合适的权重参数
    2. 最左边一列称为输入层， 最右边一列称为输出层；中间一列称为中间层， 有时也成为隐藏层；另外， 把输入层到输出层一次称为第0层、 第1层、 第2层...
2. 感知机：

$$y = h(b + w_1x_1 + ... + w_mx_m)$$

3. 激活函数：
    1. 前述的h(x)函数会将输入信号的总和转换为输出信号， 这种函数一般称为激活函数；式子先计算加权输入信号和偏置的总和， 记为a， 然后用h()函数将a转换为输出y。

2. Sigmoid函数：神经网络中经常使用的一个激活函数就是sigmoid函数；阶跃函数和sigmoid函数具有相似的特性：输入小时， 输出接近0；随着输入增大， 输出向1靠近；输出信号值都在0-1之间。
    1. 优点：函数处处可导，便于求导；可将函数值的范围压缩至[0, 1]，可用于压缩数据，且幅度不变；便于前向传输。
    2. 缺点：在趋向无穷的地方，函数值变化很小，容易出现梯度消失，不利于深层神经的反馈传输；幂函数的梯度计算复杂；收敛速度比较慢
3. 非线性函数：激活函数不能使用线性函数， 如果采用线性函数， 加深神经网络的层数就没有意义。
4. 常见的激活函数选择：ReLU，Sigmoid，Tanh，Leaky ReLU；一般， 回归问题可以使用恒等函数， 二元分类问题可以使用sigmoid函数， 多元分类问题可以使用softmax函数

4.9.3 BP神经网络

1. 算法流程：
    1. 将输入样本提供给输入层神经元
    2. 逐层将信号前传至隐层、输出层，产生输出层的结果
    3. 计算输出层误差
    4. 将误差反向传播至隐藏层神经元
    5. 根据隐层神经元对连接权重和阈值进行调整
    6. 上述过程循环进行，直至达到某些停止条件为止
2. 优点：
    1. 能够自适应、自主学习。 BP可以根据预设参数更新规则，通过不断调整神经网络中的参数，以达到最符合期望的输出
    2. 拥有很强的非线性映射能力
    3. 误差的反向传播采用的是链式法则，推导过程严谨且科学；算法的泛化能力很强
3. 缺点：
    1. 存在多个极小点：谷底并不是一个，而是多个，因此这些谷底就是局部最优点，不是全局最优的
    2. 梯度消失：梯度消失原因是链式求导，导致梯度逐层递减，通过链式求导把各层连接起来的，但是因为激活函数是sigmod函数，取值在1和-1之间，因此每次求导都会比原来小，当层次较多时，就会导致求导结果也就是梯度接近于0。

CH5 大数据计算框架基础

5.1 Hadoop概述

5.1.1 大数据

1. 三个理念：
    1. 用全量替代样本

    2. 兼容不精确
    3. 更加注重相关规律
2. 挑战：
    1. 业务部门无清晰的大数据需求
    2. 企业内部数据"孤岛"严重
    3. 数据可用性低，质量差
    4. 数据相关管理技术和架构
    5. 数据安全
    6. 大数据人才缺乏
3. 机遇：
    1. 数据分析成为大数据技术的核心
    2. 广泛采用实时性的数据处理方式
    3. 基于云的数据分析平台将更加完善

5.1.2 Hadoop发展

1. 发展：ASF-Apache软件基金会；Nutch项目
2. 问题：日益增加的数据量；数据传输速度
3. 解决：将一个数据集存储到多个硬盘里，然后并行读取
4. 新的问题：硬件故障；读取数据的正确性问题

5.1.3 Hadoop生态

Hadoop生态系统中常用的组件列举如下，不同的组件分别提供特定的服务
Hive；Zookeeper；Flume；HBase；Spark；Kafka

5.1.4 Hadoop技术

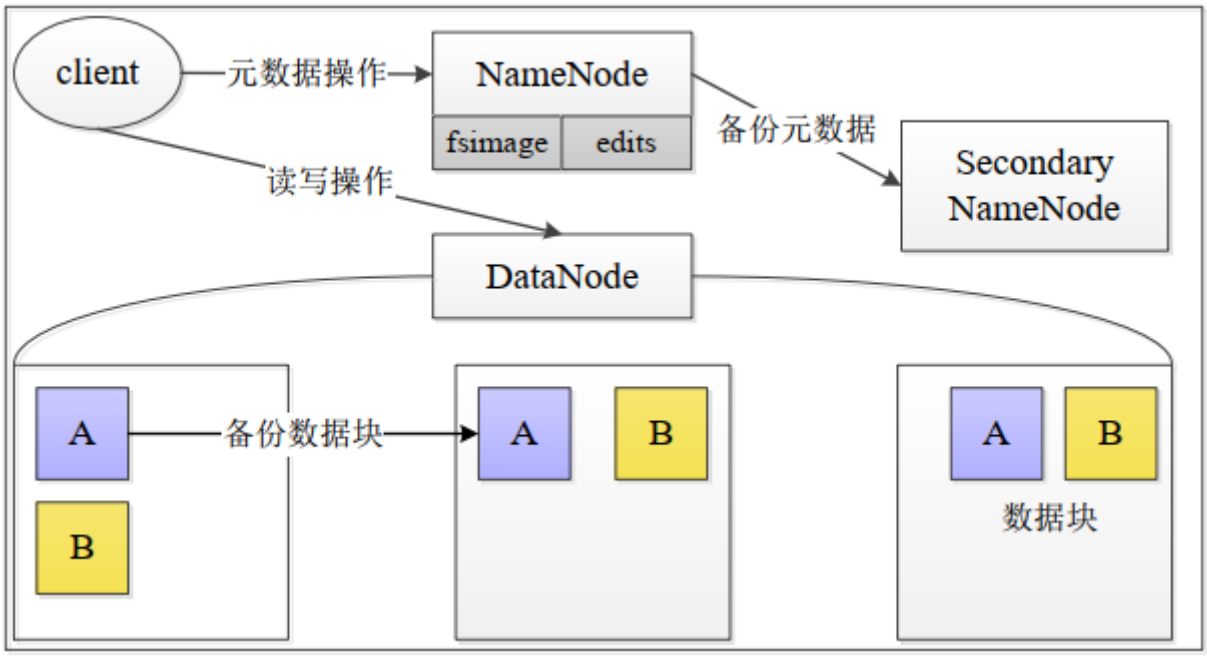
1. 定义：Apache Hadoop软件库是一个框架，它允许使用简单的编程模型跨计算机集群分布式处理大型数据集。它被设计成从单个服务器扩展到数千台机器，每台机器都提供本地计算和存储。该库本身的设计目的不是依靠硬件来提供高可用性，而是在应用层检测和处理故障，从而在计算机集群上提供高可用性服务，每个计算机集群都可能发生故障。
2. **计算机集群** 简单地说，集群就是指一组（若干个）相互独立的计算机，利用高速通信网络组成的一个较大的计算机服务系统，每个集群节点（即集群中的每台计算机）都是运行各自服务的独立服务器。
3. **分布式** 分布式系统是若干独立计算机的集合，这计算机对用户来说就像单个相关系统。分布式系统一定是由多个节点组成的系统。其中，节点指的是计算机服务器，而且这些节点一般不是孤立的，而是互通的。这些连通的节点上部署了我们的节点，并且相互的操作会有协同。不同的业务模块部署在不同的服务器上或者同一个业务模块分拆多个子业务部署在不同的服务器上，解决高并发的问题，提供可扩展性以及高可用性。
4. Hadoop特点：Hadoop是一个能够让用户轻松搭建和使用的分布式计算平台，用户可以轻松地在Hadoop上开发和运行处理海量数据的应用程序。

