大数据学院人工智能

技术亮点高频问题集

(2018版)

目录

# 一、 常见算法面试题

## 凸图形：

一个图形F，对在F内的任意两个点A、B，若线段AB上的所有点恒在图形F内，则称该图形F为凸图形。否则称之为非凸图形。

注：这里的图形F可以是平面图形、空间图形等。

## 监督学习、无监督学习和强化学习：

机器学习发展到现在，一般划分为监督学习(supervised learning)，半监督学习(semi-supervised learning)以及无监督学习(unsupervised learning)三类。举个具体的对应例子，则是比如说，在NLP词义消岐中，也分为监督的消岐方法，和无监督的消岐方法。在有监督的消岐方法中，训练数据是已知的，即每个词的语义分类是被标注了的；而在无监督的消岐方法中，训练数据是未经标注的。

上面所介绍的常见的分类算法属于监督学习，聚类则属于无监督学习(反过来说，监督学习属于分类算法则不准确，因为监督学习只是说我们给样本sample同时打上了标签（label），然后同时利用样本和标签进行相应的学习任务，而不是仅仅局限于分类任务。常见的其他监督问题，比如相似性学习，特征学习等等也是监督的，但是不是分类)。

再举个例子，正如人们通过已知病例学习诊断技术那样，计算机要通过学习才能具有识别各种事物和现象的能力。用来进行学习的材料就是与被识别对象属于同类的有限数量样本。监督学习中在给予计算机学习样本的同时，还告诉计算各个样本所属的类别。若所给的学习样本不带有类别信息,就是无监督学习(浅显点说：同样是学习训练，监督学习中，给的样例比如是已经标注了如心脏病的，肝炎的；而无监督学习中，就是给你一大堆的样例，没有标明是何种病例的)。

而在支持向量机导论一书给监督学习下的定义是：当样例是输入/输出对给出时，称为监督学习，有关输入/输出函数关系的样例称为训练数据。而在无监督学习中，其数据不包含输出值，学习的任务是理解数据产生的过程

### 常见的监督学习算法有哪些： 感知机、svm、人工神经网络、决策树、逻辑回归

监督学习：可用于一个特定的数据集（训练集）具有某一属性（标签），但是其他数据没有标签或者需要预测标签的情况。

无监督学习：可用于给定的没有标签的数据集（数据不是预分配好的），目的就是要找出数据间的潜在关系。

强化学习：位于这两者之间，每次预测都有一定形式的反馈，但是没有精确的标签或者错误信息。

常见的监督学习：决策树、朴素贝叶斯分类、最小二乘法、逻辑回归、支持向量机、集成方法

常见的无监督学习：聚类算法、主成分分析、奇异值分解、独立成分分析。

常用的分类器包括SVM、KNN、贝叶斯、线性回归、逻辑回归、决策树、随机森林、xgboost、GBDT、boosting、神经网络NN。

常见的降维方法包括TF-IDF、主题模型LDA、主成分分析PCA等等

常用的回归技术：线性回归、多项式回归、逐步回归、岭回归、套索回归、ElasticNet回归（是套索回归和岭回归技术的混合体）

## 脚本语言：

脚本英文为Script。实际上脚本就是程序，一般都是有应用程序提供的编程语言。应用程序包括浏览器(JavaScript、VBScript)、多媒体创作工具，应用程序的宏和创作系统的批处理语言也可以归入脚本之类。脚本同我们平时使用的VB、C语言的区别主要是:

· 1、脚本语法比较简单，比较容易掌握;

·2、脚本与应用程序密切相关，所以包括相对应用程序自身的功能;

· 3、脚本一般不具备通用性，所能处理的问题范围有限。

·4、脚本多为解释执行。

## LDA：

LDA(Latent Dirichlet Allocation)是一种文档主题生成模型，也称为一个三层贝叶斯概率模型，包含词、主题和文档三层结构。所谓生成模型，就是说，我们认为一篇文章的每个词都是通过"以一定概率选择了某个主题，并从这个主题中以一定概率选择某个词语"这样一个过程得到。文档到主题服从多项式分布，主题到词服从多项式分布。

LDA是一种非监督机器学习技术，可以用来识别大规模文档集(document collection)或语料库(corpus)中潜藏的主题信息。它采用了词袋(bag of words)的方法，这种方法将每一篇文档视为一个词频向量，从而将文本信息转化为了易于建模的数字信息。但是词袋方法没有考虑词与词之间的顺序，这简化了问题的复杂性，同时也为模型的改进提供了契机。每一篇文档代表了一些主题所构成的一个概率分布，而每一个主题又代表了很多单词所构成的一个概率分布

## 算子 ：

广义的讲，对任何函数进行某一项操作都可以认为是一个算子，甚至包括求幂次，开方都可以认为是一个算子，只是有的算子我们用了一个符号来代替他所要进行的运算罢了，所以大家看到算子就不要纠结，他和f(x)的f没区别，它甚至和加减乘除的基本运算符号都没有区别，只是他可以对单对象操作罢了(有的符号比如大于、小于号要对多对象操作)。又比如取概率P{X<x}，概率是集合{X<x}(他是属于实数集的子集)对[0,1]区间的一个映射，我们知道实数域和[0,1]区间是可以一一映射的(这个后面再说)，所以取概率符号P，我们认为也是一个算子，和微分，积分算子算子没区别。总而言之，算子就是映射，就是关系，就是变换。

## 停用词：

停用词是指在信息检索中，为节省存储空间和提高搜索效率，在处理自然语言数据(或文本)之前或之后会自动过滤掉某些字或词，这些字或词即被称为Stop Words(停用词)。这些停用词都是人工输入、非自动化生成的，生成后的停用词会形成一个停用词表。但是，并没有一个明确的停用词表能够适用于所有的工具。甚至有一些工具是明确地避免使用停用词来支持短语搜索的。

## RNN：

RNN的目的使用来处理序列数据。在传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是这种普通的神经网络对于很多问题却无能无力。例如，你要预测句子的下一个单词是什么，一般需要用到前面的单词，因为一个句子中前后单词并不是独立的。RNN之所以称为循环神经网路，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。理论上，RNNs能够对任何长度的序列数据进行处理。但是在实践中，为了降低复杂性往往假设当前的状态只与前面的几个状态相关。

RNN的目的使用来处理序列数据。在传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是这种普通的神经网络对于很多问题却无能无力。例如，你要预测句子的下一个单词是什么，一般需要用到前面的单词，因为一个句子中前后单词并不是独立的。RNNs之所以称为循环神经网路，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。理论上，RNNs能够对任何长度的序列数据进行处理。但是在实践中，为了降低复杂性往往假设当前的状态只与前面的几个状态相关。我觉得可以这样理解RNN，把多个传统神经网络的隐藏层神经元连接起来，每个神经网络表示某一时刻输入输出的一个模型。这样把多个连续时刻的模型连在一起，就可以结合上下文（前后时刻的内容）来进行分析预测了。

RNN有一个缺陷：随着时间距离的增大，RNN会丧失联系距它很远信息的能力

## CNN：

卷积神经网络。在机器学习中，卷积神经网络是一种深度前馈人工神经网络，已成功地应用于图像识别。

卷积神经网络，是一种前馈神经网络，人工神经元可以响应周围单元，可以进行大型图像处理。卷积神经网络包括卷积层和池化层。

卷积神经网络包括一维卷积神经网络、二维卷积神经网络以及三维卷积神经网络。一维卷积神经网络常应用于序列类的数据处理;二维卷积神经网络常应用于图像类文本的识别;三维卷积神经网络主要应用于医学图像以及视频类数据识别。

## SPARK：

Apache Spark是一个开源的集群运算框架,最初是由加州大学柏克莱分校AMPLab所开发.相对于Hadoop的MapReduce会在运行完工作后将中介数据存放到磁盘中,Spark使用了内存运算技术,能在数据尚未写入硬盘时在内存分析运算.Spark在内存内运算速度能做到比Hadoop MapReduce的运算速度快100倍,即便是运行程序于硬盘时，Spark也能快上10倍速度。[1]Spark允许用户将数据加载至集群内存，并多次对其进行查询，非常适合用于机器学习算法。

使用Spark需要搭配集群管理员和分布式存储系统。Spark支持独立模式（本地Spark集群）、Hadoop YARN或Apache Mesos的集群管理。[3] 在分布式存储方面，Spark可以和HDFS[4]、 Cassandra[5] 、OpenStack Swift和Amazon S3等接口搭载。 Spark也支持伪分布式（pseudo-distributed）本地模式，不过通常只用于开发或测试时以本机文件系统取代分布式存储系统。在这样的情况下，Spark仅在一台机器上使用每个CPU核心运行程序。

### Spark的特点:

快, 易用, 通用,兼容性

1. 快:

与Hadoop的MapReduce相比,Spark基于内存的运算速度要快100倍上,即使基于硬盘的运算速度也快10倍.Spark实现了搞笑的DAG的执行引擎,从而可以通过内存来高效处理流数据

1. 易用:

Spark 支持 Java、Python 和 Scala 的 API，还支持超过 80 种高级算法，使用户可以 快速构建不同的应用。而且 Spark 支持交互式的 Python 和 Scala 的 shell，可以非常方 便地在这些 shell 中使用 Spark 集群来验证解决问题的方法

1. 通用:

Spark提供了统一的解决方案.Spark可以用于批处理,交互式查询(Spark SQL).实时流处理(Spark Streaming),机器学习和图计算(GraphX),这些不同类型的处理都可以在同一应用中无缝使用.Spark统一的解决方案非常具有吸引力,毕竟任何公司都想用统一的平台去处理遇到的问题，减少开发和维护的人力成本和部署平台的物力成本。

另外Spark还可以很好的融入Hadoop的体系结构中可以直接操作HDFS,提供Hive on Spark , Pig on Sparkde 框架集群成Hadoop.

1. 兼容性:

Spark 可以非常方便地与其他的开源产品进行融合。比如，Spark 可以使用Hadoop 的 YARN 和 Apache Mesos 作为它的资源管理和调度器.并且可以处理所有 Hadoop 支持的数据，包括 HDFS、HBase 和 Cassandra 等。这对于已经部署Hadoop 集群的用户特别重要，因为不需要做任何数据迁移就可以使用 Spark 的强大处理能力。Spark 也可以不依赖于第三方的资源管理和调度器，它实现了Standalone 作为其内置的资源管理和调度框架，这样进一步降低了 Spark 的使用门槛，使得所有人都可以非常容易地部署和使用 Spark。此外，Spark 还提供了在EC2 上部Standalone 的 Spark 集群的工具。

## 模型评估指标：

### 混淆矩阵

实际\预测 1[T] 0[F]

1 [P](正例) TP FP

0 [N](负例) TN FN

解释: TP:正确地预测正例

FP:错误地预测正例

### 准确率

accuracy =(TP+FN)/(TP+FP+TN+FN)

准确率=正确预测正负例的个数/总个数

### 精确率

精确率(精准率,查准率)=正确预测正例的个数/实际正例的个数 [命中率=正确预测正的个数/预测正的个数]

召回率：

覆盖率(又叫召回率,查全率) =正确预测正例的个数/正确预测正负例的个数

### F1分数

F1分数(平衡F分数)=2(精准率\*召回率)/(精准率+召回率) 即：精准率和召回率的调和平均数

### AUC值

是ROC曲线下面的面积

使用AUC值作为评价标准是因为很多时候ROC曲线并不能清晰的说明哪个分类器的效果更好，而作为一个数值，对应AUC更大的分类器效果更好。

### ROC曲线

曲线越接近左上角，模型越好

## IF-IDF：

TF-IDF的主要思想是：如果某个词或短语在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类

TF-IDF实际上是：TF \* IDF，TF词频(Term Frequency)，IDF逆向文件频率(Inverse Document Frequency)。TF表示词条在文档d中出现的频率。IDF的主要思想是：如果包含词条t的文档越少，也就是n越小，IDF越大，则说明词条t具有很好的类别区分能力。

词频（term frequency，TF）指的是某一个给定的词语在该文件中出现的频率。这个数字是对词数(term count)的归一化，以防止它偏向长的文件。

TF =某关键词出现次数/文章中关键词总数

TF =某关键词出现次数/文章中出现最多次数关键词的出现次数

逆向文件频率（inverse document frequency，IDF）是一个词语普遍重要性的度量。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到

IDF =log（语料库文档总数/（包含该词的文档数+1））

之所以要+1是为了防止分母为0。由此可见，当一个词被越多的文档包含，则IDF值就越小，也就是说这个词很常见，不是最重要的能区分文章特性的关键词

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。TF-IDF法的精度并不是不高

## 贝叶斯：

### 条件概率：

（又称后验概率）就是事件A在另外一个事件B已经发生条件下的发生概率。条件概率表示为P(A|B)，读作“在B条件下A的概率”

### 联合概率：

表示两个事件共同发生的概率。A与B的联合概率表示为或者

### 边缘概率：

（又称先验概率）是某个事件发生的概率。边缘概率是这样得到的：在联合概率中，把最终结果中那些不需要的事件通过合并成它们的全概率，而消去它们（对离散随机变量用求和得全概率，对连续随机变量用积分得全概率），这称为边缘化（marginalization），比如A的边缘概率表示为P(A)，B的边缘概率表示为P(B)。

接着，考虑一个问题：P(A|B)是在B发生的情况下A发生的可能性。

首先，事件B发生之前，我们对事件A的发生有一个基本的概率判断，称为A的先验概率，用P(A)表示；

其次，事件B发生之后，我们对事件A的发生概率重新评估，称为A的后验概率，用P(A|B)表示；

类似的，事件A发生之前，我们对事件B的发生有一个基本的概率判断，称为B的先验概率，用P(B)表示；

同样，事件A发生之后，我们对事件B的发生概率重新评估，称为B的后验概率，用P(B|A)表示。

### 朴素贝叶斯：

朴素贝叶斯法是基于贝叶斯定理与特征条件独立假设的分类方法

事件A和B同时发生的概率为在A发生的情况下发生B或者在B发生的情况下发生A： P(A∩B)=P(A)∗P(B|A)=P(B)∗P(A|B)

所以有： P(A|B)=P(B|A)∗P(A)/P(B)

优点：

对小规模的数据表现很好，适合多分类任务，适合增量式训练。

缺点：

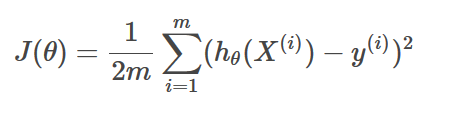
对输入数据的表达形式很敏感（离散、连续，值极大极小之类的）

## 线性回归：

般用来做连续值的预测，预测结果为一个连续值。训练时学习样本提供学习的特征向量，还要提供样本的实际结果。所以他是一种有监督学习。

h(x)=θtX

在通过学习得到的映射函数h(x)中，需要通过训练集得到特征系数向量θ={θ1,θ2,...θn}.

