技术栈

目录

[Overview 7](#_Toc503785846)

[Statistic learning 8](#_Toc503785847)

[感知机 8](#_Toc503785848)

[K近邻算法 8](#_Toc503785849)

[朴素贝叶斯法 9](#_Toc503785850)

[决策树 10](#_Toc503785851)

[逻辑斯谛回归与最大熵模型 10](#_Toc503785852)

[支持向量机 10](#_Toc503785853)

[提升方法 10](#_Toc503785854)

[EM算法及其推广 11](#_Toc503785855)

[隐马尔可夫模型 12](#_Toc503785856)

[条件随机场 12](#_Toc503785857)

[Machine learning 13](#_Toc503785858)

[线性模型 20](#_Toc503785859)

[Deep learning 24](#_Toc503785860)

[概率论 25](#_Toc503785861)

[线性代数与微积分 27](#_Toc503785862)

[开发架构设计 28](#_Toc503785863)

[执行入口参数解析 28](#_Toc503785864)

[功能路由调度 28](#_Toc503785865)

[功能逻辑封装 28](#_Toc503785866)

[公共模块抽象 28](#_Toc503785867)

[面向对象开发应用 28](#_Toc503785868)

[设计模式 28](#_Toc503785869)

[继承 29](#_Toc503785870)

[封装 29](#_Toc503785871)

[多态 29](#_Toc503785872)

[Single Responsibility Principle 29](#_Toc503785873)

[Open Closed Principle 29](#_Toc503785874)

[Liskov Substitution Principle 29](#_Toc503785875)

[Interface Segregation Principle 29](#_Toc503785876)

[Dependence Inversion Principle 30](#_Toc503785877)

[Law Of Demeter 30](#_Toc503785878)

[omposite/Aggregate Reuse Principle CARP 30](#_Toc503785879)

[对象模型要素 30](#_Toc503785880)

[开发日志 31](#_Toc503785881)

[开发异常处理 32](#_Toc503785882)

[Scala 34](#_Toc503785883)

[Java 35](#_Toc503785884)

[设计架构 35](#_Toc503785885)

[参数解析及功能调度 35](#_Toc503785886)

[业务逻辑开发 35](#_Toc503785887)

[公共层抽象 35](#_Toc503785888)

[Hadoop 36](#_Toc503785889)

[角色服务 36](#_Toc503785890)

[Yarn 36](#_Toc503785891)

[HDFS 36](#_Toc503785892)

[High Available 36](#_Toc503785893)

[测试 37](#_Toc503785894)

[Spark 38](#_Toc503785895)

[Spark sql 38](#_Toc503785896)

[Data warehouse 38](#_Toc503785897)

[Sql function org.apache.spark.sql.functions 39](#_Toc503785898)

[Spark Streaming 40](#_Toc503785899)

[Structured Streaming 40](#_Toc503785900)

[Cost-Based Optimizer 41](#_Toc503785901)

[Table & Column Statistics 41](#_Toc503785902)

[Benchmarks and Query Analysis 41](#_Toc503785903)

[Data mining 42](#_Toc503785904)

[技术方法 42](#_Toc503785905)

[统计学 42](#_Toc503785906)

[机器学习 42](#_Toc503785907)

[数据仓库 43](#_Toc503785908)

[搜索引擎 43](#_Toc503785909)

[知识发现（KDD）过程 43](#_Toc503785910)

[数据清理（消除噪声和删除不一致数据） 43](#_Toc503785911)

[数据集成（多种数据源可以组合在一起） 44](#_Toc503785912)

[数据选择（从数据库中提取与分析任务相关的数据） 44](#_Toc503785913)

[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） 44](#_Toc503785914)

[数据挖掘（基本步骤，使用智能方法提取数据模式） 44](#_Toc503785915)

[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） 44](#_Toc503785916)

[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） 44](#_Toc503785917)

[挖掘频繁模式、关联和相关性 44](#_Toc503785918)

[Deep learning platform 46](#_Toc503785919)

[Install & deploy 46](#_Toc503785920)

[Python 46](#_Toc503785921)

[Tersorflow 46](#_Toc503785922)

[Anaconda3 46](#_Toc503785923)

[PyCharm 46](#_Toc503785924)

[Learn & devlop 46](#_Toc503785925)

[Python 46](#_Toc503785926)