1. 高可靠性：数据存储有多个备份，集群部署在不同机器上，可以防止一个节点宕机造成集群损坏。当数据处理请求失败时， Hadoop将自动重新部署计算任务。
2. 高扩展性： Hadoop是在可用的计算机集群间分配数据并完成计算任务的。为集群添加新的节点并不复杂，因此集群可以很容易地进行节点的扩展从而扩大集群。
3. 高效性： Hadoop可以在节点之间动态地移动数据，在数据所在节点进行并行处理，并保证各个节点的动态平衡，因此处理速度非常快。
4. 高容错性： Hadoop的分布式文件系统HDFS在存储文件时将在多台机器或多个节点上存储文件的备份副本，当读取该文件出错或某一台机器宕机时，系统会调用其他节点上的备份文件，保证程序顺利运行。
5. 低成本： Hadoop是开源的，即不需要支付任何费用即可下载并安装使用，节省了软件购买的成本。
6. 可构建在廉价机器上： Hadoop不要求机器的配置达到极高的水准，大部分普通商用服务器即可满足要求，通过提供多个副本和容错机制提高集群的可靠性。
7. Hadoop基本框架用Java语言编写： Hadoop是一个由Java语言开发的框架，因此运行在Linux系统上是非常理想的， Hadoop上的应用程序也可以使用其他语言编写，如C++和Python。

5.1.4.1 Hadoop技术——HDFS

1. Hadoop Distributed File System：分布式文件系统：HDFS是一种旨在普通硬件上运行的分布式文件系统，与现有的分布式文件系统有许多相似之处，但也存在明显的区别。 HDFS具有高度的容错能力，旨在部署在低成本硬件上。 HDFS支持对应用程序数据进行高吞吐量访问，并且适用于具有海量数据集的读写。 HDFS是Hadoop的核心组件之一，用于存储数据。
2. HDFS架构：HDFS并不是一个单机文件系统，而是分布在多个集群节点上的文件系统。节点之间通过网络通信进行协作，提供多个节点的文件信息，使每个用户均可以看到文件系统的文件，使多台机器上的多用户可以分享文件和存储空间。HDFS是一个主/从（Master/Slave）体系架构的分布式文件系统。 HDFS支持传统的层次型文件组织结构，使得用户或应用程序可以创建目录，再将文件保存至目录中。文件系统命名空间的层次结构和大多数现有的文件系统类似，可以通过文件路径对文件执行创建、读取、 更新和删除

操作。

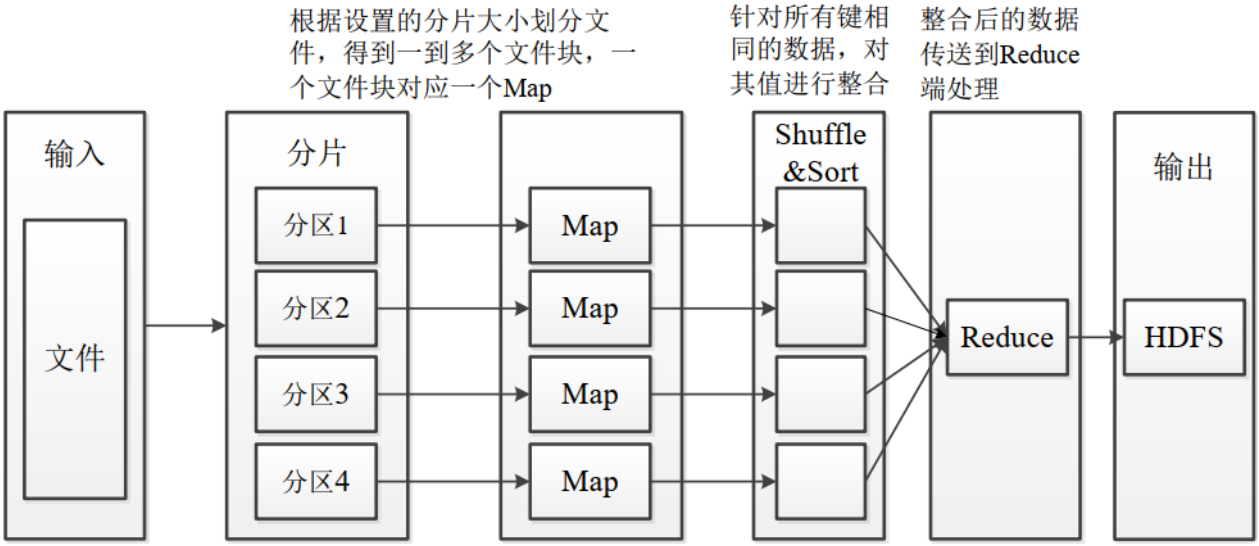


1. NameNode：用于存储元数据以及处理客户端发出的请求。元数据不是具体的文件内容，包含3类重要信息。第1类是文件和目录自身的属性信息，如文件名、目录名、父目录信息、文件大小、创建时间、修改时间等；第2类是记录文件内容存储的相关信息，如文件分块情况、副本个数、每个副本所在的DataNode信息等；第3类是用于记录HDFS中所有DataNode的信息，用于DataNode管理。
2. Secondary NameNode：用于备份NameNode的数据，周期性将edits文件合并到fsimage文件并在本地备份，将新的fsimage文件存储至NameNode，覆盖原有的fsimage文件，删除edits文件，并创建一个新的edits文件继续存储文件当前的修改状态。
3. DataNode：DataNode是真正存储数据的地方。在DataNode中，文件以数据块的形式进行存储。 Hadoop 3.x默认128 MB为一个数据块，如果存储一个大小为129 MB的文件，那么文件将被分为两个数据块进行存储。