代价函数：

其中h(x)为需要学习的函数，m为训练集样本的个数。xi表示训练集中第i个样本的特征向量，yi表示第i个样本的标签。

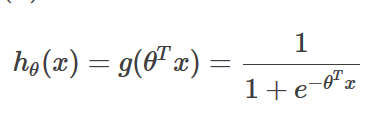
梯度下降就是一个不断地最小化损失函数的过程。

## 逻辑回归：

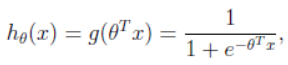
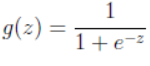
### 原理:

是处理二分类问题。为了实现逻辑回归分类器，我们可以在线性回归的基础上。添加一个sigmoid函数，进而得到一个范围在0~1之间的数值。任何大于0.5的数据会被分为1类，小于0.5即被分为0类。得到sigmoid函数，简单来说，是为了将标签归类为[0,1]的范围内

预测值为



LR回归是一个线性的二分类模型，主要是计算在某个样本特征下事件发生的概率，比如根据用户的浏览购买情况作为特征来计算它是否会购买这个商品，抑或是它是否会点.击这个商品。然后LR的最终值是根据一个线性和函数再通过一个sigmod函数来求得，这个线性和函数权重与特征值的累加以及加上偏置求出来的，所以在训练LR时也就是在训练线性和函数的各个权重值w

### 优缺点：

优点：

* + - 1. 实现简单
      2. 分类是计算量小，速度快，存储资源低

缺点：

1. 容易欠拟合
2. 只能处理二分类问题（softmax可以用于多分类），且必须现性可分

### 逻辑回归为什么要对特征进行离散化：

1. 离散特征的增加和减少都很容易，易于模型的快速迭代；
2. 稀疏向量内积乘法运算速度快，计算结果方便存储，容易扩展；
3. 离散化后的特征对异常数据有很强的鲁棒性：比如一个特征是年龄>30是1，否则0。如果特征没有离散化，一个异常数据“年龄300岁”会给模型造成很大的干扰；
4. 逻辑回归属于广义线性模型，表达能力受限；单变量离散化为N个后，每个变量有单独的权重，相当于为模型引入了非线性，能够提升模型表达能力，加大拟合；
5. 离散化后可以进行特征交叉，由M+N个变量变为M\*N个变量，进一步引入非线性，提升表达能力；
6. 特征离散化后，模型会更稳定，比如如果对用户年龄离散化，20-30作为一个区间，不会因为一个用户年龄长了一岁就变成一个完全不同的人。当然处于区间相邻处的样本会刚好相反，所以怎么划分区间是门学问；
7. 特征离散化以后，起到了简化了逻辑回归模型的作用，降低了模型过拟合的风险

## 决策树：

### 什么是决策树：

所谓决策树，顾名思义，是一种树，一种依托于策略抉择而建立起来的树。

    机器学习中，决策树是一个预测模型；他代表的是对象属性与对象值之间的一种映射关系。树中每个节点表示某个对象，而每个分叉路径则代表的某个可能的属性值，而每个叶结点则对应从根节点到该叶节点所经历的路径所表示的对象的值。决策树仅有单一输出，若欲有复数输出，可以建立独立的决策树以处理不同输出。从数据产生决策树的机器学习技术叫做决策树学习, 通俗点说就是决策树，说白了，这是一种依托于分类、训练上的预测树，根据已知预测、归类未来。

### 分类树：

输出叶子节点中所属类别最多的哪一类

### 回归树：

输出叶子节点中各个样本值的平均值

### 理想的决策树：

1. 叶子节点数量尽量少
2. 叶子节点的深度尽量小（太深可能会过拟合）

### 解决决策树的过拟合：

1. 剪枝
   1. 前置剪枝：在分裂节点的时候设计比较苛刻的条件，如不满足则直接停止分裂（这样干决策树无法到最优，也无法得到比较好的效果）
   2. 后置剪枝：在树建立完之后，用单个节点代替子树，节点的分类采用子树中主要的分类（这种方法比较浪费前面的建立过程）
2. 交叉验证
3. 随即森林

### 优缺点：

优点：

计算量简单，可解释性强，比较适合处理有缺失属性值的样本，能够处理不相关的特征；

分类精度高；

生成的模式简单；

对噪声数据有很好的健壮性

缺点：

单颗决策树分类能力弱，并且对连续值变量难以处理；

容易过拟合（后续出现了随机森林，减小了过拟合现象）

### ID3, C4.5,CART

C4.5算法是机器学习算法中的一种分类决策树算法，核心是ID3算法

C4.5算法产生的分类规则易理解，准确率较高

ID3算法用信息增益来选择属性

C4.5算法用信息增益率来选择属性，继承了ID3算法的优点。

CART决策树使用基尼指数来选择划分属性，基尼值的定义如下:

直观来说，GIni(D)反映了从数据集D中随机抽取两个样本，其类别标记不一样的概率，因此GIni(D)越小，则数据集D纯度越高。

### 1.预剪枝 2.后剪枝

#### 预剪枝

1. 预先设定一个树的最大深度，如果分裂后的树的深度达到了阈值，则停止分裂。
2. 设定一个分裂阈值，若分裂得到信息增益不大于这个阈值，则停止分裂。
3. 按照树的准确度，如果分裂后树的准确度提升，则分裂。

#### 优缺点:

预剪枝使得决策树很多的分支都没有展开，这不仅降低了过拟合的风险，还显著减少了决策树训练和预测的时间。所以基于贪心算法的预剪枝可能带来欠拟合的风险。

#### 后剪枝

首先生成与训练集完全拟合的决策树，然后自下而上地逐层剪枝，如果一个节点的子节点被删除后，决策树的准确度没有降低。那么就将该节点设置为叶节点。（基于的原则是Occam剪刀：具有相似效果的两个模型选择较简单的那个）

决策树算法的优点如下：

## SVM：

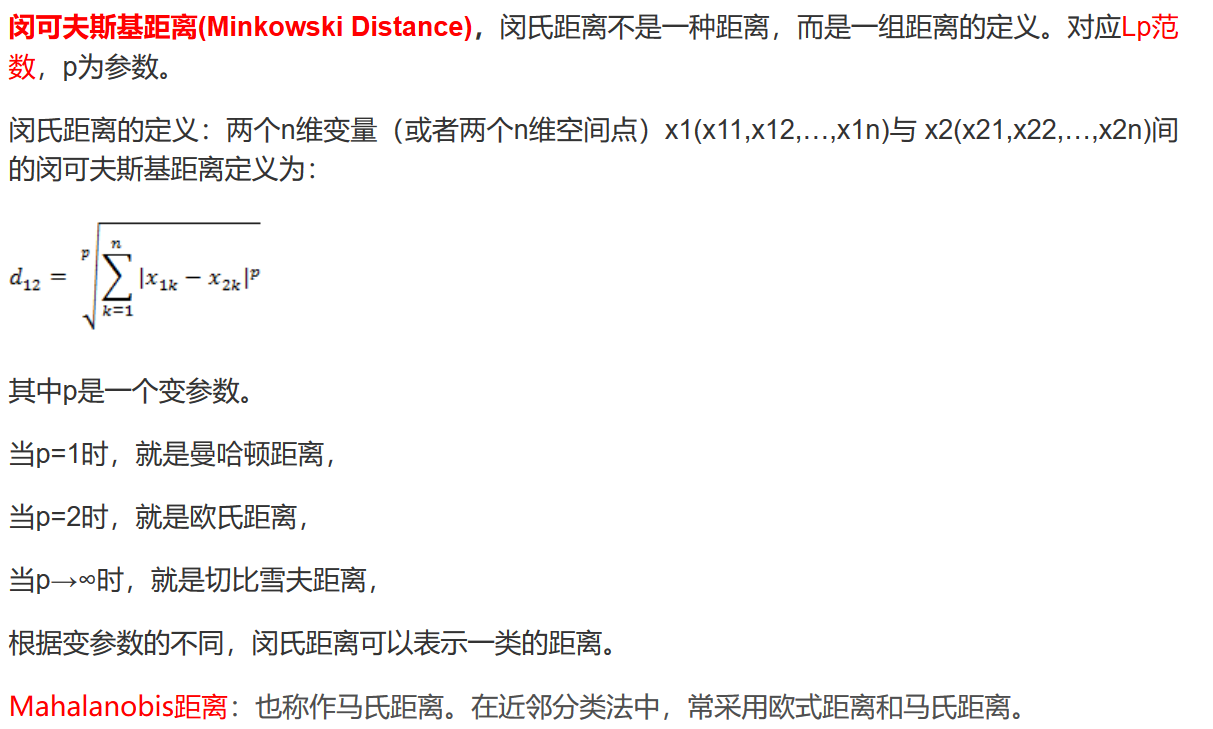
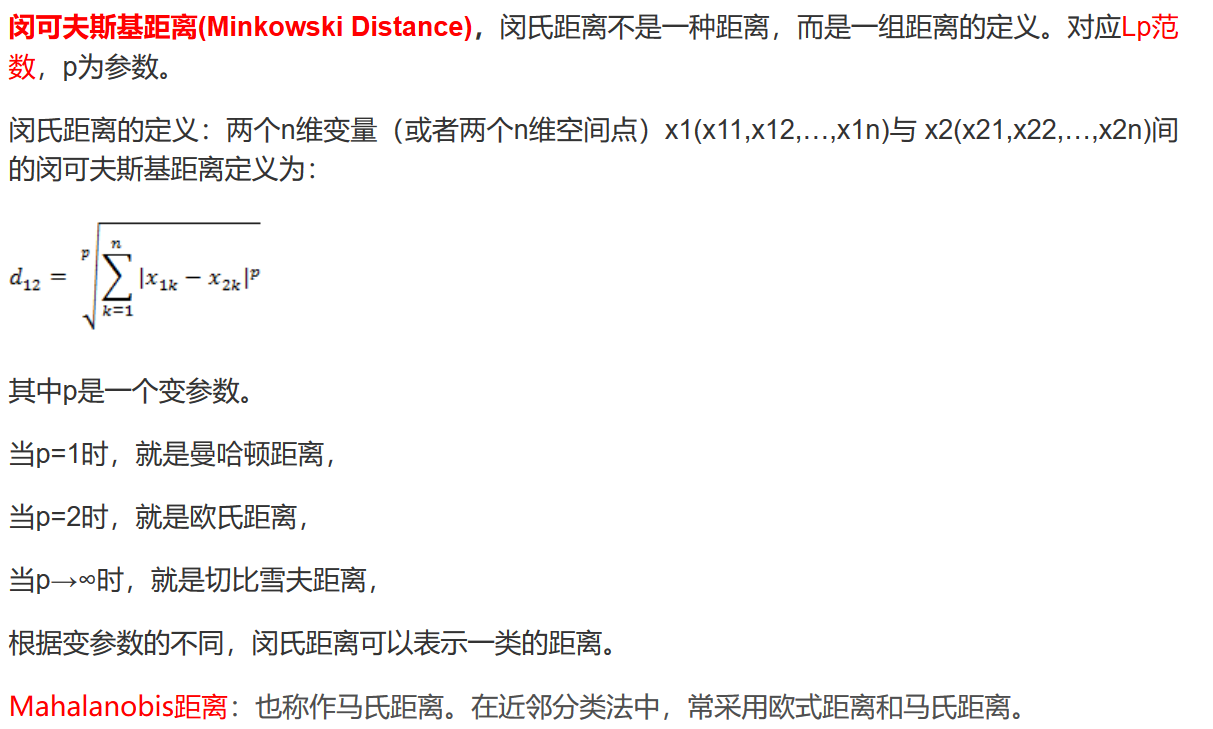
向量的范数可以简单形象的理解为向量的长度，或者向量到零点的距离，或者相应的两个点之间的距离。