[Tensorflow 47](#_Toc503785927)

[TensorBoard 51](#_Toc503785928)

[Deep learning 53](#_Toc503785929)

[数据库设计 54](#_Toc503785930)

[理论 54](#_Toc503785931)

[3大范式 54](#_Toc503785932)

[Memory db 55](#_Toc503785933)

[Alluxio 55](#_Toc503785934)

[Graph db 56](#_Toc503785935)

[TinkerPop3 56](#_Toc503785936)

[Neo4j 57](#_Toc503785937)

[TitanDB 57](#_Toc503785938)

[JanusGraph 57](#_Toc503785939)

[RPC & serialization 58](#_Toc503785940)

[Thrift 58](#_Toc503785941)

[Avro 58](#_Toc503785942)

[Protocol Buffers 58](#_Toc503785943)

[Data science 59](#_Toc503785944)

[认知构建 60](#_Toc503785945)

[时间管理 60](#_Toc503785946)

[番茄工作法 61](#_Toc503785947)

[专利杂记 62](#_Toc503785948)

[引用 63](#_Toc503785949)

[代码 64](#_Toc503785950)

[tensorBoard 64](#_Toc503785951)

[附录 69](#_Toc503785952)

# 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

估计\偏差\方差

## 感知机

假设输入空间(特征空间)是, 输出空间是 .输入 表示实例的特征向量,对应于输入空间(特征空间)的点:输出表示实例的类别. 由输入空间到输出空间的如下函数: 称为感知机. w权值向量,b偏置

## K近邻算法

给定一个训练数据集,对新的输入实例,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,就把该输入实例分为这个类

k-nearest neighbor基本要素:

1. K值的选择
   1. K值减小:整体模型变得复杂，容易发生过拟合
   2. K值增大:近似误差增大,整体模型变得简单
   3. 通常采用交叉验证法来选取最优的K值
2. 距离度量(欧氏距离\曼哈顿距离)
3. 分类决策规则
   1. Kd tree

## 朴素贝叶斯法

## 决策树

1. 特征选择
   1. 决定用哪个特征来划分特征空间
2. 决策树的生成(局部最优)
   1. ID3算法
      1. 在决策树各个结点上应用信息增益准则选择特征,递归地构建决策树
   2. C4.5算法
      1. 用信息增益比来选择特征
3. 决策树的剪枝(全局最优)
   1. 简化决策树模型
   2. 极小化决策树整体损失函数或代价函数来实现
4. CART(classification and regression tree):

熵:表示随机变量不确定性的度量

## 逻辑斯谛回归与最大熵模型

最大熵原理:学习概率模型时,在所有可能的概率模型(分布)中,熵最大的模型是最好的模型.

## 支持向量机

## 提升方法

将弱可学习算法提升为强可学习算法的统计学习方法

在分类问题中，它通过改变训练样本的权重，学习多个分类器，并将这些分类器进行线性组合，提高分类的性能

思想：对于一个复杂任务来说，将多个专家的判断进行适当的综合所得出的判断，要比其中任何一个专家单独的判断好

在概率近似正确(probably approximately correct, PAC)学习框架中,一个概念(一个类),如果存在一个多项式的学习算法能够学习它,并且正确率很高,那么就称这个概念是强可学习的;一个概念,如果存在一个多项式的学习算法能够学习它,学习的正确率仅比随机猜测略好,那么就称这个概念是弱可学习的.

问题:

1. 在每一轮如何改变训练数据的权值或概率分布
2. 如何将弱分类组合成一个强分类器

## EM算法及其推广

E步:求期望(expectation)

M步:求极大(maximization)

期望极大算法(expectation maximization algorithm) 简单性\普适性

1. 观察变量:极大似然估计法\贝叶斯估计法 来估计模型参数
2. 隐变量:EM算法

## 隐马尔可夫模型

## 条件随机场

概率无向图模型是由无向图表示的联合概率分布.无向图上的结点之间的连续关系表示了联合分布的随机变量集合之间的条件独立性,即马尔可夫性.因此,概率无向图模型也称为马尔可夫随机场.