3. HDFS的特点：
    1. 高容错性。 HDFS上传的数据自动保存多个副本，通过增加副本的数量增加HDFS的容错性。如果某一个副本丢失，那么HDFS将复制其他节点上的副本。
    2. 适合大规模数据的处理。 HDFS能够处理GB、 TB甚至PB级别的数据，数量级规模可达百万，数量非常大。
    3. 流式数据访问。 HDFS以流式数据访问模式存储超大文件，有着“一次写入，多次读取”的特点，且文件一旦写入，不能修改，只能增加，以保证数据的一致性。
    4. 局限性：不适合低延迟数据访问，无法高效存储大量小文件、不支持多用户写入及任意修改文件。
4. 适用范围：
    1. 适合：存储超大文件；流式数据访问；运行在廉价设备上
    2. 不适合：低时间延迟的数据访问；大量的小文件；多用户写入，任意修改文件

5.1.4.2 Hadoop技术——MapReduce

1. 定义：是一个软件框架，基于该框架能够容易地编写应用程序，这些应用程序能够运行在由上千个商用机器组成的大集群上，并以一种可靠的，具有容错能力的方式并行地处理上TB级别的海量数据集。软件框架——并行处理——可靠容错——大规模集群——海量数据集。
2. 核心思想：
    1. Map：负责“分”，即把复杂的任务分解为若干个“简单的任务”来处理；数据或计算的规模相对原任务要大大缩小；就近计算原则，即任务会分配到存放着所需数据的节点上进行计算；这些小任务可以并行计算，彼此间几乎没有依赖关系。
    2. Reduce：负责对map阶段的结果进行汇总
    3. MapReduce是Hadoop的核心计算框架，是用于大规模数据集（大于1TB）并行运算的编程模型，主要包括Map（映射）和Reduce（规约）两个阶段。当启动一个MapReduce任务时， Map端将会读取HDFS上的数据，将数据映射成所需要的键值对类型并传至Reduce端。Reduce端接收Map端键值对类型的中间数据，并根据不同键进行分组，对每一组键相同的数据进行处理，得到新的键值对并输出至HDFS。
3. 作业流程：



    1. 数据的输入与分片
    2. Map阶段数据处理
    3. Shuffle&Sort阶段数据整合
    4. Reduce阶段数据处理
    5. 数据输出

5.1.4.3 Hadoop技术——HBase

1. 简介：HBase是一个高可靠、高性能、面向列、可伸缩的分布式非结构化数据库，同时是一个列存储非关系型数据库，主要用于存储非结构化和半结构化的松散数据。 HBase是Hadoop Database的简称，是Apache软件基金会Hadoop项目的一部分，它是参考谷歌BigTable的模型利用Java实现的。HBase的目标是处理数据量非常庞大的表，并且可以通过水平扩展的方式，利用廉价计算机集群处理由超过10亿行数据和数百万列元素组成的数据表。
    1. 发展：略

2. 意义：关于数据存储和访问的大多数解决方案，特别是一些关系类型的，在构建时并没有考虑超大规模和分布式的特点；通过线性方式从下到上增加节点来进行扩展。
3. 概念：HBase是一种构建在HDFS之上的分布式、 面向列的存储系统。在需要实时读写、随机访问超大规模数据集时，可以使用HBase。

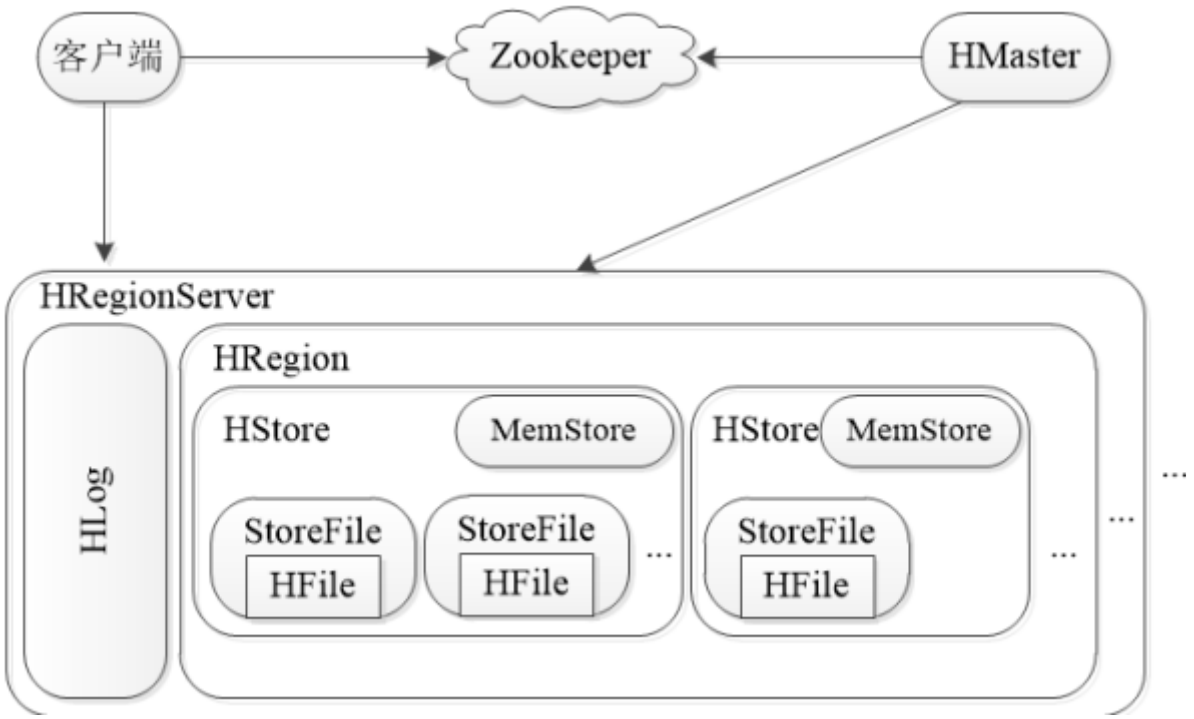
2. HBase特点：
   1. 基本特点：容量巨大；可扩展；稀疏性；高性能；多版本；支持过期；Hadoop原生支持。
   2. 与传统数据库RDBMS对比：