### 常用的向量的范数:

L1范数:||x||为x向量各个元素绝对值之和 对应于曼哈顿距离

L2范数:||x||为x向量各个元素平方和的1/2次方。 对应于欧式距离

Lp范数:||x||为x向量各个元素绝对值p次方和的1/p次方 对应于闵可夫斯基距离Lꝏ范数:||x||为x向量各个元素绝度值最大那个元素的绝对值



中文名叫支持向量机。是一个面向数据的分类算法，基本模型是在特征空间上找到最佳的分离超平面使得训练集上正负样本间隔最大。SVM是用来解决二分类问题的有监督学习算法，在引入了核方法之后也可以用来解决非线性问题。

### 简单了解SVM是什么?

Svm就是一个分类器，它用于回归的时候称为SVR(Support Vector Regression ),SVM和SVR本质上是一样的。

svm用于解决分类分体，基于逻辑回归有大边界。相当于在支持向量机中嵌入一个额外的安全因子。（大间距）

### 一般SVM有下面三种：

硬间隔支持向量机（线性可分支持向量机）：当训练数据线性可分时，可通过硬间隔最大化学得一个线性可分支持向量机。硬间隔支持向量机要求所有的样本均被最佳超平面正确划分

软间隔支持向量机：当训练数据近似线性可分时，可通过软间隔最大化学得一个线性支持向量机。软间隔支持向量机允许某些样本点不满足间隔大于等于1的条件

非线性支持向量机：当训练数据线性不可分时，可通过核方法以及软间隔最大化学得一个非线性支持向量机

### 常用核函数：

线性核

多项式核

高斯核(RBF)

拉普拉斯核

Sigmoid核

### 优缺点：

优点：

1、使用核函数可以向高维空间进行映射

2、使用核函数可以解决非线性的分类

3、分类思想很简单，就是将样本与决策面的间隔最大化

4、分类效果较好

5、在中小量样本规模的时候容易得到数据和特征之间的非线性关系

缺点：

1、对大规模数据训练比较困难

2、无法直接支持多分类，但是可以使用间接的方法来做

3、对缺失数据敏感，对非线性问题没有通用的解决方案，核函数的正确选择不容易，计算复杂度高，主流的算法可以达到的复杂度，这对大规模的数据是吃不消的

## Tensorflow的计算图：

Tensorflow是一个通过计算图的形式来表述计算的编程系统，计算图也叫数据流图，可以把计算图看做是一种有向图，Tensorflow中的每一个节点都是计算图上的一个Tensor, 也就是张量，而节点之间的边描述了计算之间的依赖关系(定义时)和数学操作(运算时)



## 欧几里得距离和曼哈顿距离的差别：

### 欧氏距离：

最常见的两点之间或多点之间的距离表示法，又称之为欧几里得度量，它定义于欧几里得空间中。欧氏距离虽然很有用，但也有明显的缺点。它将样品的不同属性（即各指标或各变量量纲）之间的差别等同看待，这一点有时不能满足实际要求。例如，在教育研究中，经常遇到对人的分析和判别，个体的不同属性对于区分个体有着不同的重要性。因此，欧氏距离适用于向量各分量的度量标准统一的情况。

### 曼哈顿距离：

我们可以定义曼哈顿距离的正式意义为L1-距离或城市区块距离，也就是在欧几里得空间的固定直角坐标系上两点所形成的线段对轴产生的投影的距离总和，例如在平面上，坐标（x1, y1）的点P1与坐标（x2, y2）的点P2的曼哈顿距离为：，要注意的是，曼哈顿距离依赖座标系统的转度，而非系统在座标轴上的平移或映射。当坐标轴变动时，点间的距离就会不同。通俗来讲，想象你在曼哈顿要从一个十字路口开车到另外一个十字路口，驾驶距离是两点间的直线距离吗？显然不是，除非你能穿越大楼。而实际驾驶距离就是这个“曼哈顿距离”，这也是曼哈顿距离名称的来源， 同时，曼哈顿距离也称为城市街区距离(City Block distance)。曼哈顿距离和欧式距离一般用途不同，无相互替代性

## 过拟合：

随着训练过程的进行，模型复杂度增加，在training data上的error渐渐减小，但是在验证集上的error却反而渐渐增大——因为训练出来的网络过拟合了训练集, 对训练集外的数据却不work, 这称之为泛化(generalization)性能不好。泛化性能是训练的效果评价中的首要目标，没有良好的泛化，就等于南辕北辙, 一切都是无用功

### 过拟合原因：

数据不规范,数据量少,数据穿越,统计特征用到了未来的信息或者标签信息

算法过于复杂

一些假设条件（如样本独立同分布）可能是不成立的

训练样本过少不能对整个空间进行分布估计

### 解决方案：

1、将数据规范化，处理缺失值，增加数据量，重采样，添加噪声数据

2、正则化，控制模型复杂程度，

3、early stoping，减少迭代次数，减少树的深度，

4、学习率调大/小点、

5、融合几个模型

6、早停止：如在训练中多次迭代后发现模型性能没有显著提高就停止训练

7、交叉验证

8、特征选择/特征降维

9、创建一个验证集是最基本的防止过拟合的方法。我们最终训练得到的模 型目标是要在验证集上面有好的表现，而不是训练集。

### 降低过拟合的办法如下：

正则化是针对过拟合而提出的，以为在求解模型最优的是一般优化最小的经验风险，现在在该经验风险上加入模型复杂度这一项（正则化项是模型参数向量的范数），并使用一个rate比率来权衡模型复杂度与以往经验风险的权重，如果模型复杂度越高，结构化的经验风险会越大，现在的目标就变为了结构经验风险的最优化，可以防止模型训练过度复杂，有效的降低过拟合的风险。

### L1正则化：

目标函数中增加所有权重w参数的绝对值之和, 逼迫更多w为零(也就是变稀疏. L2因为其导数也趋0, 奔向零的速度不如L1给力了). 大家对稀疏规则化趋之若鹜的一个关键原因在于它能实现特征的自动选择。一般来说，xi的大部分元素（也就是特征）都是和最终的输出yi没有关系或者不提供任何信息的，在最小化目标函数的时候考虑xi这些额外的特征，虽然可以获得更小的训练误差，但在预测新的样本时，这些没用的特征权重反而会被考虑，从而干扰了对正确yi的预测。稀疏规则化算子的引入就是为了完成特征自动选择的光荣使命，它会学习地去掉这些无用的特征，也就是把这些特征对应的权重置为0。

### L2正则化：

目标函数中增加所有权重w参数的平方之和, 逼迫所有w尽可能趋向零但不为零. 因为过拟合的时候, 拟合函数需要顾忌每一个点, 最终形成的

拟合函数波动很大, 在某些很小的区间里, 函数值的变化很剧烈, 也就是某些w非常大. 为此, L2正则化的加入就惩罚了权重变大的趋势.

### L1和L2的区别：

1、L1是Lasso Regression，表示向量中每个元素绝对值的和：L1范数的解通常是稀疏性的，倾向于选择数目较少的一些非常大的值或者数目较多的insignificant的小值。

2、L2是岭回归，Ridge Regression，是欧氏距离也就是平方和的平方根。L2范数越小，可以使得w的每个元素都很小，接近于0，但L1范数不同的是他不会让它等于0而是接近于0。

3、L1正则化的w可取的值是转置的方形，L2对应的是圆形。这样损失函数l（w）的最小值更容易在L1对应的边角上取得，从而这些维度变成0了。

从贝叶斯的角度来看，加入正则项相当于加入了一种先验。即当训练一个模型时，仅依靠前的训练数据集是不够的，为了实现更好的泛化能力，往往需要加入先验项。

L1范数相当于加入了一个Laplacean先验；

L2范数相当于加入了一个Gaussian先验。

L2对大数的惩罚更大，但是解相对来说比较均匀

### L1和L2正则先验分别服从什么分布

L1是拉普拉斯分布，L2是高斯分布

L2先验趋向零周围, L1先验趋向零本身

### 随机失活(dropout)

在训练的运行的时候，让神经元以超参数p的概率被激活(也就是1-p的概率被设置为0), 每个w因此随机参与, 使得任意w都不是不可或缺的, 效果类似于数量巨大的模型集成。

### 逐层归一化(batch normalization)

这个方法给每层的输出都做一次归一化(网络上相当于加了一个线性变换层), 使得下一层的输入接近高斯分布. 这个方法相当于下一层的w训练时避免了其输入以偏概全, 因而泛化效果非常好.

### 提前终止(early stopping)

理论上可能的局部极小值数量随参数的数量呈指数增长, 到达某个精确的最小值是不良泛化的一个来源. 实践表明, 追求细粒度极小值具有较高的泛化误差。这是直观的，因为我们通常会希望我们的误差函数是平滑的, 精确的最小值处所见相应误差曲面具有高度不规则性, 而我们的泛化要求减少精确度去获得平滑最小值, 所以很多训练方法都提出了提前终止策略. 典型的方法是根据交叉叉验证提前终止: 若每次训练前, 将训练数据划分为若干份, 取一份为测试集, 其他为训练集, 每次训练完立即拿此次选中的测试集自测. 因为每份都有一次机会当测试集, 所以此方法称之为交叉验证. 交叉验证的错误率最小时可以认为泛化性能最好, 这时候训练错误率虽然还在继续下降, 但也得终止继续训练了

## LR和SVM的联系与区别：

### 联系：

1、LR和SVM都可以处理分类问题，且一般都用于处理线性二分类问题（在改进的情况下可以处理多分类问题）

2、两个方法都可以增加不同的正则化项，如l1、l2等等。所以在很多实验中，两种算法的结果是很接近的

### 区别：

1、LR是参数模型，SVM是非参数模型。

2、从目标函数来看，区别在于逻辑回归采用的是logistical loss，SVM采用的是hinge loss，这两个损失函数的目的都是增加对分类影响较大的数据点的权重，减少与分类关系较小的数据点的权重。

3、SVM的处理方法是只考虑support vectors，也就是和分类最相关的少数点，去学习分类器。而逻辑回归通过非线性映射，大大减小了离分类平面较远的点的权重，相对提升了与分类最相关的数据点的权重。

4、逻辑回归相对来说模型更简单，好理解，特别是大规模线性分类时比较方便。而SVM的理解和优化相对来说复杂一些，SVM转化为对偶问题后,分类只需要计算与少数几个支持向量的距离,这个在进行复杂核函数计算时优势很明显,能够大大简化模型和计算。

5、logic 能做的 svm能做，但可能在准确率上有问题，svm能做的logic有的做不了。

## LR与线性回归的区别与联系：

1: LR工业上一般指Logistic Regression(逻辑回归)而不是Linear Regression(线性回归). LR在线性回归的实数范围输出值上施加sigmoid函数将值收敛到0~1范围, 其目标函数也因此从差平方和函数变为对数损失函数, 以提供最优化所需导数（sigmoid函数是softmax函数的二元特例, 其导数均为函数值的f\*(1-f)形式）。请注意, LR往往是解决二元0/1分类问题的, 只是它和线性回归耦合太紧, 不自觉也冠了个回归的名字(马甲无处不在). 若要求多元分类,就要把sigmoid换成大名鼎鼎的softmax了。

2：个人感觉逻辑回归和线性回归首先都是广义的线性回归，其次经典线性模型的优化目标函数是最小二乘，而逻辑回归则是似然函数

另外线性回归在整个实数域范围内进行预测，敏感度一致，而分类范围，需要在[0,1]。逻辑回归就是一种减小预测范围，将预测值限定为[0,1]间的一种回归模型，因而对于这类问题来说，逻辑回归的鲁棒性比线性回归的要好。

3：逻辑回归的模型本质上是一个线性回归模型，逻辑回归都是以线性回归为理论支持的。但线性回归模型无法做到sigmoid的非线性形式，sigmoid可以轻松处理0/1分类问题。

## GBDT和Xgboost的区别：

集成学习主要可以分为三大类，Boosting, Bagging, Stacking。Boosting的代表有AdaBoost, gbdt, xgboost。而Bagging的代表则是随机森林 (Random Forest)。Stacking 的话，好像还没有著名的代表，可以视其为一种集成的套路。Bagging方法有放回地采样同数量样本训练每个学习器, 然后再一起集成(简单投票); Boosting方法使用全部样本(可调权重)依次训练每个学习器, 迭代集成(平滑加权).

决策树属于最常用的学习器, 其学习过程是从根建立树, 也就是如何决策叶子节点分裂. ID3/C4.5决策树用信息熵计算最优分裂, CART决策树用基尼指数计算最优分裂, xgboost决策树使用二阶泰勒展开系数计算最优分裂.

