技术栈

目录

[Overview 5](#_Toc491794884)

[Scala 6](#_Toc491794885)

[Java 7](#_Toc491794886)

[Spark 8](#_Toc491794887)

[Spark sql 8](#_Toc491794888)

[Data warehouse 8](#_Toc491794889)

[Spark Streaming 8](#_Toc491794890)

[Structured Streaming 8](#_Toc491794891)

[Memory db 10](#_Toc491794892)

[Alluxio 10](#_Toc491794893)

[Graph db 11](#_Toc491794894)

[TinkerPop3 11](#_Toc491794895)

[Neo4j 12](#_Toc491794896)

[TitanDB 12](#_Toc491794897)

[JanusGraph 12](#_Toc491794898)

[RPC & serialization 13](#_Toc491794899)

[Thrift 13](#_Toc491794900)

[Avro 13](#_Toc491794901)

[Protocol Buffers 13](#_Toc491794902)

[引用 14](#_Toc491794903)

[附录 15](#_Toc491794904)

# Overview

# Statistic learning

机器学习本质上属于应用统计学

统计学主要方法:

1. 频率派估计
2. 贝叶斯推断

# Machine learning

机器学习算法是一种能够从数据中学习的算法.

对于某类***任务T***和***性能度量P***,一个计算机程序被认为可以从***经验E***中学习是指,通过经验E改进后,它在任务T上有性能度量P衡量的性能有所提升

#### 任务T

通常机器学习任务定义为机器学习系统应该如何处理样本.

样本是指我们从某些希望机器学习系统处理的对象或事件中收集到的已经量化的特质feature的集合

1. 表示成向量:

机器学习任务

1. 分类
2. 输入缺失分类
3. 回归
4. 转录
5. 机器翻译
6. 结构化输出
7. 异常检测
8. 合成和采样
9. 缺失值填补
10. 去噪
11. 密度估计(概率质量函数估计)

#### 性能度量P

#### 经验E

监督学习:

无监督学习

随机梯度下降

# Deep learning

# 概率论

概率论是用于表示不确定性声明的数学框架;不仅提供了量化不确定性的方法,也提供了用于导出新的不确定性声明.

1. 告诉AI系统如何推理,据此设计算法来计算或估算有概率论导出的表达式
2. 用概率和统计从理论上分析我们提出的AI系统的行为

不确定性3种可能来源:

1. 被建模系统内在的随机性
2. 不完全观测
3. 不完全建模

随机变量:可以随机地取不同值的变量

概率分布:描述随机变量或一簇随机变量在每一个可能取到的状态的可能性大小

1. 离散型变量:
   1. 概率分布:概率质量函数P(x)
   2. 联合概率分布(作用于多个随机变量):P(x=*x*, y=*y*)
   3. :归一化
   4. 均匀分布:
2. 连续型变量:
   1. 概率分布:概率密度函数
   2. p的定义域必须是x所有可能状态的集合
      1. 落在区间[a,b]的概率是

边缘概率分布:定义在子集上的概率分布

条件概率:我们感兴趣的是某个事件,在给定其他事件发生时出现的概率

1. 条件概率的链式法则(乘法法则)
2. 独立性和条件独立性

贝叶斯规则

已知P(y|x)及P(x)时计算P(x|y); 注:

量化信息：

* 非常可能发生的事件信息量比较少，并在极端情况下，确保能够发生的事件应该没有信息量
* 较不可能发生的事件具有更高的信息量
* 独立事件应具有增量的信息

**自信息**：

结构化概率模型(图模型):用图来表示 把概率分布分解成许多因子的乘积形式

# 线性代数与微积分

# 开发架构设计

## 执行入口参数解析

## 功能路由调度

## 功能逻辑封装

## 公共模块抽象

## 面向对象开发应用

### 设计模式

### 继承

共用

### 封装

复用

### 多态

多种实现

### Single Responsibility Principle

一个类有且仅有一个职责，只有一个引起它变化的原因

### Open Closed Principle

软件实体如模块、类、函数应该对扩展开放，而对修改关闭

### Liskov Substitution Principle

所有引用基类的地方必须能透明地使用其子类的对象

### Interface Segregation Principle

不能强迫用户去依赖那些他们不使用的接口

使用多个功能单一、高内聚的接口总比使用一个庞大的接口要好

### Dependence Inversion Principle

高层模块不应该依赖底层模块，两者都应该依赖其抽象；抽象不应该依赖细节；细节应该依赖抽象

### Law Of Demeter

亦称为“最少知识原则（Principle of Least Knowledge）”

一个对象应该对其他对象保持最少的了解

### omposite/Aggregate Reuse Principle CARP

要尽量使用合成/聚合,尽量不要使用继承.

* 聚合表示的是整体和部分的关系，表示“含有”，整体由部分组合而成，部分可以脱离部分作为一个独立的个体而存在
* 组合是更强的“拥有”，部分组成整体，且不可分割，部分不能脱离整体而单独存在。组合关系中，部分和整体的生命周期一样，组合的新的对象完全支配其组成部分，包括它们的创建和湮灭等。一个组合关系的成分对象是不能与另一个组合关系共享的

### 对象模型要素

* 主要要素：
  + 抽象；
  + 封装；
  + 模块化；
  + 层次结构；
* 次要要素：
  + 类型
  + 持久
  + 并发

# 开发日志

经验法则

* 日志必须覆盖功能逻辑的“奇经八脉”
* 日志要有层次感
* 理想境界：输出的日志可以用代码进行分析
* 实践：代码记录日志，日志统一代码实现使用 slf4j

**private val** **LOG** **=** **LoggerFactory.**getLogger**(this.**getClass**)**

**private** **final** **static** Logger LOG **=** LoggerFactory**.**getLogger**(**WorldCountMR**.**class**);**

# 开发异常处理

经验法则

* 功能逻辑开发异常捕获

# Scala

# Java

## 设计架构

### 参数解析及功能调度

### 业务逻辑开发

#### 业务最小粒度处理

比如数据仓库数据导入，层次分别为dataWareHouse、database、table；此时业务最小处理逻辑为table，添加异常处理，使程序更具有健壮性；



### 公共层抽象

# Hadoop

## 角色服务

### Yarn

#### ResourceManager

#### NodeManager

#### HistoryServer

### HDFS

#### NameNode

#### DataNode

### High Available

# Spark

Spark philosophy: Unified engine for complete data applications & high-level user-friendly Apis

## Spark sql

### Data warehouse

#### Metadata

TABLE\_PARAMS :

|  |
| --- |
| TBL\_ID,PARAM\_KEY,PARAM\_VALUE  11,COLUMN\_STATS\_ACCURATE,false  11,numFiles,1  11,numRows,-1  11,rawDataSize,-1  11,spark.sql.partitionProvider,catalog  11,spark.sql.sources.schema.numParts,1  11,spark.sql.sources.schema.part.0,"{"type":"struct","fields":[{"name":"key","type":"string","nullable":true,"metadata":{}},{"name":"ZJLX\_ZJHM","type":"string","nullable":true,"metadata":{}},{"name":"PERSON\_ZJLX","type":"string","nullable":true,"metadata":{}},{"name":"PERSON\_ZJHM","type":"string","nullable":true,"metadata":{}}]}"  11,spark.sql.statistics.colStats.ZJLX\_ZJHM.avgLen,20  11,spark.sql.statistics.colStats.ZJLX\_ZJHM.distinctCount,3  11,spark.sql.statistics.colStats.ZJLX\_ZJHM.maxLen,21  11,spark.sql.statistics.colStats.ZJLX\_ZJHM.nullCount,0  11,spark.sql.statistics.colStats.ZJLX\_ZJHM.version,1  11,spark.sql.statistics.numRows,3  11,spark.sql.statistics.totalSize,191  11,totalSize,191  11,transient\_lastDdlTime,1505112813 |