对比项	HBase	RDBMS
硬件	集群商用硬件	较贵的多处理器硬件
容错	单个或少个节点宕机对HBase没有影响	需要额外较复杂的配置
数据大小	TB到PB级数据，千万到十亿行	GB到TB级数据，十万到百万行
数据层	一个分布式、多维度的、排序的Map	行或列导向
数据类型	只有byte	多种数据类型支持
事务	单个行的ACID	支持表间和行间的ACID
查询语言	支持自身提供的API	SQL
索引	Rowkey索引	支持
吞吐量	每秒百万查询	每秒千次查询

   3. 与Hive对比：HBase和Hive在大数据架构中处在不同位置， HBase主要解决实时数据查询问题， Hive主要解决数据处理和计算问题，二者一般是配合使用。

对比项	HBase	Hive
延迟性	在线，低延迟	批处理，较高延迟
结构化	非结构化数据	结构化数据
适用人员	程序员	分析人员
适用场景	海量明细数据的随机实时查询，如日志明细、交易清单、轨迹行为等	离线的批量数据计算

3. 核心功能模块：HBase数据库主要由客户端Client、协调服务模块ZooKeeper、主节点服务HMaster、从节点服务HRegionServer和数据表分片Region 5个核心功能模块组成。



4. HBase的数据模型：传统行数据库以行的形式储存数据，每行数据包含多列，每列只有单个值。在HBase中，数据实际储存在一个“映射” (Map)中，并且“映射”的键（Key）是会被排序的。基于排序，用户可以自定义一个“行键”（Row Key），使“相关的”数据存储在相近的地方。

5.2 Spark

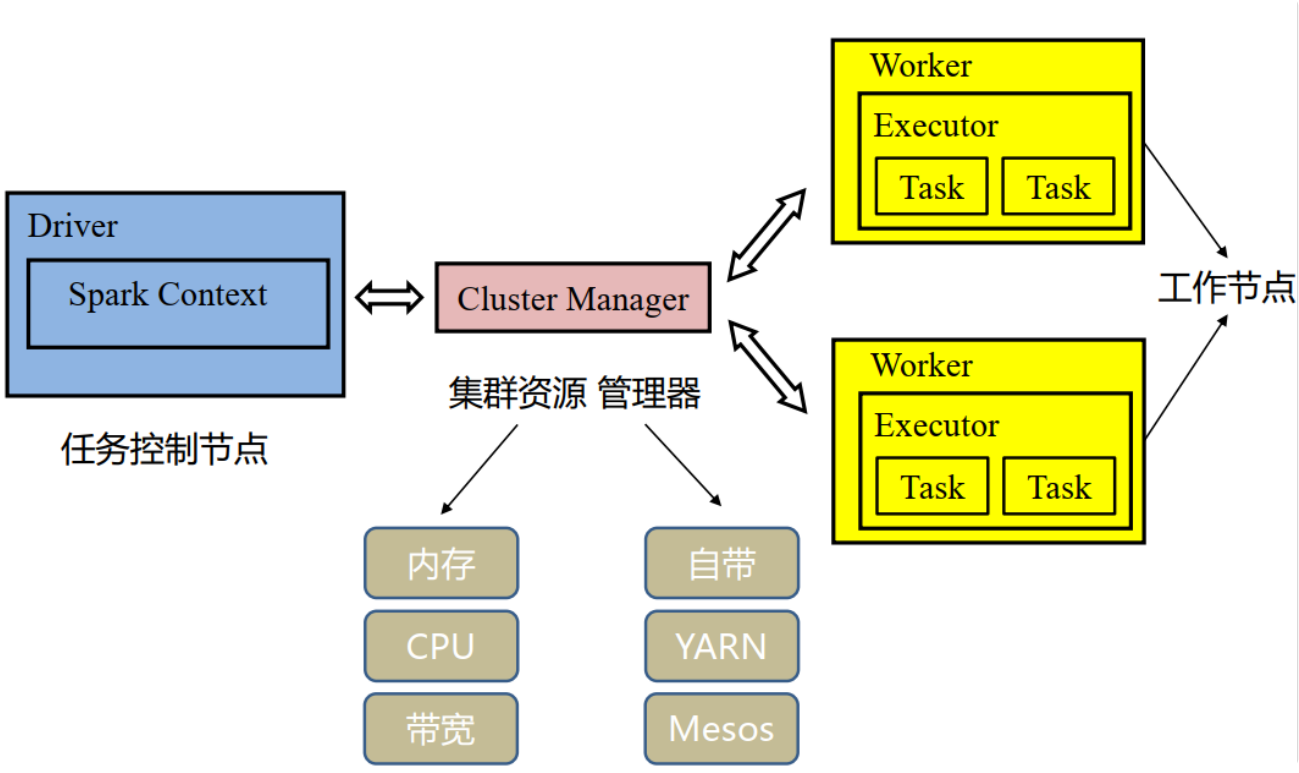
5.2.1 Spark简介

1. 发展：Spark诞生于加州大学伯克利分校AMP实验室，最初属于伯克利大学的研究性项目。该实验室的研究人员在基于Hadoop MapReduce框架进行工作时，发现MapReduce对于迭代和交互式计算任务的计算效率并不高。因此，研究人员开发Spark主要是为了提高交互式查询和迭代算法的计算效率。
2. 特点：快速；易用；通用；随处运行；代码简洁。
3. 生态：Spark生态圈也称为BDAS（伯克利数据分析栈），由AMP实验室打造，是致力于在算法（Algorithms）、 机器（Machines）、 人（People）之间通过大规模集成展现大数据应用的一个平台。
    1. 以Spark Core为核心， 可以从HDFS、 AmazonS3和HBase等数据源中读取数据，并支持不同的程序运行模式，同时可以以Mesos、 YARN、亚马逊EC2或Spark自带的Standalone作为资源管理器调度作业（Job）完成Spark应用程序的计算。
    2. Spark应用程序计算的整个过程可以来自不同的组件，如Spark SQL的即时查询、Spark MLlib的机器学习、 Spark Streaming的实时处理应用、GraphX的图处理和SparkR的数学计算等。
    3. 重要组件的简要介绍如下：
        1. Spark Core： Spark的核心，提供底层框架及核心支持。
        2. Spark SQL： 可以执行SQL查询，支持基本的SQL语法和Hive SQL语法，读取的数据源包括Hive表、 HDFS、关系数据库（如MySQL）等。