下面所提到的学习器都是决策树:

### Bagging方法:

    学习器间不存在强依赖关系, 学习器可并行训练生成, 集成方式一般为投票;

    Random Forest属于Bagging的代表, 放回抽样, 每个学习器随机选择部分特征去优化;

随机森林Random Forest是一个包含多个决策树的分类器。至于AdaBoost，则是英文"Adaptive Boosting"（自适应增强）的缩写。GBDT（Gradient Boosting Decision Tree），即梯度上升决策树算法，相当于融合决策树和梯度上升boosting算法。

### Boosting方法:

    学习器之间存在强依赖关系、必须串行生成, 集成方式为加权和;

    Adaboost属于Boosting, 采用指数损失函数替代原本分类任务的0/1损失函数;

    GBDT属于Boosting的优秀代表, 对函数残差近似值进行梯度下降, 用CART回归树做学习器, 集成为回归模型;

    xgboost属于Boosting的集大成者, 对函数残差近似值进行梯度下降, 迭代时利用了二阶梯度信息, 集成模型可分类也可回归. 由于它可在特征粒度上并行计算, 结构风险和工程实现都做了很多优化, 泛化, 性能和扩展性都比GBDT要好。

### Xgboost的优点：

1.损失函数是用泰勒展式二项逼近，而不是像gbdt里的就是一阶导数

2.对树的结构进行正则化约束,防止模型过度复杂,降低了过拟合的可能性

3.节点分裂的方式不同，gbdt是用的gini系数，xgboost是经过优化推导后的

## 分类与聚类：

Classification (分类)，对于一个 classifier ，通常需要你告诉它“这个东西被分为某某类”这样一些例子，理想情况下，一个 classifier 会从它得到的训练集中进行“学习”，从而具备对未知数据进行分类的能力，这种提供训练数据的过程通常叫做 supervised learning (监督学习)，

而Clustering(聚类)，简单地说就是把相似的东西分到一组，聚类的时候，我们并不关心某一类是什么，我们需要实现的目标只是把相似的东西聚到一起，因此，一个聚类算法通常只需要知道如何计算相似 度就可以开始工作了，因此 clustering 通常并不需要使用训练数据进行学习，这在 Machine Learning 中被称作 unsupervised learning (无监督学习)

### 常见的分类与聚类算法：

所谓分类，就是根据文本的特征或属性，划分到已有的类别中

常用的分类算法包括：

决策树分类法，朴素的贝叶斯分类算法(native Bayesian classifier)、基于支持向量机(SVM)的分类器，神经网络法，k-最近邻法(k-nearest neighbor，kNN)，模糊分类法,随机森林，逻辑回归，KNN等等。

分类作为一种监督学习方法，要求必须事先明确知道各个类别的信息，并且断言所有待分类项都有一个类别与之对应。但是很多时候上述条件得不到满足，尤其是在处理海量数据的时候，如果通过预处理使得数据满足分类算法的要求，则代价非常大，这时候可以考虑使用聚类算法。

而K均值(K-means clustering)聚类则是最典型的聚类算法(当然，除此之外，还有很多诸如属于划分法K-MEDOIDS算法、CLARANS算法；属于层次法的BIRCH算法、CURE算法、CHAMELEON算法等；基于密度的方法：DBSCAN算法、OPTICS算法、DENCLUE算法等；基于网格的方法：STING算法、CLIQUE算法、WAVE-CLUSTER算法；基于模型的方法，本系列后续会介绍其中几种)

## Xgboost：

### 为什么xgboost要用泰勒展开，优势在哪里

xgboost使用了一阶和二阶偏导, 二阶导数有利于梯度下降的更快更准. 使用泰勒展开取得函数做自变量的二阶导数形式,可以在不选定损失函数具体形式的情况下,仅仅依靠输入数据的值就可以进行叶子分裂优化计算,本质上也就把损失函数的选取和模型算法优化/参数选择分开了.这种去耦合增加了xgboost的适用性,使得它按需选取损失函数, 可以用于分类,也可以用于回归

### xgboost如何寻找最优特征？是又放回还是无放回的呢？

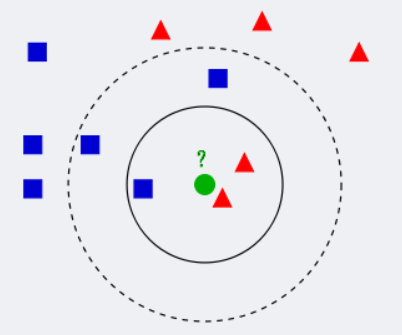
xgboost在训练的过程中给出各个特征的增益评分，最大增益的特征会被选出来作为分裂依据, 从而记忆了每个特征对在模型训练时的重要性 -- 从根到叶子中间节点涉及某特征的次数作为该特征重要性排序.

xgboost属于boosting集成学习方法, 样本是不放回的, 因而每轮计算样本不重复. 另一方面, xgboost支持子采样, 也就是每轮计算可以不使用全部样本, 以减少过拟合. 进一步地, xgboost 还有列采样, 每轮计算按百分比随机采样一部分特征, 既提高计算速度又减少过拟合

## K近邻算法（KNN）

K近邻算法，即K-Nearest Neighbor algorithm，简称KNN算法，可以简单的认为是：K个最近的邻居，当K=1时，算法便成了最近邻算法，即寻找最近的那个邻居。

  所谓K近邻算法，即是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（也就是上面所说的K个邻居），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。根据这个说法，咱们来看下引自维基百科上的一幅图：



如上图所示，有两类不同的样本数据，分别用蓝色的小正方形和红色的小三角形表示，而图正中间的那个绿色的圆所标示的数据则是待分类的数据。也就是说，现在，我们不知道中间那个绿色的数据是从属于哪一类（蓝色小正方形or红色小三角形），下面，我们就要解决这个问题：给这个绿色的圆分类。

如果K=3，绿色圆点的最近的3个邻居是2个红色小三角形和1个蓝色小正方形，少数从属于多数，基于统计的方法，判定绿色的这个待分类点属于红色的三角形一类。

如果K=5，绿色圆点的最近的5个邻居是2个红色三角形和3个蓝色的正方形，还是少数从属于多数，基于统计的方法，判定绿色的这个待分类点属于蓝色的正方形一类。

  于此我们看到，当无法判定当前待分类点是从属于已知分类中的哪一类时，我们可以依据统计学的理论看它所处的位置特征，衡量它周围邻居的权重，而把它归为(或分配)到权重更大的那一类。这就是K近邻算法的核心思想

### K如何选取：

KNN中的K值选取对K近邻算法的结果会产生重大影响。

如果选择较小的K值，就相当于用较小的领域中的训练实例进行预测，“学习”近似误差会减小，只有与输入实例较近或相似的训练实例才会对预测结果起作用，与此同时带来的问题是“学习”的估计误差会增大，换句话说，K值的减小就意味着整体模型变得复杂，容易发生过拟合；

如果选择较大的K值，就相当于用较大领域中的训练实例进行预测，其优点是可以减少学习的估计误差，但缺点是学习的近似误差会增大。这时候，与输入实例较远（不相似的）训练实例也会对预测器作用，使预测发生错误，且K值的增大就意味着整体的模型变得简单。K=N，则完全不足取，因为此时无论输入实例是什么，都只是简单的预测它属于在训练实例中最多的累，模型过于简单，忽略了训练实例中大量有用信息。

在实际应用中，K值一般取一个比较小的数值，例如采用交叉验证法（简单来说，就是一部分样本做训练集，一部分做测试集）来选择最优的K值。

### KNN算法的优缺点

优点:

1. 思想简单，易于理解，易于实现
2. 无需估计参数，无需训练；
3. 精度高，对异常值不敏感，
4. 特别适合于多分类问题。

缺点:

1. 计算复杂度高，空间复杂度高。
2. 可解释性差。无法给出决策树那样的规则。
3. 当样本分布不平衡时，新的样本会归类为主导样本，从而不能更好的接近实际分类结果。

### 优化问题

压缩训练样本；

确定最终的类别时，不是简单的采用投票法，而是进行加权投票，距离越近权重越高。

### KNN算法实现步骤:

1 计算距离

2选择距离最小的K个点

3 排序

### Means算法的流程

1. 初始化常数K,随机选取初始点为质心
2. 重复计算样本与每个质心之间的相似度，将样本归类到最相似的簇中
3. 将各个簇中的数据求平均值，作为新的质心
4. 重复2,3过程，直到质心不在改变
5. 输出最终的质心以及每个类

使用K均值算法对测试数据进行聚类，一般情况下效果不错。但有时候效果就不好了，因为K均值聚类收敛的是局部最小值，而不是全局最小值。

### K-Means优缺点:

优点:

计算复杂度

思想很简单，实际上是一个优化全局MSE(Mean Squre Error)和局部MSE的过程。

缺点:

分类结果依赖于分类中心的初始化(可以通过多次K-means取最优来解决)。

对类别规模差异太明显的数据效果不好。

同理，K-means对于距离非常近的类别(blobs)的分类效果也不好。

可参考：https://blog.csdn.net/v\_july\_v/article/details/8203674

## 归一化：

### 为何要经常对数据做归一化：

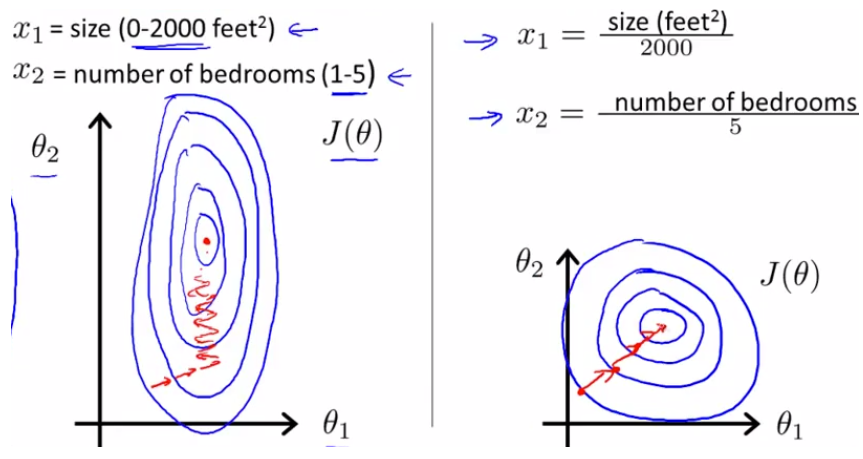
1. 归一化后加快了梯度下降求最优解的速度
2. 归一化有可能提高精度
3. 能不归一化最好不归一化，之所以进行数据归一化是因为各维度的量纲不相同。而且需要看情况进行归一化
4. 有些模型在各维度进行了不均匀的伸缩后，最优解与原来不等价（如SVM）需要归一化
5. 有些模型伸缩有与原来等价，如：LR则不用归一化，但是实际中往往通过迭代求解模型参数，如果目标函数太扁（想象一下很扁的高斯模型）迭代算法会发生不收敛的情况，所以最坏进行数据归一化
6. 其实本质是由于loss函数不同造成的,SVM用了欧拉距离,如果一个特征很大就会把其他的维度dominated。而LR可以通过权重调整使得损失函数不变

### 归一化为什么能提高梯度下降法求解最优解的速度

如下图所示，蓝色的圈圈图代表的是两个特征的等高线。其中左图两个特征X1和X2的区间相差非常大，X1区间是[0,2000]，X2区间是[1,5]，其所形成的等高线非常尖。当使用梯度下降法寻求最优解时，很有可能走“之字型”路线（垂直等高线走），从而导致需要迭代很多次才能收敛；

而右图对两个原始特征进行了归一化，其对应的等高线显得很圆，在梯度下降进行求解时能较快的收敛。

因此如果机器学习模型使用梯度下降法求最优解时，归一化往往非常有必要，否则很难收敛甚至不能收敛。



### 归一化有可能提高精度：

一些分类器需要计算样本之间的距离（如欧氏距离），例如KNN。如果一个特征值域范围非常大，那么距离计算就主要取决于这个特征，从而与实际情况相悖（比如这时实际情况是值域范围小的特征更重要）

### 归一化的类型

1. 线性归一化

这种归一化方法比较适用在数值比较集中的情况。这种方法有个缺陷，如果max和min不稳定，很容易使得归一化结果不稳定，使得后续使用效果也不稳定。实际使用中可以用经验常量值来替代max和min。

2）标准差标准化

经过处理的数据符合标准正态分布，即均值为0，标准差为1，其转化函数为：

　　 其中μ为所有样本数据的均值，σ为所有样本数据的标准差。

3）非线性归一化

经常用在数据分化比较大的场景，有些数值很大，有些很小。通过一些数学函数，将原始值进行映射。该方法包括 log、指数，正切等。需要根据数据分布的情况，决定非线性函数的曲线，比如log(V, 2)还是log(V, 10)等

### 哪些机器学习算法不需要做归一化处理：

概率模型（树形模型）不需要归一化，因为它们不关心变量的值，而是关心变量的分布和变量之间的条件概率，如决策树、rf。而像adaboost、svm、lr、KNN、KMeans之类的最优化问题就需要归一化。

归一化和标准化主要是为了使计算更方便比如两个变量的量纲不同可能一个的数值远大于另一个那么他们同时作为变量的时候可能会造成数值计算的问题，比如说求矩阵的逆可能很不精确或者梯度下降法的收敛比较困难，还有如果需要计算欧式距离的话可能量纲也需要调整所以我估计lr 和 knn 保准话一下应该有好处。至于其他的算法我也觉得如果变量量纲差距很大的话先标准化一下会有好处。

### 树形结构为什么不需要归一化：

数值缩放，不影响分裂点位置。因为第一步都是按照特征值进行排序的，排序的顺序不变，那么所属的分支以及分裂点就不会有不同。对于线性模型，比如说LR，我有两个特征，一个是(0,1)的，一个是(0,10000)的，这样运用梯度下降时候，损失等高线是一个椭圆的形状，这样我想迭代到最优点，就需要很多次迭代，但是如果进行了归一化，那么等高线就是圆形的，那么SGD就会往原点迭代，需要的迭代次数较少。