概率无向图模型或马尔可夫随机场的联合概率分布可以分解为无向图最大团上的正值函数的乘积的形式

条件随机场是给定输入随机变量X条件下,输出随机变量Y的条件概率分布模型,其形式为参数化的对数线性模型.条件随机场的最大特点是假设输出变量之间的联合概率分布构成概率无向图模型,即马尔可夫随机场.条件随机场是判别模型

# Machine learning

机器学习算法是一种能够从数据中学习的算法.

1. 特定数据集
2. 代价函数
3. 优化过程
4. 模拟

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.(*假设用P来评估计算机程序在某任务类T上的性能，若一个程序通过利用经验E在T中任务上获得了性能改善，则我们就说关于T和P，该程序对E进行了学习*.)

#### 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

决定机器学习算法效果是否好的因素:

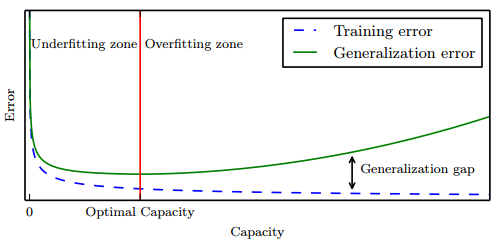
1. 降低训练误差
2. 缩小训练误差和测试误差的差距

欠拟合underfitting:指模型不能再训练集上获得足够低的误差

过拟合overfitting:指训练误差和测试误差之间的差距太大

奥卡姆剃刀:在同样能解释已知观察现象的假设中,我们应该挑选”最简单”的哪一个

容量:指其拟合各种函数的能力



没有免费午餐定理:在所有可能的数据生成分布上平均之后,每一个分类算法在未事先观测的点上都有相同的错误率

回归任务最常用的性能度量是”均方误差”

分类任务常用的性能度量:

1. 错误率
2. 精度

|  |  |  |
| --- | --- | --- |
| 真实情况 | 预测结果 | |
| 正例 | 反例 |
| 正例 | TP(真正例) | FN(假反例) |
| 反例 | FP(假正例) | TN(真反例) |

1. 查准率:
   1. 在商品推荐系统中,为了尽可能少打扰用户,更希望推荐内容确是用户感兴趣的,此时查准率更重要
   2. 真正例率:
2. 查全率:
   1. 而在逃犯信息检索系统中,更希望尽可能少漏掉逃犯,此时查全率更重要
   2. 假

二分类代价矩阵

|  |  |  |
| --- | --- | --- |
| 真实类别 | 预测类别 | |
| 第0类 | 第1类 |
| 第0类 | 0 |  |
| 第1类 |  | 0 |

代价敏感错误率与代价曲线

比较校验

1. 假设校验
2. 交叉验证t校验
3. McNemar校验
4. Friedman校验与Nemenyi后续校验

偏差与方差

### 线性模型

线性回归:试图学得一个线性模型以尽可能准确地预测实值

#### 经验E

##### 无监督学习

训练含有很多特征的**数据集(设计矩阵)**,然后学习出这个数据集上有用的结构性质(概率分布、密度估计、合成、去噪)

###### 主成分分析

###### K-means clustering

##### 监督学习

训练含有很多特征的数据集，不过数据集中的样本都有一个标签(label)或目标(target)

#### 随机梯度下降

Stochastic gradient descent

模型:全局性规则

模式:局部性结果

学习(训练):从数据中学得模型的过程

欲预测的是离散值:分类(classification)

欲预测的是连续值:回归(regression)

根据训练数据是否拥有标记信息,学习任务可大致划分为2大类:

1. 监督学习(supervised learning):分类\回归
2. 无监督学习(unsupervised learning):聚类

机器学习的目标是使学得的模型能很好地适用于”新样本”;

泛化(generalization):学得模型适用于新样本的能力

NFL(No free lunch theorem):没有免费的午餐

要讨论算法的相对优劣,必须要针对具体的学习问题

1. 50s-70s:推理期
2. 75s- :知识期
3. 95s- :统计学习

数据挖掘:从海量数据中发掘知识

统计学主要是通过机器学习对数据挖掘发挥影响,而机器学习领域和数据库领域则是数据挖掘的两大支撑

大数据时代3大技术:

1. 机器学习:提供数据分析能力
2. 云计算:提供数据处理能力
3. 众包(crowdsourcing):提供数据标记能力

数据集D 🡪 训练集S 和 测试集T

1. 留出法:将D划分为两个互斥的集合
   1. 窘境:若令训练集S包含绝大多数样本,则训练出的模型可能更接近于用D训练出的模型,但由于T比较小,评估结果可能不够稳定准确;(2/3 ~ 4/5 用于训练)
2. 交叉验证法(k折交叉验证):数据集D划分为k个大小的互斥子集(分层采样);
   1. 留一法: 子集包含一个元素
3. 自助法:

# 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

## 测试

各服务web页面可正常，eg:50070、60010 etc

功能性测试

相关命令测试，eg: hadoop dfs –ls /

Yarn job运行测试：

hadoop jar $HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples\*.jar pi 10 100

Spark job运行测试：

run-example --master yarn-cluster SparkPi 100

性能测试

Hdfs测试：

hadoop jar $HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples\*.jar randomwriter

Sort测试：

hadoop jar $HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples\*.jar sort

# Spark

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

For accumulator updates performed inside ***action(*reduce**(*func*)***)*** only, spark guarantee that each task’s update to the accumulator will only be applied once.

## Spark sql

积累：

REFRESH TABLE [db\_name**.**]table\_name -- 刷新表数据

### 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

# PlatformTools

## Zeppelin

### Install & run

|  |
| --- |
| 1. 下载：zeppelin-0.7.3-bin-netinst.tgz |
| 1. cd ${ZEPPELIN\_HOME} |
| 1. cp ./conf/zeppelin-site.xml.template ./conf/zeppelin-site.xml |
| 1. vim ./conf/zeppelin-site.xml #改变服务端口 |
|  |
| 1. Browser：open <http://hostname:8686> |
| 1. Config interpreter |
| 1. Keyboard shortcuts |
| 1. 实践 |
| 1. %spark.r    1. R本地安装package       1. install.packages("/usr/local/envTch/rPackage/knitr\_1.17.tar.gz ", repos = NULL, type="source") |

# 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 platform

## Install & deploy

### Python

|  |
| --- |
| 1. 安装python    1. Python 3.5.x 64-bit    2. Python 3.6.x 64-bit |

### Tersorflow

|  |
| --- |
| 1. pip3 install --upgrade tensorflow |
|  |

### Scikit-learn

|  |
| --- |
| 1. pip3 install --upgrade scikit-learn |
| 1. python -mpip install -U scipy |
|  |
|  |

### Anaconda3

|  |
| --- |
| 1. 安装Anaconda3(可选) |

### PyCharm

|  |
| --- |
| 1. 安装PyCharm |

### Jupyter

|  |
| --- |
| 1. python3 -m pip install --upgrade pip 2. python3 -m pip install jupyter |

## Learn & devlop

### Python

|  |
| --- |
| *# coding=utf-8* **import** sys  print(**"hello world"**) print(**"当前interpretor："**, sys.executable) |

### Tensorflow

|  |
| --- |
| *# coding=utf-8* **import** tensorflow **as** tf **import** sys  hello = tf.constant(**'Hello, TensorFlow!'**) sess = tf.Session() print(sess.run(hello))  print(sys.executable) |
| [*tensorBoard*](#_tensorBoard)*代码* |