3. MLlib： Spark的机器学习算法库，实现了一些常见的机器学习算法和实用程序，包括分类、回归、聚类、协同过滤、特征降维和底层优化等
4. Spark Streaming：可以进行实时数据流式计算
5. GraphX：内置了许多图的相关算法，可以直接使用，如在移动社交关系分析中可直接使用图计算相关算法进行处理和分析。
6. SparkR： AMP实验室发布的一个R开发包，使得R语言编写的程序可以作为Spark的Job在集群运行，极大地扩展了R的数据处理能力。

4. Spark与Hadoop联系
   1. Spark对标Hadoop中的计算模块MapReduce，但计算速度要远高于MR。
   2. Spark没有提供文件系统，所以它必需与分布式文件系统合作才可以运行，它本身只是一个并行计算框架。
   3. Spark可以使用Hadoop的HDFS作为文件存储，也可以与其他分布式文件系统进行数据存储，但一般都使用HDFS。
   4. Spark有自带的独立集群管理器，同时也支持Hadoop的YARN资源调度管理框架。
5. Spark与Hadoop区别：
   1. 计算速度快（处理数据的设计模式不同）（磁盘I/O开销不同）：MR基于磁盘进行计算；Spark基于内存进行计算。
   2. Spark操作简单，代码更简洁：Spark用Scala语言编写，相比java语言编写的Hadoop程序更加简洁；纯面向对象编程语言， 支持封装、 继承， 所有操作均通过向对象发送消息执行；具备较强的并发性、函数式编程语言，可以更好地支持分布式系统；具有很强的兼容性和移植性： Scala运行于JVM上， 能够与Java进行互操作， 具有与Java一样的平台移植性

5.2.2 Spark架构



1. RDD：Resillient Distributed Dataset；弹性分布式数据集；是Spark最核心的概念
2. DAG：有向无环图：反应RDD之间的依赖关系
3. Executor：执行器：作为工作节点并行执行任务

5.2.3 Spark应用场景

目前大数据的应用非常广泛，其应用场景的普遍特点是计算量大、效率高。而Spark恰恰满足了这些要求，所以Spark项目一经推出便受到开源社区的广泛关注和好评。目前Spark已经发展成为大数据处理领域非常炙手可热的开源项目。国内外大多数大型企业公司对Spark的应用也十分广泛。

5.2.4 Spark集群安装配置：

1. Spark环境可分为单机版环境、 单机伪分布式环境和完全分布式环境。
2. 完全分布式环境是主从模式，即其中一台机器作为主节点master，其他的几台机器作为子节点slave。

5.2.5 RDD

5.2.5.1 RDD简介：

1. 弹性分布式数据集（RDD）表示一个关于分区数据元素的集合，具有如下特点：
    1. 不可变性： RDD是一种不可变的数据结构，一经创建则不可在原地修改，一个修改RDD的操作都会返回一个新的RDD
    2. 容错性：当一个节点出现故障时，该节点上存储的数据将无法访问， Spark会在其他节点上重建丢失的RDD分区数据
    3. 分片： RDD表示一组数据的分区，这些分区分布在多个集群节点上，当Spark在单节点上运行时，所有的分区数据都会在当前节点上
    4. 接口： RDD是一个处理数据的接口，为多种数据源提供了一个处理数据的统一接口，同时也可用于处理存储于多个节点内存中的数据
    5. 强类型： RDD可表示同一类型数据的分布式集合， 包括Integer、 Long、String或开发者自己定义的类型
    6. 驻留在内存中： Spark允许RDD在内存中缓存或长期驻留， 对一个缓存在内存中的RDD进行操作比操作没缓存的RDD要快很多
2. 操作：RDD操作归为两类：转换（Transformation） 和行动（Action）
    1. 转换操作是由一个RDD转换到另一个新的RDD， 如： map操作在RDD中是一个转换操作， map转换操作会让RDD中的每一个数据都通过一个指定函数得到一个新的RDD
    2. 动作操作用于向Driver程序返回值或将值写入到文件当中， 如： reduce动作会使用同一个指定函数让RDD中的所有数据做一次聚合， 把运算的结果返回。
3. RDD运行过程：
    1. RDD读入外部数据源进行创建， 例如使用textFile函数加载本地数据
    2. RDD经过一系列的转换操作，每一次转换都会产生不同的RDD以供给下一转换使用

3. 最后一个RDD经动作操作进行转换，并输出到外部数据源

4. 依赖关系：RDD之间的依赖关系涉及到窄依赖和宽依赖， Shuffle操作是区分上述依赖关系的依据：包含Shuffle操作为宽依赖，不含Shuffle则为窄依赖。
    1. Spark通过分析各个RDD的依赖关系生成了DAG， 再通过分析各个RDD中的分区之间依赖系决定如何划分Stage， 划分依据是窄依赖和宽依赖， 具体方式如下：• 在DAG中进行反向遍历， 遇到宽依赖就断开• 遇到窄依赖机就把当前的RDD加入到当前Stage• 将窄依赖尽量划分到一个Stage中， 可实现流水线计算。
    2. 窄依赖对于作业优化非常有利， 只有窄依赖可以实现流水线优化， 宽依赖包含Shuffle过程， 无法实现流水线方式处理。

5. 总结RDD在Spark架构中的运行过程：
    1. 提交所写的RDD代码给Spark框架， Spark框架根据提供代码生成一个GAG图。
    2. DAGScheduler将提交的DAG图划分为多个Stage， 每个Stage包括若干个Task， 每个Task会被分配给TaskScheduler。
    3. TaskScheduler会把Task分配给WorkNode上的Executor进程， 然后由Executor进程派发的线程执行Task， 即可完成RDD整个执行过程。