另外，树模型是不能进行梯度下降的，因为树模型是阶跃的，阶跃点是不可导的，并且求导没意义，所以树模型（回归树）寻找最优点事通过寻找最优分裂点完成的。

## 标准化与归一化的区别：

### 归一化方法：

1、把数变为（0，1）之间的小数主要是为了数据处理方便提出来的，把数据映射到0～1范围之内处理，更加便捷快速。

2、把有量纲表达式变为无量纲表达式 归一化是一种简化计算的方式，即将有量纲的表达式，经过变换，化为无量纲的表达式，成为纯量。

### 特点：

对不同特征维度的伸缩变换的目的是使各个特征维度对目标函数的影响权重是一致的，即使得那些扁平分布的数据伸缩变换成类圆形。这也就改变了原始数据的一个分布。

### 好处：

1 提高迭代求解的收敛速度

2 提高迭代求解的精度

### 标准化方法：

数据的标准化是将数据按比例缩放，使之落入一个小的特定区间。由于信用指标体系的各个指标度量单位是不同的，为了能够将指标参与评价计算，需要对指标进行规范化处理，通过函数变换将其数值映射到某个数值区间。

### 特点：

对不同特征维度的伸缩变换的目的是使得不同度量之间的特征具有可比性。同时不改变原始数据的分布

### 好处

1、使得不同度量之间的特征具有可比性，对目标函数的影响体现在几何分布上，而不是数值上

2、不改变原始数据的分布

## 简要说一下完整机器学习项目的流程：

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，指的我们明确我们可以获得什么样的数据，目标是一个分类还是回归或者是聚类的问题，如果都不是的话，如果划归为其中的某类问题。

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个数量级的差距。

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效力。

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，全面分析误差产生误差的原因: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。

## 熵：

用来表示随机变量的不确定性

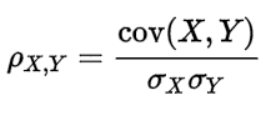
1948年，香农Claude E. Shannon引入信息（熵），将其定义为离散随机事件的出现概率。一个系统越是有序，信息熵就越低；反之，一个系统越是混乱，信息熵就越高。所以说，信息熵可以被认为是系统有序化程度的一个度量

## 协方差和相关性的区别：

相关性是协方差的标准化格式。协方差本身很难做比较。例如：如果我们计算工资（$）和年龄（岁）的协方差，因为这两个变量有不同的度量，所以我们会得到不能做比较的不同的协方差。



为了解决这个问题，我们计算相关性来得到一个介于-1和1之间的值，就可以忽略它们各自不同的度量



## 线性分类器和非线性分类器的区别及优劣：

线性和非线性是针对，模型参数和输入特征来讲的；比如输入x，模型y=ax+ax^2那么就是非线性模型，如果输入是x和X^2则模型是线性的。

线性分类器可解释性好，计算复杂度较低，不足之处是模型的拟合效果相对弱些。

非线性分类器效果拟合能力较强，不足之处是数据量不足容易过拟合、计算复杂度高、可解释性不好。

常见的线性分类器有：LR,贝叶斯分类，单层感知机、线性回归

常见的非线性分类器：决策树、RF、GBDT、多层感知机

SVM两种都有（看线性核还是高斯核）

## 激活函数：

用的非线性激活函数有sigmoid、tanh、relu等等，前两者sigmoid/tanh比较常见于全连接层，后者relu常见于卷积层。这里先简要介绍下最基础的sigmoid函数

sigmoid的函数表达式如下：



其中z是一个线性组合，比如z可以等于：b + w1\*x1 + w2\*x2。通过代入很大的正数或很小的负数到g(z)函数中可知，其结果趋近于0或1

也就是说，sigmoid函数的功能是相当于把一个实数压缩至0到1之间。当z是非常大的正数时，g(z)会趋近于1，而z是非常小的负数时，则g(z)会趋近于0。

### 对比下Sigmoid、Tanh、ReLu这三个激活函数

sigmoid函数又称logistic函数，应用在Logistic回归中。logistic回归的目的是从特征学习出一个0/1分类模型，而这个模型是将特性的线性组合作为自变量，由于自变量的取值范围是负无穷到正无穷。因此，使用logistic函数将自变量映射到(0,1)上，映射后的值被认为是属于y=1的概率因而，sigmoid函数是Logstic回归模型的激活函数

#### sigmoid函数有如下几个缺点：

正向计算包含指数，反向传播的导数也包含指数计算和除法运算，因而计算复杂度很高。

输出的均值非0。这样使得网络容易发生梯度消失或梯度爆炸。这也是batch normalization要解决的问题。

假如sigmoid函数为f(x)，那么f'(x)=f(x)(1-f(x))，因为f(x)输出在0-1之间，那么f'(x)恒大于0。 这就导致全部的梯度的正负号都取决于损失函数上的梯度。这样容易导致训练不稳定，参数一荣俱荣一损俱损。

同样的，f'(x)=f(x)(1-f(x))，因为f(x)输出在0-1之间，那么f'(x)输出也在0-1之间，当层次比较深时，底层的导数就是很多在0-1之间的数相乘，从而导致了梯度消失问题。

#### tanh

同sigmoid类似，但是输出值在-1到1之间，均值为0，是其相对于sigmoid的提升。但是因为输出在-1，1之间，因而输出不能被看做是概率

#### ReLU

相对于sigmoid和tanh来说，有如下优点：

计算量下，没有指数和除法运算。

不会饱和，因为在x>0的情况下，导数恒等于1

收敛速度快，在实践中可以得知，它的收敛速度是sigmoid的6倍。

Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生

但是Relu也有缺点，缺点在于，

如果有一个特别大的导数经过神经单元使得输入变得小于0，这样会使得这个单元永远得不到参数更新，因为输入小于0时导数也是0. 这就形成了很多dead cell。

## 卷积：

对图像（不同的数据窗口数据）和滤波矩阵（一组固定的权重：因为每个神经元的多个权重固定，所以又可以看做一个恒定的滤波器filter）做**内积**（逐个元素相乘再求和）的操作就是所谓的『卷积』操作，也是卷积神经网络的名字来源。

## 梯度下降、梯度消失、梯度爆炸：

### 梯度下降流程：

1）首先对θ赋值，这个值可以是随机的，也可以让θ是一个全零的向量。

    2）改变θ的值，使得J(θ)按梯度下降的方向进行减少。

### 梯度下降法找到的一定是下降最快的方向么？ 梯度下降法并不是下降最快的方向，它只是目标函数在当前的点的切平面（当然高维问题不能叫平面）上下降最快的方向。在practical implementation中，牛顿方向（考虑海森矩阵）才一般被认为是下降最快的方向

梯度下降不一定能够找到全局的最优解，也有可能只是一个局部最优解。当然，如果损失函数是凸函数，梯度下降法得到的解就一定是全局最优解

梯度下降法的优化思想是用当前位置负梯度方向作为搜索方向，因为该方向为当前位置的最快下降方向，所以也被称为是“最速下降法”。最速下降法越接近目标值，步长越小，前进越慢

正因为梯度度下降法在接近最优解的区域收敛速度明显变慢，所以利用梯度下降法求解需要很多次的迭代。在机器学习中，基于基本的梯度下降法发展了两种梯度下降方法，分别为随机梯度下降法和批量梯度下降法

### 随机梯度下降：

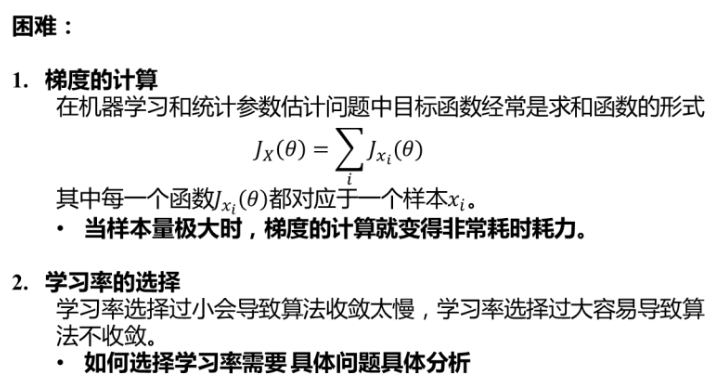
普通的梯度下降算法在更新回归系数时要遍历整个数据集，是一种批处理方法，这样训练数据特别忙庞大时，可能出现如下问题：

1）收敛过程可能非常慢；

2）如果误差曲面上有多个局极小值，那么不能保证这个过程会找到全局最小值。

为了解决上面的问题，实际中我们应用的是梯度下降的一种变体被称为随机梯度下降。随机梯度下降的思想是根据每个单独的训练样本来更新权值

### 随机梯度下降法的问题和挑战：



### 梯度消失：

两种情况下梯度消失经常出现，一是在深层网络中，二是采用了不合适的损失函数，比如sigmoid。因为通常神经网络所用的激活函数是sigmoid函数，这个函数有个特点，就是能将负无穷到正无穷的数映射到0和1之间，并且对这个函数求导的结果是f ′(x)=f(x)(1−f(x))。因此两个0到1之间的数相乘，得到的结果就会变得很小了。神经网络的反向传播是逐层对函数偏导相乘，因此当神经网络层数非常深的时候，最后一层产生的偏差就因为乘了很多的小于1的数而越来越小，最终就会变为0，从而导致层数比较浅的权重没有更新，这就是梯度消失。

### 梯度爆炸：

梯度爆炸就是由于初始化权值过大，前面层会比后面层变化的更快，就会导致权值越来越大，梯度爆炸的现象就发生了。

其实梯度爆炸和梯度消失问题都是因为网络太深，网络权值更新不稳定造成的，本质上是因为梯度反向传播中的连乘效应

### sigmoid 时，消失和爆炸哪个更易发生：

sigmoid 导数的最大值为0.25，通常 abs(w)<1，则激活函数的导数与权重的积小于0.25，前面的层比后面的层梯度变化更小，故变化更慢，从而引起了梯度消失问题。当权值过大，前面层比后面层梯度变化更快，则引起梯度爆炸问题。因为 sigmoid 导数最大为1/4，故只有当 abs(w)>4 时才可能出现梯度爆炸，而最普遍发生的是梯度消失问题

### 如何解决梯度消失和梯度爆炸：

1、对于RNN，可以通过梯度截断，避免梯度爆炸

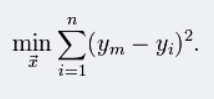
　 2、可以通过添加正则项，避免梯度爆炸

　 3、使用LSTM等自循环和门控制机制，避免梯度消失

4、优化激活函数，使用 ReLU、LReLU、ELU、maxout 等激活函数   
 sigmoid函数的梯度随着x的增大或减小和消失，而ReLU不会。

## 最小二乘法：

最小二乘法（又称最小平方法）是一种数学优化技术。它通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和为最小。用函数表示为：



 使误差「所谓误差，当然是观察值与实际真实值的差量」平方和达到最小以寻求估计值的方法，就叫做最小二乘法，用最小二乘法得到的估计，叫做最小二乘估计。当然，取平方和作为目标函数只是众多可取的方法之一。

### 最小二乘法的优良性：

最小二乘使得误差平方和最小，并在各个方程的误差之间建立了一种平衡，从而防止某一个极端误差取得支配地位

计算中只要求偏导后求解线性方程组，计算过程明确便捷

最小二乘可以导出算术平均值作为估计值

### 最小二乘法的原理之一：

当估计误差服从正态分布时，最小二乘法等同于极大似然估计。 如果 y = f(x) + e, 其中y 是目标值，f(x)为估计值，e为误差项。而由于中心极限定理的原因，很多误差分布确实服从正态分布，这也是最小二乘法能够十分有效的一个原因。

## Python：

### Python到底是什么语言：

1、Python是解释型语言。这意味着不像C和其他语言，Python运行前不需要编译。其他解释型语言包括PHP和Ruby

2、Python是动态类型的，这意味着你不需要在声明变量时指定类型。你可以先定义x=111，然后 x=”I’m a string”。

3、Python是面向对象语言，所有允许定义类并且可以继承和组合。Python没有访问访问标识如在C++中的public, private, 这就非常信任程序员的素质，相信每个程序员都是“成人”了~

4、在Python中，函数是一等公民。这就意味着它们可以被赋值，从其他函数返回值，并且传递函数对象。类不是一等公民。

5、写Python代码很快，但是跑起来会比编译型语言慢。幸运的是，Python允许使用C扩展写程序，所以瓶颈可以得到处理。Numpy库就是一个很好例子，因为很多代码不是Python直接写的，所以运行很快。

6、Python使用场景很多 – web应用开发、大数据应用、数据科学、人工智能等等。它也经常被看做“胶水”语言，使得不同语言间可以衔接上。

7、Python能够简化工作  ，使得程序员能够关心如何重写代码而不是详细看一遍底层实现。

### Python是如何进行内存管理的：

从三个方面来说,一对象的引用计数机制,二垃圾回收机制,三内存池机制

一、对象的引用计数机制

Python内部使用引用计数，来保持追踪内存中的对象，所有对象都有引用计数。

引用计数增加的情况：

1，一个对象分配一个新名称

2，将其放入一个容器中（如列表、元组或字典）

引用计数减少的情况：

1，使用del语句对对象别名显示的销毁

2，引用超出作用域或被重新赋值

sys.getrefcount( )函数可以获得对象的当前引用计数

多数情况下，引用计数比你猜测得要大得多。对于不可变数据（如数字和字符串），解释器会在程序的不同部分共享内存，以便节约内存。

二、垃圾回收

1，当一个对象的引用计数归零时，它将被垃圾收集机制处理掉。

2，当两个对象a和b相互引用时，del语句可以减少a和b的引用计数，并销毁用于引用底层对象的名称。然而由于每个对象都包含一个对其他对象的应用，因此引用计数不会归零，对象也不会销毁。（从而导致存储器泄露）。为解决这一问题，解释器会定期执行一个循环检测器，搜索不可访问对象的循环并删除它们。