***analysis table tablename compute statistics [for columns*** ***column\_name\_1, column\_name\_1, …]***

**Cost-Based Optimizer**

### Sql function [org.apache.spark.sql.functions](https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.functions$)

#### Aggregate functions

* approx\_count\_distinct : 功能等价于 count(distinct column)
  + Aggregate function: returns the approximate number of distinct items in a group

#### Collection functions

#### Date time functions

#### Math functions

#### Misc functions

* def crc32(e: Column): Column
* def hash(cols: Column\*): Column
* def md5(e: Column): Column
* def sha1(e: Column): Column
* def sha2(e: Column, numBits: Int): Column

#### Non-aggregate functions

#### Sorting functions

#### String function

#### UDF functions

#### Window functions

## Spark Streaming

## Structured Streaming

## Cost-Based Optimizer

### Table & Column Statistics

***analysis table tablename compute statistics [for columns column\_name\_1, column\_name\_1, …]***

### Benchmarks and Query Analysis

Hash join: Broadcast vs Shuffle

# Data mining

## 技术方法

### 统计学

### 机器学习

#### 监督学习

##### 分类 classification

##### 回归 regression

###### 线性回归line regression

###### 逻辑回归 logistic regression

#### 非监督学习

##### 聚类 clustering

##### 推荐 recommendation

##### 降维 dimensionality reducing

#### 半监督学习

#### 增强学习

### 数据仓库

Spark-sql

### 搜索引擎

Solr、ES

## 知识发现（KDD）过程

### 数据清理（消除噪声和删除不一致数据）

### 数据集成（多种数据源可以组合在一起）

### 数据选择（从数据库中提取与分析任务相关的数据）

### 数据变换（通过汇总或聚集操作，把数据变换和统一成适合挖掘的形式）

### 数据挖掘（基本步骤，使用智能方法提取数据模式）

### 模式评估（根据某种兴趣度度量，识别代表知识的真正有趣的模式）

### 知识表示（使用可视化和知识表示技术，向用户提供挖掘的知识）

## 挖掘频繁模式、关联和相关性

频繁模式：频繁的出现在数据集中的模式（如项集、子序列、子结构）

* 关联规则
  + 找出所有的频繁项集
  + 由频繁项集产生强关联规则
* 支持度
  + 最小支持度
* 置信度 confidence(A ==> B) = P(B | A) = support(A U B)/support(A) = support\_count(A U B)/support\_count(A)
  + 最小置信度

Apriori算法：通过限制候选产生发现频繁项集

* 先验性质：频繁项集的所有非空子集也一定是频繁的
  + ()
    - 连接步：通过将与自身连接产生**候选k** () 项集的集合
    - 剪枝步：扫描数据集D，对 每个候选计数；候选支持度计数与最小支持度比较进行过滤，得到 。

FP-growth算法：基于频繁模式增长，构造一个高度压缩的数据结构(FP树)，压缩原来的事务数据库

垂直数据格式的算法

# Deep learning

# 数据库设计

## 理论

### 3大范式

# Memory db

## [Alluxio](https://github.com/Alluxio/alluxio)

# Graph db

## [TinkerPop3](https://github.com/apache/tinkerpop)

## [Neo4j](https://github.com/neo4j/neo4j)

## [TitanDB](https://github.com/thinkaurelius/titan)

## [JanusGraph](https://github.com/JanusGraph/janusgraph)

# RPC & serialization

## Thrift

## Avro

## Protocol Buffers

# 认知构建

## 时间管理

* 进程切换非常昂贵，避免多任务，保持单进程。
* 研究表明，集中注意力、高效工作，每天最多3小时。
* 划分任务的优先级，不要把‘急切’当做‘重要’。
  + 

### 番茄工作法

# 引用

# 附录