5.2.5.2 RDD的创建

1. 从文件系统中加载：`sc.textFile("file_path")`
2. 通过并行集合（数组）创建RDD：`sc.paralleize(array)`

5.2.5.3 转换操作

操作	含义
filter(func)	筛选出满足func的元素，并返回一个新的数据集
map(func)	将每个元素传递到函数func中，并将结果返回为一个新的数据集
flatMap(func)	与map()相似，但是每个输入元素可以映射到0或多个输出结果
groupByKey()	应用于（K,V）键值对的数据集时，返回一个新的（K，Iterable）形式的数据集
reduceByKey(func)	应用于（K,V）键值对的数据集时，返回一个新的（K,V）形式数据集，其中每个值是将每个key传递到函数func中进行聚合后的结果

```
reduceByKey(func) = func(gruopByKey())

sortByKey()  ===>   只能通过键来排序
sortBy()     ===>   可以通过表达式按照任意参数排序
```

```
mapValue(func)   ===>   对键值对RDD的每一个value应用一个func，key不变
rdd1.join(rdd2)  ===>   对于给定的两个数据数据集（K， V1）和（K， V2），只有在两个数据集中都存在的key才会被输出，最终得到一个（K,(V1,V2)）类型的数据集
```

5.2.5.4 行动操作：Action API

*count() 返回数据集中的元素个数

*collect() 以数组的形式返回数据集中的所有元素

*first() 返回数据集中的第一个元素

*take(n) 以数组的形式返回数据集中的前n个元素

*reduce(func) 通过函数func（输入两个参数并返回一个值）聚合数据集中的元素

*foreach(func) 将数据集中的每个元素传递到函数func中运行

5.2.5.5 惰性机制

RDD的Transformation操作都是惰性求值的,也就是说Transformation操作不会开始真正的计算，只有在执行Action操作的时候Spark才会真正开始计算。转化操作不会立刻执行，而是在内部记录下所要执行的操作的相关信息，必要时再执行。

5.2.6 Spark MLlib

1. MLlib是构建在Spark上的分布式机器学习数据库，充分利用了Spark的内存计算和适合迭代型计算的优势，使性能大幅提升，同时Spark算子丰富的表现力，让大规模机器学习的算法开发不再复杂
2. MLlib 由 4 部分组成：数据类型，数学统计计算库，算法评测和机器学习算法
3. MLlib中提供了当前主要的机器学习算法， 支持分类回归、 聚类、 推荐、 降维优化、 特征抽取筛选等

5.2.7 Spark Streaming

1. Spark Streaming是Spark API的一个扩展， 支持实时数据流的可扩展、 高吞吐量、 容错流处理。
2. DStream是Spark Streaming中一个非常重要的概念， Spark Streaming读取数据时会得到一个DStream编程模型， 而DStream提供了一系列操作的方法。

3. Spark Streaming是一个构建在Spark之上的子框架， 是Spark生态圈中用于处理流式数据的分布式流式处理框架。流式数据可以从许多来源（如Kafka、 Kinesis或TCP套接字） 获取， 获取的数据可以使用map()、 reduce()、 join()和window()等高级函数表达的复杂算法进行处理。
4. Spark Streaming能够和SparkSQL、 MLlib、 GraphX进行无缝集成， 使得用户可以将Spark的数据查询、 机器学习与图形处理算法应用于数据流。

5.3 Flume概述

5.3.1 Flume技术介绍

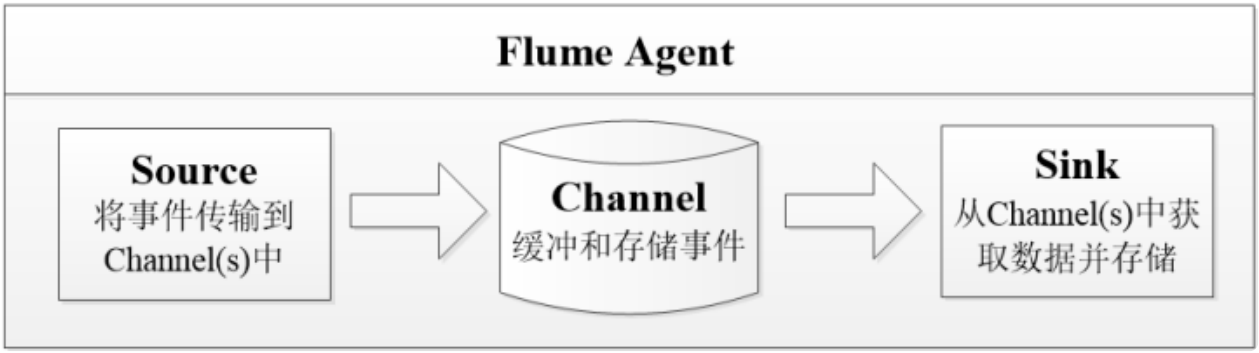
1. 简介：Apache Flume是一个从可以收集例如日志、 事件等数据资源， 并将这些数量庞大的数据从各项数据资源中集中起来存储的工具/服务。Flume是一个高可用、 高可靠、 分布式的采集工具， 其设计的原理是基于数据流将数据从各种网站服务器上汇集起来存储到HDFS， HBase等集中存储器中。 通俗一点来说就是Flume是一个很可靠、 方便、 强大的日志采集工具，是目前大数据领域数据采集最常用的一个框架。
2. 发展历程：lume最初是由Cloudera开发的日志收集系统， 受到了业界的认可与广泛应用， 后来逐步演化成支持任何流式数据收集的通用系统。 Flume目前存在两个版本， 初始的发行版本 Flume OG（original generation） 和重构后的版本Flume NG（Next/New Generation 。
3. 基本思想与特性：lume采用了插拔式软件架构， 所有组件均是可插拔的， 使得用户可以根据自己的需求定制每个组件。 Flume本质上是一个中间件， 主要具有以下几个特点。
    1. 良好的扩展性。 Flume的架构是完全分布式的， 没有任何中心化组件， 非常容易扩展。
    2. 高度定制化。 Flume采用的是插拔式架构， 各组件可以进行插拔式配置， 用户可以很容易的根据需求自由定义。
    3. 良好的可靠性。 Flume内置了事务支持， 能保证发送的每条数据都能够被下一跳收到而不丢失。
    4. 可恢复性。 Flume的可恢复性依赖于其核心组件Channel。 Channel的缓存类设置为FileChannel时， 事件可持久化到本地文件系统中。

5.3.2 Flume架构

1. Flume架构：Flume 的数据流是通过一系列称为Agent的组件构成的， Flume以Agent 为最小的独立运行单位， 一个Agent就是一个JVM。 Flume是一个完整的数据采集框架， 其基本架构及数据流动模型图如下图。 Flume含有三个核心组件， 分别是Source、 Channel、 Sink。 通过这些组件，Event可以从一个地方流向另一个地方。
    1. Event被外部Source（例如Web Server） 发送到Source， 被发送的Event要有特定的格式。（例如，Avro Source可以用来接收来自客户端的AvroEvent或者其他Flume Agent。）
    2. 当收到Event时， Source会将Event存储进一个或多个Channel。 该Channel是一个活动存储， 用于保存Event直至它被Sink消费。