### TensorBoard

|  |
| --- |
| 1. tensorboard --logdir=path/to/log-directory |
|  |
|  |

# Deep learning

# 数据库设计

## 理论

### 3大范式

## 实践

### Oracle

|  |
| --- |
| ORA-01000, the maximum-open-cursors error, is an extremely common error in Oracle database development. In the context of Java, it happens when the application attempts to open more ResultSets than there are configured cursors on a database instance.  Common causes are:   1. Configuration mistake    * You have more threads in your application querying the database than cursors on the DB. One case is where you have a connection and thread pool larger than the number of cursors on the database.    * You have many developers or applications connected to the same DB instance (which will probably include many schemas) and together you are using too many connections.    * **Solution:**      + [Increasing the number of cursors](https://stackoverflow.com/a/2583309/1395668) on the database (if resources allow) or      + Decreasing the number of threads in the application. 2. Cursor leak    * The applications is not closing ResultSets (in JDBC) or cursors (in stored procedures on the database)    * **Solution**: Cursor leaks are bugs; increasing the number of cursors on the DB simply delays the inevitable failure. Leaks can be found using [static code analysis](http://findbugs.sourceforge.net/), [JDBC](http://code.google.com/p/log4jdbc/) or application-level logging, and [database monitoring](http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html).  Background This section describes some of the theory behind cursors and how JDBC should be used. If you don't need to know the background, you can skip this and go straight to 'Eliminating Leaks'. What is a cursor? A cursor is a resource on the database that holds the state of a query, specifically the position where a reader is in a ResultSet. Each SELECT statement has a cursor, and PL/SQL stored procedures can open and use as many cursors as they require. You can find out more about cursors on [Orafaq](http://www.orafaq.com/wiki/Cursor).  A database instance typically serves several different schemas, many different users each with multiple sessions. To do this, it has a fixed number of cursors available for all schemas, users and sessions. When all cursors are open (in use) and request comes in that requires a new cursor, the request fails with an ORA-010000 error. Finding and setting the number of cursors The number is normally configured by the DBA on installation. The number of cursors currently in use, the maximum number and the configuration can be accessed in the Administrator functions in [Oracle SQL Developer](http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html). From SQL it can be set with:  ALTER SYSTEM SET OPEN\_CURSORS=1337 SID='\*' SCOPE=BOTH; Relating JDBC in the JVM to cursors on the DB The JDBC objects below are tightly coupled to the following database concepts:   * JDBC Connection is the client representation of a database session and provides database transactions. A connection can have only a single transaction open at any one time (but transactions can be nested) * A JDBC ResultSet is supported by a single cursor on the database. When close() is called on the ResultSet, the cursor is released. * A JDBC CallableStatement invokes a stored procedure on the database, often written in PL/SQL. The stored procedure can create zero or more cursors, and can return a cursor as a JDBC ResultSet.   JDBC is thread safe: It is quite OK to pass the various JDBC objects between threads.  For example, you can create the connection in one thread; another thread can use this connection to create a PreparedStatement and a third thread can process the result set. The single major restriction is that you cannot have more than one ResultSet open on a single PreparedStatement at any time. See [Does Oracle DB support multiple (parallel) operations per connection?](https://stackoverflow.com/questions/5947316/does-oracle-db-support-multiple-parallel-operations-per-connection)  Note that a database commit occurs on a Connection, and so all DML (INSERT, UPDATE and DELETE's) on that connection will commit together. Therefore, if you want to support multiple transactions at the same time, you must have at least one Connection for each concurrent Transaction. Closing JDBC objects A typical example of executing a ResultSet is:  Statement stmt = conn.createStatement();  try {  ResultSet rs = stmt.executeQuery( "SELECT FULL\_NAME FROM EMP" );  try {  while ( rs.next() ) {  System.out.println( "Name: " + rs.getString("FULL\_NAME") );  }  } finally {  try { rs.close(); } catch (Exception ignore) { }  }  } finally {  try { stmt.close(); } catch (Exception ignore) { }  }  Note how the finally clause ignores any exception raised by the close():   * If you simply close the ResultSet without the try {} catch {}, it might fail and prevent the Statement being closed * We want to allow any exception raised in the body of the try to propagate to the caller. If you have a loop over, for example, creating and executing Statements, remember to close each Statement within the loop.   In Java 7, Oracle has introduced the [AutoCloseable interface](http://docs.oracle.com/javase/7/docs/api/java/lang/AutoCloseable.html) which replaces most of the Java 6 boilerplate with some nice syntactic sugar. Holding JDBC objects JDBC objects can be safely held in local variables, object instance and class members. It is generally better practice to:   * Use object instance or class members to hold JDBC objects that are reused multiple times over a longer period, such as Connections and PreparedStatements * Use local variables for ResultSets since these are obtained, looped over and then closed typically within the scope of a single function.   