三、存储器池机制

Python提供了对内存的垃圾收集机制，但是它将不用的内存放到内存池而不是返回给操作系统。

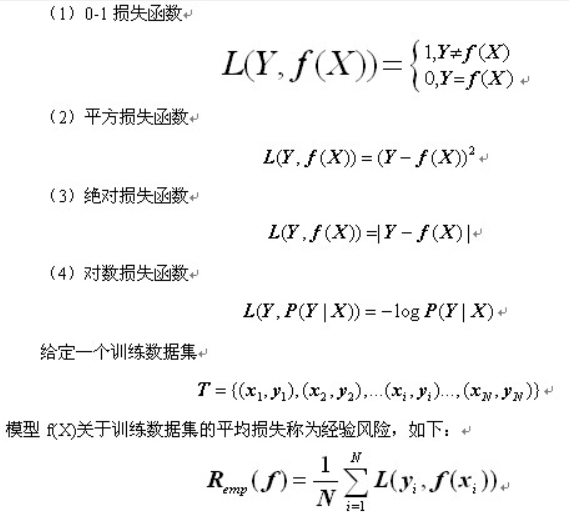
1，Pymalloc机制。为了加速Python的执行效率，Python引入了一个内存池机制，用于管理对小块内存的申请和释放。

2，Python中所有小于256个字节的对象都使用pymalloc实现的分配器，而大的对象则使用系统的malloc。

3，对于Python对象，如整数，浮点数和List，都有其独立的私有内存池，对象间不共享他们的内存池。也就是说如果你分配又释放了大量的整数，用于缓存这些整数的内存就不能再分配给浮点数。

## 损失函数：

对于给定的输入X，由f(X)给出相应的输出Y，这个输出的预测值f(X)与真实值Y可能一致也可能不一致（要知道，有时损失或误差是不可避免的），用一个损失函数来度量预测错误的程度。损失函数记为L(Y, f(X))。常用的损失函数有以下几种：



## 深度学习：

### 如何提升深度学习的性能：

1. 通过数据提升性能   
 2. 通过算法提升性能   
 3. 通过算法调参提升性能   
 4. 通过嵌套模型提升性能

### 什麽样的资料集不适合用深度学习：

数据集太小，数据样本不足时，深度学习相对其它机器学习算法，没有明显优势。

数据集没有局部相关特性，目前深度学习表现比较好的领域主要是图像／语音／自然语言处理等领域，这些领域的一个共性是局部相关性。图像中像素组成物体，语音信号中音位组合成单词，文本数据中单词组合成句子，这些特征元素的组合一旦被打乱，表示的含义同时也被改变。对于没有这样的局部相关性的数据集，不适于使用深度学习算法进行处理。举个例子：预测一个人的健康状况，相关的参数会有年龄、职业、收入、家庭状况等各种元素，将这些元素打乱，并不会影响相关的结果。

## 随机森林：

### 随机森林如何处理缺失值：

1、（na.roughfix）简单粗暴，对于训练集,同一个class下的数据，如果是分类变量缺失，用众数补上，如果是连续型变量缺失，用中位数补。

2、（rfImpute）这个方法计算量大，至于比方法一好坏？不好判断。先用na.roughfix补上缺失值，然后构建森林并计算proximity matrix，再回头看缺失值，如果是分类变量，则用没有缺失的观测实例的proximity中的权重进行投票。如果是连续型变量，则用proximity矩阵进行加权平均的方法补缺失值。然后迭代4-6次

### 随机森林如何评估特征重要性？

衡量变量重要性的方法有两种，Decrease GINI 和 Decrease Accuracy：

1、 Decrease GINI： 对于回归问题，直接使用argmax(VarVarLeftVarRight)作为评判标准，即当前节点训练集的方差Var减去左节点的方差VarLeft和右节点的方差VarRight。

2、 Decrease Accuracy：对于一棵树Tb(x)，我们用OOB样本可以得到测试误差1；然后随机改变OOB样本的第j列：保持其他列不变，对第j列进行随机的上下置换，得到误差2。至此，我们可以用误差1-误差2来刻画变量j的重要性。基本思想就是，如果一个变量j足够重要，那么改变它会极大的增加测试误差；反之，如果改变它测试误差没有增大，则说明该变量不是那么的重要。

### 随机森林优缺点:

优点：

1.具有极高的准确率

2.随机性的引入，使得随机森林不容易过拟合，而且使得随机森林有很好的抗噪声能力

3.能处理很高维度的数据，并且不用做特征选择

4.既能处理离散型数据，也能处理连续性数据，数据集无需规范化

5.训练速度快，可以得到变量的重要性排序

6.容易实现并行化

缺点:

当随机森林中的决策树个数很多时，训练时需要的空间和时间会较大

随机森林的构建过程大致如下:

将生成的每棵决策树组成随机森林。对于分类问题，按多棵树分类器投票决定最终分类结果；对于回归问题，由多棵树预测值的均值决定最终预测结果。

## Kmeans：

### 优化Kmeans

使用kd树或者ball tree

将所有的观测实例构建成一颗kd树，之前每个聚类中心都是需要和每个观测点做依次距离计算，现在这些聚类中心根据kd树只需要计算附近的一个局部区域即可

### KMeans初始类簇中心点的选取：

k-means++算法选择初始seeds的基本思想就是：初始的聚类中心之间的相互距离要尽可能的远。

1. 从输入的数据点集合中随机选择一个点作为第一个聚类中心

2. 对于数据集中的每一个点x，计算它与最近聚类中心(指已选择的聚类中心)的距离D(x)

3. 选择一个新的数据点作为新的聚类中心，选择的原则是：D(x)较大的点，被选取作为聚类中心的概率较大

4. 重复2和3直到k个聚类中心被选出来

5. 利用这k个初始的聚类中心来运行标准的k-means算法

## 特征选择：

特征选择是一个重要的数据预处理过程，主要有两个原因：一是减少特征数量、降维，使模型泛化能力更强，减少过拟合;二是增强对特征和特征值之间的理解

### 常见的特征选择方式：

1. 去除方差较小的特征

2. 正则化。1正则化能够生成稀疏的模型。L2正则化的表现更加稳定，由于有用的特征往往对应系数非零。

3. 随机森林，对于分类问题，通常采用基尼不纯度或者信息增益，对于回归问题，通常采用的是方差或者最小二乘拟合。一般不需要feature engineering、调参等繁琐的步骤。它的两个主要问题，1是重要的特征有可能得分很低（关联特征问题），2是这种方法对特征变量类别多的特征越有利（偏向问题）。

4. 稳定性选择。是一种基于二次抽样和选择算法相结合较新的方法，选择算法可以是回归、SVM或其他类似的方法。它的主要思想是在不同的数据子集和特征子集上运行特征选择算法，不断的重复，最终汇总特征选择结果，比如可以统计某个特征被认为是重要特征的频率（被选为重要特征的次数除以它所在的子集被测试的次数）。理想情况下，重要特征的得分会接近100%。稍微弱一点的特征得分会是非0的数，而最无用的特征得分将会接近于0

## 数据预处理：

1. 缺失值，填充缺失值fillna：

i. 离散：None,

ii. 连续：均值。

iii. 缺失值太多，则直接去除该列

2. 连续值：离散化。有的模型（如决策树）需要离散值

3. 对定量特征二值化。核心在于设定一个阈值，大于阈值的赋值为1，小于等于阈值的赋值为0。如图像操作

4. 皮尔逊相关系数，去除高度相关的列

## 数据不平衡：

这主要是由于数据分布不平衡造成的。解决方法如下：

1、采样，对小样本加噪声采样，对大样本进行下采样

2、数据生成，利用已知样本生成新的样本

3、进行特殊的加权，如在Adaboost中或者SVM中

4、采用对不平衡数据集不敏感的算法

5、改变评价标准：用AUC/ROC来进行评价

6、采用Bagging/Boosting/ensemble等方法

7、在设计模型的时候考虑数据的先验分布

注： 将附件：1601W面试准备的题对于内容排版放入此章节，同时整合下

周考题：

https://me.csdn.net/ldq\_learning 以及附件评估指标的内容

负责人：牛老师

# 二、 数据结构面试题

[1.什么是数据](#_Toc494266848)?

答：所有能被输入到计算机中，且能被计算机处理的符号的集合。是计算机操作的对象的总称。

2.什么是数据类型？

答：在一种程序语言中，变量所具有的数据种类。整型、浮点型、字符型等。

3.什么是逻辑结构？

答：数据之间的相互关系。

①集合：结构中的数据元素除了同属于一种类型外，别无其它关系。

②线性结构：数据元素之间一对一的关系

③树形结构：数据元素之间一对多的关系

④图状结构或网状结构：结构中的数据元素之间存在多对多的关系

[4.什么是数据结构](#_Toc494266848)?

答： 数据结构是计算机存储、组织数据的方式。数据结构是指相互之间存在一种或多种特定关系的数据元素的集合。通常情况下，精心选择的数据结构可以带来更高的运行或者存储效率。数据结构往往同高效的检索算法和索引技术有关。

[5.常用数据结构有哪些？](#_Toc494266849)

答：①数组②栈③队列④链表⑤树⑥图⑦堆⑧散列表

6.什么是算法五个特性？

答：有穷性、确定性、可行性、输入、输出。

7.什么是算法的设计要求？

答：正确性、可读性、健壮性、高效率与低存储量需求。

8.什么是时间复杂度？

答：算法的执行时间与原操作执行次数之和成正比。时间复杂度有小到大：O(1)、O(logn)、O(n)、O(nlogn)、O()、O()。幂次时间复杂度有小到大O()、O(n!)、O()

9.什么是空间复杂度？

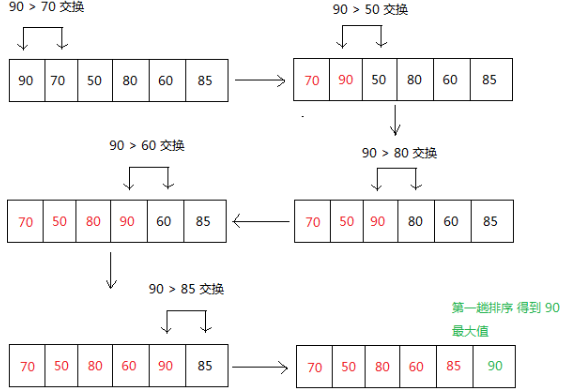
答：若输入数据所占空间只取决于问题本身，和算法无关，则只需要分析除输入和程序之外的辅助变量所占额外空间。

[10.常见的排序算法有哪些？](#_Toc494266850)

答：冒泡排序、选择排序、插入排序、快速排序、希尔排序、二叉堆排序等

[11.](#_Toc494266851)冒泡排序算法原理？

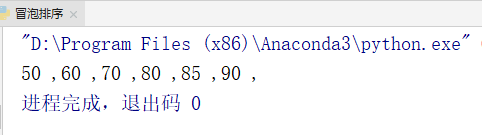
答：n个元素，前后两个元素比较大小，如果前面元素大于后面元素，则交换位置，直到排序完成。第一次排序见下图，以此类推得到从小到大序列



代码如下：

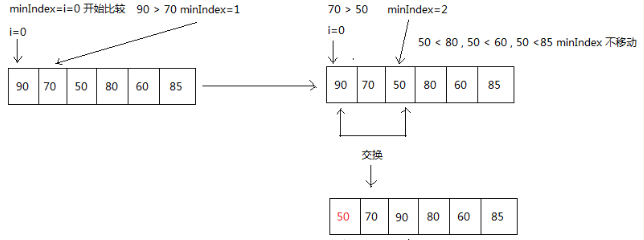
# -\*- coding: utf-8 -\*-  
def sort(arrays):  
 # 第一个循环排序次数  
 length = len(arrays)  
 for i in range(0,length-1):  
 # 第二个循环控制每次要比较的元素册数  
 for j in range(0,length-1):  
 if arrays[j] > arrays[j+1]:  
 flag = arrays[j]  
 arrays[j] = arrays[j+1]  
 arrays[j+1] = flag  
# 学生分数成绩  
scores = [90,70,50,80,60,85]  
# 将分数按照从小到大排列  
sort(scores)  
# 输出排序后的成绩  
length = len(scores)  
for i in range(0,length):  
 print(scores[i],',',end='')