3. Sink把Event从Channel中移除并把Event放进外部存储库， 如HDFS。Source和Sink在Agent里面是异步运行的。

2. 核心概念：
    1. Event： 一个数据单元， 带有一个可选的消息头。
    2. Flow： Event从源点到达目的点的迁移的抽象。
    3. Client： 操作位于源点处的Event， 将其发送到Flume Agent。
    4. Agent： 一个独立的Flume进程， 包含组件Source、 Channel、 Sink。
    5. Source： 用来消费传递到该组件的Event。
    6. Channel： 中转Event的一个临时存储， 保存有Source组件传递过来的Event。
    7. Sink： 从Channel中读取并移除Event， 将Event传递到Flow Pipeline中的下一个Agent。
3. 核心组件：Source；Channel；Sink



    1. Source：Source是数据的收集端， 负责将数据捕获后进行特殊的格式化， 将数据封装到事件（Event） 里， 然后将事件推入Channel中。Flume提供了各种Source的实现， 包括Avro Source、 Exec Source、Spooling Directory Source、 Net Cat Source、 Syslog Source、Syslog TCP Source、 Syslog UDP Source、 HTTP Source、HDFSSource等。 如果内置的Source无法满足需要， Flume还支持自定义Source。
    2. Channel：连接Source和Sink的组件， 可以看作一个数据的缓冲区（数据队列）。Channel可以将事件暂存到内存中， 也可以将事件持久化到本地磁盘上， 直到Sink处理完该事件。对于Channel， Flume提供了Memory Channel、 JDBC Chanel、FileChannel等缓存类型。 其中Memory Channel可以实现高速的吞吐， 但是无法保证数据的完整性。 在官方文档上已经建议使用File Channel替换Memory Recover Channel。
    3. Sink：取出Channel中的数据，进行相应的存储文件系统、数据库，或者提交到远程服务器。Flume也提供了各种Sink的实现，包括HDFS sink、Logger sink、 Avro sink、 File Roll sink、 Null sink、 HBase sink等。

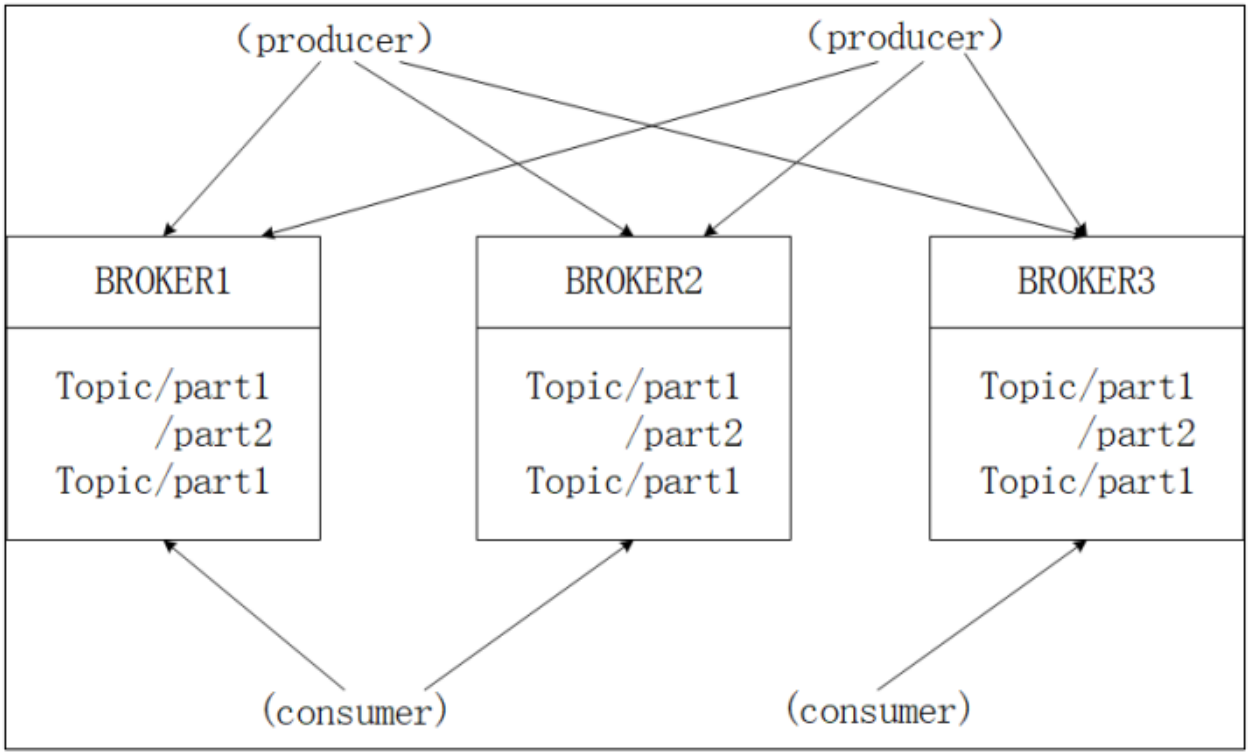
5.3.3 应用场景

在大数据背景下， Apache Flume作为一个分布式的、 可靠的数据采集软件系统， 主要从大量分散的数据源中收集、 汇聚以及迁移大规模的日志数据并进行存储。业界对于Flume这一开源分式技术的应用也在不断地拓展中， 以下总结了Flume的三大应用场景： 电子商务、 ETL工具、 银行事务。