There is, however, one exception: If you are using EJBs, or a Servlet/JSP container, you have to follow a strict threading model:   * Only the Application Server creates threads (with which it handles incoming requests) * Only the Application Server creates connections (which you obtain from the connection pool) * When saving values (state) between calls, you have to be very careful. Never store values in your own caches or static members - this is not safe across clusters and other weird conditions, and the Application Server may do terrible things to your data. Instead use stateful beans or a database. * In particular, never hold JDBC objects (Connections, ResultSets, PreparedStatements, etc) over different remote invocations - let the Application Server manage this. The Application Server not only provides a connection pool, it also caches your PreparedStatements.  Eliminating leaks There are a number of processes and tools available for helping detect and eliminating JDBC leaks:   1. During development - catching bugs early is by far the best approach:    1. Development practices: Good development practices should reduce the number of bugs in your software before it leaves the developer's desk. Specific practices include:       1. [Pair programming](http://en.wikipedia.org/wiki/Pair_programming), to educate those without sufficient experience       2. [Code reviews](http://en.wikipedia.org/wiki/Code_review) because many eyes are better than one       3. [Unit testing](http://en.wikipedia.org/wiki/Unit_testing) which means you can exercise any and all of your code base from a test tool which makes reproducing leaks trivial       4. Use [existing libraries](https://stackoverflow.com/questions/520585/connection-pooling-options-with-jdbc-dbcp-vs-c3p0) for connection pooling rather than building your own    2. Static Code Analysis: Use a tool like the excellent [Findbugs](http://findbugs.sourceforge.net/) to perform a static code analysis. This picks up many places where the close() has not been correctly handled. Findbugs has a plugin for Eclipse, but it also runs standalone for one-offs, has integrations into Jenkins CI and other build tools 2. At runtime:    1. Holdability and commit       1. If the ResultSet holdability is ResultSet.CLOSE\_CURSORS\_OVER\_COMMIT, then the ResultSet is closed when the Connection.commit() method is called. This can be set using Connection.setHoldability() or by using the overloaded Connection.createStatement() method.    2. Logging at runtime.       1. Put good log statements in your code. These should be clear and understandable so the customer, support staff and teammates can understand without training. They should be terse and include printing the state/internal values of key variables and attributes so that you can trace processing logic. Good logging is fundamental to debugging applications, especially those that have been deployed.       2. You can add a debugging JDBC driver to your project (for debugging - don't actually deploy it). One example (I have not used it) is [log4jdbc](http://code.google.com/p/log4jdbc/). You then need to do some simple analysis on this file to see which executes don't have a corresponding close. Counting the open and closes should highlight if there is a potential problem          1. Monitoring the database. Monitor your running application using the tools such as the SQL Developer 'Monitor SQL' function or [Quest's TOAD](http://www.quest.com/toad/). Monitoring is described in [this article](http://www.orafaq.com/node/758). During monitoring, you query the open cursors (eg from table v$sesstat) and review their SQL. If the number of cursors is increasing, and (most importantly) becoming dominated by one identical SQL statement, you know you have a leak with that SQL. Search your code and review.  Other thoughtsCan you use WeakReferences to handle closing connections? Weak and soft references are ways of allowing you to reference an object in a way that allows the JVM to garbage collect the referent at any time it deems fit (assuming there are no strong reference chains to that object).  If you pass a ReferenceQueue in the constructor to the soft or weak Reference, the object is placed in the ReferenceQueue when the object is GC'ed when it occurs (if it occurs at all). With this approach, you can interact with the object's finalization and you could close or finalize the object at that moment.  Phantom references are a bit weirder; their purpose is only to control finalization, but you can never get a reference to the original object, so it's going to be hard to call the close() method on it.  However, it is rarely a good idea to attempt to control when the GC is run (Weak, Soft and PhantomReferences let you know after the fact that the object is enqueued for GC). In fact, if the amount of memory in the JVM is large (eg -Xmx2000m) you might never GC the object, and you will still experience the ORA-01000. If the JVM memory is small relative to your program's requirements, you may find that the ResultSet and PreparedStatement objects are GCed immediately after creation (before you can read from them), which will likely fail your program.  **TL;DR:** The weak reference mechanism is not a good way to manage and close Statement and ResultSet objects. |