输出结果为：



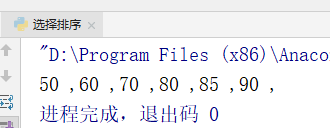
[12.](#_Toc494266851)选择排序算法原理？

答：将数组中剩下的没有排序的元素中选出最小的一个，插入已经排序的后面



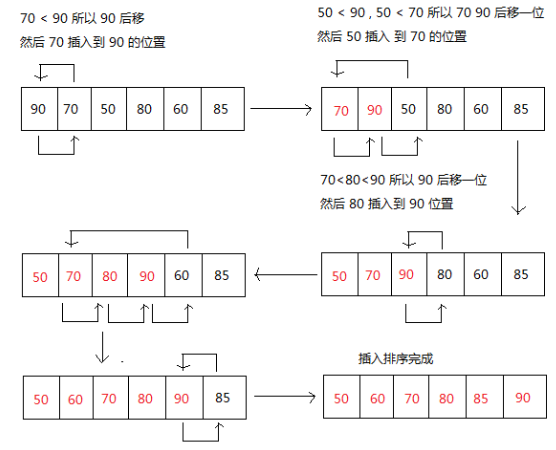
以此类推，剩下的没有排序的元素中选出最小的，插入已经排序的后面，得到从小到到序列。

#-\*- coding:utf-8 \_\*-   
def sort(arrays):  
 length = len(arrays) - 1  
 minIndex = 0 # 保存选择最小值的位置索引  
 for i in range(0,length):  
 minIndex = i  
 minValue = arrays[minIndex] # 保存每次循环最小值为循环的第一个元素的值  
 for j in range(i,length):  
 # 与每个元素比较如果小于最小值则交换值并且保存交换值的索引为最小值索引  
 if minValue > arrays[j + 1]:  
 minValue = arrays[j + 1]  
 minIndex = j + 1  
 if i != minIndex: # 判断如果最小值索引改变则把当前最小值与已排序的最后一个元素交换  
 temp = arrays[i]  
 arrays[i] = arrays[minIndex]  
 arrays[minIndex] = temp  
scores = [90,70,50,80,60,85]  
sort(scores)  
# 输出排序后的成绩  
length = len(scores)  
for i in range(0,length):  
 print(scores[i],',',end='')



[13.](#_Toc494266851)插入排序算法原理？

答：在数组中取未排序的新元素，跟已排序的元素比较，如果新元素小于已排序的元素右移，插入新元素。



def sort(arrays):

length = len(arrays)

for i in range(0,length):

insertElement = arrays[i] # 取未排序的新元素

insertPosition = i # 插入的位置

for j in range(insertPosition-1,-1,-1):

# 如果新元素小于已排序的元素右移

if insertElement < arrays[j]:

arrays[j+1] = arrays[j]

insertPosition -= 1

arrays[insertPosition] = insertElement # 插入新元素

scores = [90,70,50,80,60,85]

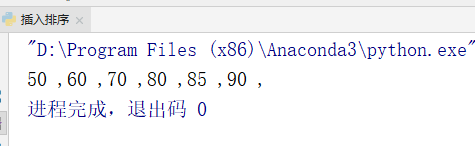
sort(scores)

# 输出排序后的成绩

length = len(scores)

for i in range(0,length):

print(scores[i],',',end='')



14.快速排序算法原理？

答：把数组分成2个子数组，然后递归调用自己，为子数组再快速排序直到完成。

def sort(array):

length = len(array)

if (length > 0):

quickSort(array,0,length - 1)

def quickSort(array,low,high):

if (low > high):

return

i = low

j = high

# 基准值

threshold = array[low]

# 从表的两端交替向中间扫描

while(i < j):

# 从右往左找到第一个小于threshold的数的位置

while (i<j and array[j]>threshold):

j -= 1

if (i<j):

array[i] = array[j] # 用比threshold小的数替换低位

i += 1

# 从左往右找到第一个大于threshold的数的位置

while(i<j and array[i]<=threshold):

i += 1

if (i<j): # 用比threshold大的数替换高位

array[j] = array[i]

j -= 1

array[i] = threshold # 将threshold替换回array[i]

# 对threshold左边的数快排

quickSort(array,low,i - 1)

# 对threshold右边的数快排

quickSort(array,i+1,high)

scores = [99,65,50,87,74,63,76,100,92]

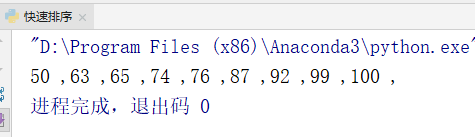
sort(scores)

# 输出排序后的成绩

length = len(scores)

for i in range(0,length):

print(scores[i],',',end='')



15.希尔排序算法原理？

高效插入排序的一种，把数组按下标的一定增量分组，对每组插入排序；随着增量逐渐减少，直至增量为1，整个数据被分成一组，排序完成。

def shellSort(array):

length = len(array)

#增量gap,并逐步缩小增量

gap = length // 2

while gap > 0 :

# 从第gap个元素，逐个对所在组进行插入排序

for i in range(gap,length):

j = i

while j-gap >= 0 and array[j] < array[j-gap]:

# 交换数组元素

swap(array,j,j-gap)

j = j-gap

gap = gap // 2

# 交换数组元素

def swap(array,a,b):

array[a] = array[a] + array[b]

array[b] = array[a] - array[b]

array[a] = array[a] - array[b]

scores = [99,65,50,87,55,74,63,76,100,92]

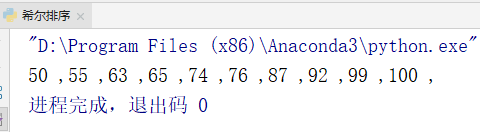
shellSort(scores)

# 输出排序后的成绩

length = len(scores)

for i in range(0,length):

print(scores[i],',',end='')



16.二叉堆排序算法原理？

二叉堆：完全二叉树中非终端结点的值均不大于（或不小于）其左、右孩子节点的值。

小顶堆：ki<=k2i 且ki<= k2i+1

大顶堆：ki>=k2i 且ki>=k2i+1

父结点下标 （i-1)/2

左子结点下标 2\*i+1

右子结点下标 2\*i+2

堆排序过程：

1.无序序列建堆

2.输出堆顶元素之后，自上向下进行调整，将堆顶元素和它的左右子树的根结点比较，把最小的元素交换到堆顶；然后不断调整，直到叶子结点，得到新堆。

class HeapSort:

array = []

# 初始化创建堆

def createHeap(self,array):

self.array = array

length = len(array)

# 对数组建堆，(length-1)//2 扫描一半有子节点的节点

for i in range((length-1)//2,-1,-1):

self.adjustHeap(i,length-1)

# 调整堆

def adjustHeap(self,currentIndex,maxLength):

noLeafValue = self.array[currentIndex] # 当前非叶子节点

# 2\*currentIndex+1 当前做子树下标

j = 2 \* currentIndex + 1

while(j<=maxLength):

if(j<maxLength and self.array[j]<self.array[j+1]):

j += 1 # j 较大的下标

if(noLeafValue >= self.array[j]):

break

self.array[currentIndex] = self.array[j]

# 上移到父节点

currentIndex = j

j = currentIndex \*2 +1

self.array[currentIndex] = noLeafValue # 要放入的位置

# 堆排序

def sort(self):

length = len(self.array)

for i in range(length-1,0,-1):

temp = self.array[0]

self.array[0] = self.array[i]

self.array[i] = temp

self.adjustHeap(0,i-1)

heapSort = HeapSort()

scores = [10,90,20,80,30,70,40,60,50]

print('没有建堆之前： ')

length = len(scores)

for i in range(0,length):

print(scores[i],',',end='')

print('\n\n')

print('建堆之后： ')

heapSort.createHeap(scores)

length = len(scores)

for i in range(0,length):

print(scores[i],',',end='')

print('\n\n')

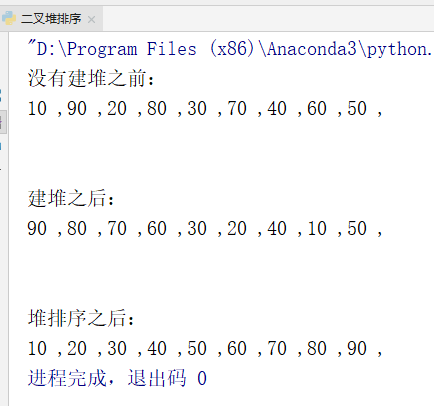
print('堆排序之后： ')

heapSort.sort()

length = len(scores)

for i in range(0,length):

print(scores[i],',',end='')



【链接】数据结构基础概念篇

https://blog.csdn.net/qq\_31196849/article/details/78529724

【链接】数据结构（全）

https://blog.csdn.net/heyuchang666/article/details/49891635

# 三、 数据挖掘面试题

## 数据挖掘150道试题总结

### 一、单选题(1-80)

1. 某超市研究销售纪录数据后发现，买啤酒的人很大概率也会购买尿布，这种属于数据挖掘的哪类问题？(A)  
A. 关联规则发现  
B. 聚类  
C. 分类  
D. 自然语言处理

2. 以下两种描述分别对应哪两种对分类算法的评价标准？ (A)  
(a)警察抓小偷，描述警察抓的人中有多少个是小偷的标准。  
(b)描述有多少比例的小偷给警察抓了的标准。

A. Precision, Recall  
B. Recall, Precision  
C. Precision, ROC  
D. Recall, ROC

3. 将原始数据进行集成、变换、维度规约、数值规约是在以下哪个步骤的任务？(C)  
A. 频繁模式挖掘  
B. 分类和预测  
C. 数据预处理  
D. 数据流挖掘

4. 当不知道数据所带标签时，可以使用哪种技术促使带同类标签的数据与带其他标签的数据相分离？(B)  
A. 分类  
B. 聚类  
C. 关联分析  
D. 隐马尔可夫链

5. 什么是KDD？ (A)  
A. 数据挖掘与知识发现  
B. 领域知识发现  
C. 文档知识发现  
D. 动态知识发现

6. 使用交互式的和可视化的技术，对数据进行探索属于数据挖掘的哪一类任务？（A）  
A. 探索性数据分析  
B. 建模描述  
C. 预测建模  
D. 寻找模式和规则

7. 为数据的总体分布建模；把多维空间划分成组等问题属于数据挖掘的哪一类任务？(B)  
A. 探索性数据分析  
B. 建模描述  
C. 预测建模  
D. 寻找模式和规则

8. 建立一个模型，通过这个模型根据已知的变量值来预测其他某个变量值属于数据挖掘的哪一类任务？(C)  
A. 根据内容检索  
B. 建模描述  
C. 预测建模  
D. 寻找模式和规则

9. 用户有一种感兴趣的模式并且希望在数据集中找到相似的模式，属于数据挖掘哪一类任务？(A)  
A. 根据内容检索  
B. 建模描述  
C. 预测建模  
D. 寻找模式和规则

11.下面哪种不属于数据预处理的方法？ (D)  
A变量代换  
B离散化  
C聚集  
D估计遗漏值

12. 假设12个销售价格记录组已经排序如下：5, 10, 11, 13, 15,35, 50, 55, 72, 92, 204, 215 使用如下每种方法将它们划分成四个箱。等频（等深）划分时，15在第几个箱子内？ (B)  
A 第一个  
B 第二个  
C 第三个  
D 第四个

13.上题中，等宽划分时（宽度为50），15又在哪个箱子里？ (A)  
A 第一个  
B 第二个  
C 第三个  
D 第四个

14.下面哪个不属于数据的属性类型：(D)  
A 标称  
B 序数  
C 区间  
D相异

15. 在上题中，属于定量的属性类型是：(C)  
A 标称  
B 序数  
C 区间  
D 相异

16. 只有非零值才重要的二元属性被称作：( C )  
A 计数属性  
B 离散属性  
C非对称的二元属性  
D 对称属性

17. 以下哪种方法不属于特征选择的标准方法： (D)  
A 嵌入  
B 过滤  
C 包装  
D 抽样

18.下面不属于创建新属性的相关方法的是： (B)  
A特征提取  
B特征修改  
C映射数据到新的空间  
D特征构造

19. 考虑值集{1、2、3、4、5、90}，其截断均值（p=20%）是 (C)  
A 2  
B 3  
C 3.5  
D 5

20. 下面哪个属于映射数据到新的空间的方法？ (A)  
A 傅立叶变换  
B 特征加权  
C 渐进抽样  
D 维归约

21. 熵是为消除不确定性所需要获得的信息量，投掷均匀正六面体骰子的熵是： (B)  
A 1比特  
B 2.6比特  
C 3.2比特  
D 3.8比特

22. 假设属性income的最大最小值分别是12000元和98000元。利用最大最小规范化的方法将属性的值映射到0至1的范围内。对属性income的73600元将被转化为：(D)  
A 0.821  
B 1.224  
C 1.458  
D 0.716

23.假定用于分析的数据包含属性age。数据元组中age的值如下（按递增序）：13，15，16，16，19，20，20，21，22，22，25，25，25，30，33，33，35，35，36，40，45，46，52，70, 问题：使用按箱平均值平滑方法对上述数据进行平滑，箱的深度为3。第二个箱子  
值为：(A)  
A 18.3  
B 22.6  
C 26.8  
D 27.9

24. 考虑值集{12 24 33 2 4 55 68 26}，其四分位数极差是：(A)  
A 31  
B 24  
C 55  
D 3

25. 一所大学内的各年纪人数分别为：一年级200人，二年级160人，三年级130人，四年级110人。则年级属性的众数是： (A)  
A 一年级  
B二年级  
C 三年级  
D 四年级

26. 下列哪个不是专门用于可视化时间空间数据的技术： (B)  
A 等高线图  
B 饼图  
C 曲面图  
D 矢量场图

27. 在抽样方法中，当合适的样本容量很难确定时，可以使用的抽样方法是： (D)  
A 有放回的简单随机抽样  
B 无放回的简单随机抽样  
C 分层抽样  
D 渐进抽样

28. 数据仓库是随着时间变化的,下面的描述不正确的是 (C)  
A. 数据仓库随时间的变化不断增加新的数据内容;  
B. 捕捉到的新数据会覆盖原来的快照;  
C. 数据仓库随事件变化不断删去旧的数据内容;  
D. 数据仓库中包含大量的综合数据,这些综合数据会随着时间的变化不断地进行重新综合.