5.4 Kafka概述

5.4.1 Kafka技术概述

1. 概念：Kafka是一款开源、 轻量级、 分布式、 可分区、 具有备份机制和基于ZooKeeper协调管理的消息系统， 也是一个功能强大的分布式流平台。
    1. 与传统的消息系统相比， Kafka能够很好地处理活跃的流数据， 使得数据在各个子系统中高性能、 低延迟地不停流转。
    2. 如果将前端应用产生的数据看成生产者（producer） 生产信息， 将后端的Hadoop、Spark等计算框架的执行流程看成消费者（consumer） 在消费并处理信息， 那么两者之间就需要一个沟通管理的桥梁以保证消息传输的效率与安全， 而Kafka则实现了生产者和消费者之间近似无缝的连接。
    3. Kafka的定位是一个分布式流处理平台， 满足3个关键特性：能够允许发布和订阅流数据， 在存储流数据时提供相应的容错机制， 当流数据到达时能够及时处理。
    4. Kafka最初由美国领英公司（LinkedIn） 公司开发， 并于2010年开源。 之后Kafka备受开源社区关注， 并成为Apache软件基金会的顶级项目。 Kafka是一款发布与订阅的消息系统， 具有高吞吐量、 分布式、 内置分区、 支持数据副本和容错的特性，适用于多种类型的数据管道和大规模信息处理。
    5. 随着技术不断完善， Kafka日益成为一个通用的数据管道， 同时兼具高性能和高伸缩性。 在LinkedIn中， Kafka既用于在线系统， 也用于离线系统；既从上游系统接收数据， 也会给下游系统输送数据；既提供消息的流转服务， 也用于数据的持久化存储。
2. Kafka基本框架：Kafka的基本框架包括生产者（producer） 、 消费者（consumer） 、 主题（Topic） 、 分区（partition） 和代理（broker）。Kafka的核心架构可以总结为生产者向Kafka服务器发送消息， 消费者从Kafka集群服务器读取消息， Kafka集群服务器依托ZooKeeper集群进行服务的协调管理。



    1. 生产者创建消息， 并将消息发布到特定的主题上。 在默认情况下生产者会将消息均衡地分布到主题的所有分区上， 而不关心特定消息会被写入哪个分区。 当然， 读者

也可以通过设置消息键和分区器为键生成一个散列值，并根据散列值将消息映射到指定的分区上， 保证同一个键的消息可以写入同一个分区中。

2. 消费者读取消息， 并按照消息生成的顺序读取它们。 消费者是消费者组的一部分，一个消费者组里的消费者订阅同一个主题， 意思是说会有一个或多个消费者共同读取一个主题。 通过这种方式， 消费者可以进行横向伸缩， 增加消费者，让它们分担负载， 分别处理部分分区的消息， 从而解决消费者消费数据的速度跟不上生产者生产数据的速度的问题。 而且当一个消费者发生故障， 群组里的其他消费者会接管其工作。
3. 主题与分区。 Kafka的消息通过主题进行分类。 主题是消息的归类， 可以理解为数据库的表， 或者文件系统里的文件夹。 主题可被细分为若干个分区， 分区则是消息的二次分类。 分区使得Kafka在并发处理上变得更加容易。 一般情况下， 分区数越多吞吐量越高， 但这要根据集群实际环境及业务场景确定。 同时，分区也是Kafka保证消息被顺序消费以及对消息进行负载均衡的基础。
4. Leader副本和Follower副本。 由于Kafka副本的存在， 所以我们需要保证一个分区的多个副本之间数据的一致性。 每个分区可以有多个副本， Kafka会选择该分区的一个副本作为Leader副本， 该分区的其他副本则作为Follower副本。只有Leader副本才负责处理客户端读写请求， Follower副本从Leader副本同步数据。
5. broker代理。 Kafka集群由一个或多个Kafka实例构成， 每一个Kafka实例称为代理（broker） 。 在生产环境中Kafka集群一般包括一台或多台服务器， 可以在一台服务器上配置一个或多个代理。 broker为消费者提供服务， 对其读取分区的请求作出反应， 返回已经提交到磁盘上的消息。 在特定的硬件及性能下，单个broker可以轻松处理数千个分区及每秒百万级的消息量。

3. Kafka的优势：
    1. 多个生产者。 Kafka可以无缝地支持多个生产者， 无论客户端使用单个主题还是多个主题。 所以适用于从多个前端系统收集数据， 并以统一的格式对外提供数据。
    2. 多个消费者。 Kafka支持多个消费者从一个单独的信息流上读取数据， 而且消费者之间互不干扰。 这与其他队列系统不同， 其他队列系统一旦被一个客户端读取， 其他客户端就无法读取它。
    3. 基于磁盘的数据存储。 在Kafka中消息被提交到磁盘， 并根据设置的保留规则进行保存。 每个主题可以设置单独的保留规则， 以便满足不同消费者的需求，也可以保留不同数量的信息。 消费者可能因为突发流量高峰期导致无法及时读取信息， 但因为Kafka中的消息可以放在磁盘中持久化保存， 消息将不会丢失，消费者依旧可以消费到之前的消息。
    4. 伸缩性。 为了能够处理海量数据， 开发者将Kafka设计为一个具有灵活伸缩性的消息订阅系统， 随着数据量不断增长， 用户可以扩展在线集群的节点而不影响系统的可用性， 一个Kafka集群可以包含上千个broker。
    5. 容错性。 假设Kafka集群有n个节点， 该集群在运行中即使有（n-1） 个节点出现故障， Kafka依旧能够正常运行。
    6. 高性能。 Kafka通过横向扩展生产者、 消费者和broker可以轻松处理巨大的信息流，还可以在处理海量数据的同时保证低延迟。

5.4.2 Kafka的应用场景

Kafka作为一款优秀的消息系统， 具有高吞吐量、 内置分区、 备份冗余分布式等特点， 为大规模消息处理提供了一种很好的解决方案。 以下为Kafka常见的应用场景。

1. 日志收集。 在日志收集上体现了Kafka支持多个生产者的优势， 收集多个应用程序的日志记录发布到主题上， 再通过Kafka以统一接口服务的方式开放给各种消费者。
2. 传递消息。 Kafka的一个基本用途是传递消息， 应用程序向用户发送通知就是通过传递消息实现的。 应用程序只需要产生消息， 而不需要关心消息的格式及消息如何被发送。 Kafka会通过格式化消息、 将多个消息放在同一个通知里发送、 最后根据用户配置的首选项来发送数据。
3. 用户活动跟踪。 Kafka经常用于收集网站用户与前端应用程序发生交互（如浏览网页、 搜索、 点击等） 时产生的用户活动相关信息， 并将这些消息发布到一个或多个主题上， 再通过消费者消费主题中的数据实现实时监控分析， 或将主题的数据直接保存到数据库中以便后续使用。
4. 运营指标。 Kafka经常被用于记录运营监控数据， 用户可以编写应用程序定期将数据发布到Kafka主题上， 以便监控系统或报警系统读取这些数据。
5. 流式处理。 Kafka可靠的传递能力让其成为流式处理系统完美的数据源， 很多基于Kafka构建的流式处理系统， 如Storm、 Spark Streaming、 Flink、Samza都是将Kafka作为可靠的数据来源。