# 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

# Data science

任务

1. 将业务的挑战转化为分析的问题
2. 设计、实施、部署大数据的统计模型和数据挖掘技术
3. 产生能被用于指导实践的洞见

技能和行为特征

1. 量化分析技能
2. 技术能力
3. 怀疑性和批判性的思维
4. 好奇心和创造力
5. 沟通和协作能力

# 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

# 专利杂记

# 引用

# 代码

## tensorBoard

|  |
| --- |
| *# coding=utf-8 """A simple MNIST classifier which displays summaries in TensorBoard. This is an unimpressive MNIST model, but it is a good example of using tf.name\_scope to make a graph legible in the TensorBoard graph explorer, and of naming summary tags so that they are grouped meaningfully in TensorBoard. It demonstrates the functionality of every TensorBoard dashboard. """* **from** \_\_future\_\_ **import** absolute\_import **from** \_\_future\_\_ **import** division **from** \_\_future\_\_ **import** print\_function  **import** argparse **import** os **import** sys  **import** tensorflow **as** tf **from** tensorflow.examples.tutorials.mnist **import** input\_data  FLAGS = **None   def** train():  *# Import data* mnist = input\_data.read\_data\_sets(FLAGS.data\_dir,  one\_hot=**True**,  fake\_data=FLAGS.fake\_data)   sess = tf.InteractiveSession()  *# Create a multilayer model.   # Input placeholders* **with** tf.name\_scope(**'input'**):  x = tf.placeholder(tf.float32, [**None**, 784], name=**'x-input'**)  y\_ = tf.placeholder(tf.float32, [**None**, 10], name=**'y-input'**)   **with** tf.name\_scope(**'input\_reshape'**):  image\_shaped\_input = tf.reshape(x, [-1, 28, 28, 1])  tf.summary.image(**'input'**, image\_shaped\_input, 10)   *# We can't initialize these variables to 0 - the network will get stuck.* **def** weight\_variable(shape):  *"""Create a weight variable with appropriate initialization."""* initial = tf.truncated\_normal(shape, stddev=0.1)  **return** tf.Variable(initial)   **def** bias\_variable(shape):  *"""Create a bias variable with appropriate initialization."""* initial = tf.constant(0.1, shape=shape)  **return** tf.Variable(initial)   **def** variable\_summaries(var):  *"""Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""* **with** tf.name\_scope(**'summaries'**):  mean = tf.reduce\_mean(var)  tf.summary.scalar(**'mean'**, mean)  **with** tf.name\_scope(**'stddev'**):  stddev = tf.sqrt(tf.reduce\_mean(tf.square(var - mean)))  tf.summary.scalar(**'stddev'**, stddev)  tf.summary.scalar(**'max'**, tf.reduce\_max(var))  tf.summary.scalar(**'min'**, tf.reduce\_min(var))  tf.summary.histogram(**'histogram'**, var)   **def** nn\_layer(input\_tensor, input\_dim, output\_dim, layer\_name, act=tf.nn.relu):  *"""Reusable code for making a simple neural net layer.  It does a matrix multiply, bias add, and then uses ReLU to nonlinearize.  It also sets up name scoping so that the resultant graph is easy to read,  and adds a number of summary ops.  """  # Adding a name scope ensures logical grouping of the layers in the graph.* **with** tf.name\_scope(layer\_name):  *# This Variable will hold the state of the weights for the layer* **with** tf.name\_scope(**'weights'**):  weights = weight\_variable([input\_dim, output\_dim])  variable\_summaries(weights)  **with** tf.name\_scope(**'biases'**):  biases = bias\_variable([output\_dim])  variable\_summaries(biases)  **with** tf.name\_scope(**'Wx\_plus\_b'**):  preactivate = tf.matmul(input\_tensor, weights) + biases  tf.summary.histogram(**'pre\_activations'**, preactivate)  activations = act(preactivate, name=**'activation'**)  tf.summary.histogram(**'activations'**, activations)  **return** activations   hidden1 = nn\_layer(x, 784, 500, **'layer1'**)   **with** tf.name\_scope(**'dropout'**):  keep\_prob = tf.placeholder(tf.float32)  tf.summary.scalar(**'dropout\_keep\_probability'**, keep\_prob)  dropped = tf.nn.dropout(hidden1, keep\_prob)   *# Do not apply softmax activation yet, see below.* y = nn\_layer(dropped, 500, 10, **'layer2'**, act=tf.identity)   **with** tf.name\_scope(**'cross\_entropy'**):  *# The raw formulation of cross-entropy,  #  # tf.reduce\_mean(-tf.reduce\_sum(y\_ \* tf.log(tf.softmax(y)),  # reduction\_indices=[1]))  #  # can be numerically unstable.  #  # So here we use tf.nn.softmax\_cross\_entropy\_with\_logits on the  # raw outputs of the nn\_layer above, and then average across  # the batch.* diff = tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_, logits=y)  **with** tf.name\_scope(**'total'**):  cross\_entropy = tf.reduce\_mean(diff)  tf.summary.scalar(**'cross\_entropy'**, cross\_entropy)   **with** tf.name\_scope(**'train'**):  train\_step = tf.train.AdamOptimizer(FLAGS.learning\_rate).minimize(  cross\_entropy)   **with** tf.name\_scope(**'accuracy'**):  **with** tf.name\_scope(**'correct\_prediction'**):  correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1))  **with** tf.name\_scope(**'accuracy'**):  accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  tf.summary.scalar(**'accuracy'**, accuracy)   *# Merge all the summaries and write them out to  # /tmp/tensorflow/mnist/logs/mnist\_with\_summaries (by default)* merged = tf.summary.merge\_all()  train\_writer = tf.summary.FileWriter(FLAGS.log\_dir + **'/train'**, sess.graph)  test\_writer = tf.summary.FileWriter(FLAGS.log\_dir + **'/test'**)  tf.global\_variables\_initializer().run()   *# Train the model, and also write summaries.  # Every 10th step, measure test-set accuracy, and write test summaries  # All other steps, run train\_step on training data, & add training summaries* **def** feed\_dict(train):  *"""Make a TensorFlow feed\_dict: maps data onto Tensor placeholders."""* **if** train **or** FLAGS.fake\_data:  xs, ys = mnist.train.next\_batch(100, fake\_data=FLAGS.fake\_data)  k = FLAGS.dropout  **else**:  xs, ys = mnist.test.images, mnist.test.labels  k = 1.0  **return** {x: xs, y\_: ys, keep\_prob: k}   **for** i **in** range(FLAGS.max\_steps):  **if** i % 100 == 0: *# Record summaries and test-set accuracy* summary, acc = sess.run([merged, accuracy], feed\_dict=feed\_dict(**False**))  test\_writer.add\_summary(summary, i)  print(**'Accuracy at step %s: %s'** % (i, acc))  **else**: *# Record train set summaries, and train* **if** i % 100 == 99: *# Record execution stats* run\_options = tf.RunOptions(trace\_level=tf.RunOptions.FULL\_TRACE)  run\_metadata = tf.RunMetadata()  summary, \_ = sess.run([merged, train\_step],  feed\_dict=feed\_dict(**True**),  options=run\_options,  run\_metadata=run\_metadata)  train\_writer.add\_run\_metadata(run\_metadata, **'step%03d'** % i)  train\_writer.add\_summary(summary, i)  print(**'Adding run metadata for'**, i)  **else**: *# Record a summary* summary, \_ = sess.run([merged, train\_step], feed\_dict=feed\_dict(**True**))  train\_writer.add\_summary(summary, i)  train\_writer.close()  test\_writer.close()   **def** main(\_):  **if** tf.gfile.Exists(FLAGS.log\_dir):  tf.gfile.DeleteRecursively(FLAGS.log\_dir)  tf.gfile.MakeDirs(FLAGS.log\_dir)  train()   **if** \_\_name\_\_ == **'\_\_main\_\_'**:  parser = argparse.ArgumentParser()  parser.add\_argument(**'--fake\_data'**, nargs=**'?'**, const=**True**, type=bool,  default=**False**,  help=**'If true, uses fake data for unit testing.'**)  parser.add\_argument(**'--max\_steps'**, type=int, default=5000,  help=**'Number of steps to run trainer.'**)  parser.add\_argument(**'--learning\_rate'**, type=float, default=0.001,  help=**'Initial learning rate'**)  parser.add\_argument(**'--dropout'**, type=float, default=0.9,  help=**'Keep probability for training dropout.'**)  parser.add\_argument(  **'--data\_dir'**,  type=str,  default=os.path.join(os.getenv(**'TEST\_TMPDIR'**, **'/tmp'**),  **'tensorflow/mnist/input\_data'**),  help=**'Directory for storing input data'**)  parser.add\_argument(  **'--log\_dir'**,  type=str,  *# default=os.path.join(os.getenv('TEST\_TMPDIR', '/tmp'),  # 'tensorflow/mnist/logs/mnist\_with\_summaries'),* default=**"E:/developPlat/blog/source/\_posts/src/py/learnTF/data/tensorBoard/mnist"**,  help=**'Summaries log directory'**)  FLAGS, unparsed = parser.parse\_known\_args()  tf.app.run(main=main, argv=[sys.argv[0]] + unparsed) |

# 附录