29. 关于基本数据的元数据是指: (D)  
A. 基本元数据与数据源,数据仓库,数据集市和应用程序等结构相关的信息;  
B. 基本元数据包括与企业相关的管理方面的数据和信息;  
C. 基本元数据包括日志文件和简历执行处理的时序调度信息;  
D. 基本元数据包括关于装载和更新处理,分析处理以及管理方面的信息.

30. 下面关于数据粒度的描述不正确的是: (C)  
A. 粒度是指数据仓库小数据单元的详细程度和级别;  
B. 数据越详细,粒度就越小,级别也就越高;  
C. 数据综合度越高,粒度也就越大,级别也就越高;  
D. 粒度的具体划分将直接影响数据仓库中的数据量以及查询质量.

31. 有关数据仓库的开发特点,不正确的描述是: (A)  
A. 数据仓库开发要从数据出发;  
B. 数据仓库使用的需求在开发出去就要明确;  
C. 数据仓库的开发是一个不断循环的过程,是启发式的开发;  
D. 在数据仓库环境中,并不存在操作型环境中所固定的和较确切的处理流,数据仓库中数据分析和处理更灵活,且没有固定的模式

32. 在有关数据仓库测试,下列说法不正确的是: (D)  
A. 在完成数据仓库的实施过程中,需要对数据仓库进行各种测试.测试工作中要包括单元测试和系统测试.  
B. 当数据仓库的每个单独组件完成后,就需要对他们进行单元测试.  
C. 系统的集成测试需要对数据仓库的所有组件进行大量的功能测试和回归测试.  
D. 在测试之前没必要制定详细的测试计划.

33. OLAP技术的核心是: (D)  
A. 在线性;  
B. 对用户的快速响应;  
C. 互操作性.  
D. 多维分析;

34. 关于OLAP的特性,下面正确的是: (D)  
(1)快速性 (2)可分析性 (3)多维性 (4)信息性 (5)共享性  
A. (1) (2) (3)  
B. (2) (3) (4)  
C. (1) (2) (3) (4)  
D. (1) (2) (3) (4) (5)

35. 关于OLAP和OLTP的区别描述,不正确的是: (C)  
A. OLAP主要是关于如何理解聚集的大量不同的数据.它与OTAP应用程序不同.  
B. 与OLAP应用程序不同,OLTP应用程序包含大量相对简单的事务.  
C. OLAP的特点在于事务量大,但事务内容比较简单且重复率高.  
D. OLAP是以数据仓库为基础的,但其最终数据来源与OLTP一样均来自底层的数据库系统,两者面对的用户是相同的.

36. OLAM技术一般简称为”数据联机分析挖掘”,下面说法正确的是: (D)  
A. OLAP和OLAM都基于客户机/服务器模式,只有后者有与用户的交互性;  
B. 由于OLAM的立方体和用于OLAP的立方体有本质的区别.  
C. 基于WEB的OLAM是WEB技术与OLAM技术的结合.  
D. OLAM服务器通过用户图形借口接收用户的分析指令,在元数据的知道下,对超级立方体作一定的操作.

37. 关于OLAP和OLTP的说法,下列不正确的是: (A)  
A. OLAP事务量大,但事务内容比较简单且重复率高.  
B. OLAP的最终数据来源与OLTP不一样.  
C. OLTP面对的是决策人员和高层管理人员.  
D. OLTP以应用为核心,是应用驱动的.

38. 设X={1，2，3}是频繁项集，则可由X产生\_\_(C)\_\_个关联规则。  
A、4  
B、5  
C、6  
D、7

40. 概念分层图是\_\_(B)\_\_图。  
A、无向无环  
B、有向无环  
C、有向有环  
D、无向有环

41. 频繁项集、频繁闭项集、最大频繁项集之间的关系是： (C)  
A、频繁项集 频繁闭项集 =最大频繁项集  
B、频繁项集 = 频繁闭项集 最大频繁项集  
C、频繁项集 频繁闭项集 最大频繁项集  
D、频繁项集 = 频繁闭项集 = 最大频繁项集

42. 考虑下面的频繁3-项集的集合：{1，2，3}，{1，2，4}，{1，2，5}，{1，3，4}，{1，3，5}，{2，3，4}，{2，3，5}，{3，4，5}假定数据集中只有5个项，采用 合并策略，由候选产生过程得到4-项集不包含（C）  
A、1，2，3，4  
B、1，2，3，5  
C、1，2，4，5  
D、1，3，4，5

43.下面选项中t不是s的子序列的是 ( C )  
A、s=<{2,4},{3,5,6},{8}> t=<{2},{3,6},{8}>  
B、s=<{2,4},{3,5,6},{8}> t=<{2},{8}>  
C、s=<{1,2},{3,4}> t=<{1},{2}>  
D、s=<{2,4},{2,4}> t=<{2},{4}>

44. 在图集合中发现一组公共子结构，这样的任务称为 ( B )  
A、频繁子集挖掘  
B、频繁子图挖掘  
C、频繁数据项挖掘  
D、频繁模式挖掘

45. 下列度量不具有反演性的是 (D)  
A、系数  
B、几率  
C、Cohen度量  
D、兴趣因子

46. 下列\_\_(A)\_\_不是将主观信息加入到模式发现任务中的方法。  
A、与同一时期其他数据对比  
B、可视化  
C、基于模板的方法  
D、主观兴趣度量

47. 下面购物篮能够提取的3-项集的最大数量是多少（C）

ID 购买项  
1 牛奶，啤酒，尿布  
2 面包，黄油，牛奶  
3 牛奶，尿布，饼干  
4 面包，黄油，饼干  
5 啤酒，饼干，尿布  
6 牛奶，尿布，面包，黄油  
7 面包，黄油，尿布  
8 啤酒，尿布  
9 牛奶，尿布，面包，黄油  
10 啤酒，饼干

A、1  
B、2  
C、3  
D、4

48. 以下哪些算法是分类算法，（B）  
A，DBSCAN  
B，C4.5  
C,K-Mean  
D,EM

49. 以下哪些分类方法可以较好地避免样本的不平衡问题， （A）

A，KNN  
B，SVM  
C，Bayes  
D，神经网络

50. 决策树中不包含一下哪种结点， (C)

A,根结点（root node)  
B,内部结点（internal node）  
C,外部结点（external node）  
D,叶结点（leaf node）

51. 不纯性度量中Gini计算公式为（其中c是类的个数） (A)  
A, B, C, D, （A）

53. 以下哪项关于决策树的说法是错误的 (C)  
A. 冗余属性不会对决策树的准确率造成不利的影响  
B. 子树可能在决策树中重复多次  
C. 决策树算法对于噪声的干扰非常敏感  
D. 寻找最佳决策树是NP完全问题

54. 在基于规则分类器的中，依据规则质量的某种度量对规则排序，保证每一个测试记录都是由覆盖它的“最好的”规格来分类，这种方案称为 (B)  
A. 基于类的排序方案  
B. 基于规则的排序方案  
C. 基于度量的排序方案  
D. 基于规格的排序方案。

55. 以下哪些算法是基于规则的分类器 (A)  
A. C4.5  
B. KNN  
C. Na?ve Bayes  
D. ANN

56. 如果规则集R中不存在两条规则被同一条记录触发，则称规则集R中的规则为（C）；  
A, 无序规则  
B，穷举规则  
C， 互斥规则  
D，有序规则

57. 如果对属性值的任一组合，R中都存在一条规则加以覆盖，则称规则集R中的规则为(B)  
A, 无序规则  
B，穷举规则  
C， 互斥规则  
D，有序规则

58. 如果规则集中的规则按照优先级降序排列，则称规则集是 (D)  
A, 无序规则  
B，穷举规则  
C， 互斥规则  
D，有序规则

59. 如果允许一条记录触发多条分类规则，把每条被触发规则的后件看作是对相应类的一次投票，然后计票确定测试记录的类标号，称为（A）  
A, 无序规则  
B，穷举规则  
C， 互斥规则  
D，有序规则

60. 考虑两队之间的足球比赛：队0和队1。假设65%的比赛队0胜出，剩余的比赛队1获胜。队0获胜的比赛中只有30%是在队1的主场，而队1取胜的比赛中75%是主场获胜。如果下一场比赛在队1的主场进行队1获胜的概率为 (C)  
A，0.75  
B,0.35  
C,0.4678  
D, 0.5738

61. 以下关于人工神经网络（ANN）的描述错误的有 (A)  
A，神经网络对训练数据中的噪声非常鲁棒  
B，可以处理冗余特征  
C，训练ANN是一个很耗时的过程  
D，至少含有一个隐藏层的多层神经网络

62. 通过聚集多个分类器的预测来提高分类准确率的技术称为 (A)  
A,组合(ensemble)  
B,聚集(aggregate)  
C，合并(combination)  
D，投票(voting)

63. 简单地将数据对象集划分成不重叠的子集，使得每个数据对象恰在一个子集中，这种聚类类型称作（ B ）  
A、层次聚类  
B、划分聚类  
C、非互斥聚类  
D、模糊聚类

64. 在基本K均值算法里，当邻近度函数采用（ A ）的时候，合适的质心是簇中各点的中位数。  
A、曼哈顿距离  
B、平方欧几里德距离  
C、余弦距离  
D、Bregman散度

65.（ C ）是一个观测值，它与其他观测值的差别如此之大，以至于怀疑它是由不同的机制产生的。  
A、边界点  
B、质心  
C、离群点  
D、核心点

66. BIRCH是一种（ B ）。  
A、分类器  
B、聚类算法  
C、关联分析算法  
D、特征选择算法

67. 检测一元正态分布中的离群点，属于异常检测中的基于（ A ）的离群点检测。  
A、统计方法  
B、邻近度  
C、密度  
D、聚类技术

68.（ C ）将两个簇的邻近度定义为不同簇的所有点对的平均逐对邻近度，它是一种凝聚层次聚类技术。  
A、MIN（单链）  
B、MAX（全链）  
C、组平均  
D、Ward方法

69.（ D ）将两个簇的邻近度定义为两个簇合并时导致的平方误差的增量，它是一种凝聚层次聚类技术。  
A、MIN（单链）  
B、MAX（全链）  
C、组平均  
D、Ward方法

70. DBSCAN在最坏情况下的时间复杂度是（ B ）。  
A、O(m)  
B、O(m2)  
C、O(log m)  
D、O(m\*log m)

71. 在基于图的簇评估度量表里面，如果簇度量为proximity(Ci , C)，簇权值为mi ，那么它的类型是（ C ）。  
A、基于图的凝聚度  
B、基于原型的凝聚度  
C、基于原型的分离度  
D、基于图的凝聚度和分离度

72. 关于K均值和DBSCAN的比较，以下说法不正确的是（ A ）。  
A、K均值丢弃被它识别为噪声的对象，而DBSCAN一般聚类所有对象。  
B、K均值使用簇的基于原型的概念，而DBSCAN使用基于密度的概念。  
C、K均值很难处理非球形的簇和不同大小的簇，DBSCAN可以处理不同大小和不同形状的簇。  
D、K均值可以发现不是明显分离的簇，即便簇有重叠也可以发现，但是DBSCAN会合并有重叠的簇。

73. 以下是哪一个聚类算法的算法流程：①构造k－最近邻图。②使用多层图划分算法划分图。③repeat：合并关于相对互连性和相对接近性而言，最好地保持簇的自相似性的簇。④until：不再有可以合并的簇。（ C ）。  
A、MST  
B、OPOSSUM  
C、Chameleon  
D、Jarvis－Patrick（JP）

74. 考虑这么一种情况：一个对象碰巧与另一个对象相对接近，但属于不同的类，因为这两个对象一般不会共享许多近邻，所以应该选择（ D ）的相似度计算方法。  
A、平方欧几里德距离  
B、余弦距离  
C、直接相似度  
D、共享最近邻

75. 以下属于可伸缩聚类算法的是（ A ）。  
A、CURE  
B、DENCLUE  
C、CLIQUE  
D、OPOSSUM

76. 以下哪个聚类算法不是属于基于原型的聚类（ D ）。  
A、模糊c均值  
B、EM算法  
C、SOM  
D、CLIQUE

77. 关于混合模型聚类算法的优缺点，下面说法正确的是（ B ）。  
A、当簇只包含少量数据点，或者数据点近似协线性时，混合模型也能很好地处理。  
B、混合模型比K均值或模糊c均值更一般，因为它可以使用各种类型的分布。  
C、混合模型很难发现不同大小和椭球形状的簇。  
D、混合模型在有噪声和离群点时不会存在问题。

78. 以下哪个聚类算法不属于基于网格的聚类算法（ D ）。  
A、STING  
B、WaveCluster  
C、MAFIA  
D、BIRCH

79. 一个对象的离群点得分是该对象周围密度的逆。这是基于（ C ）的离群点定义。  
A．概率  
B、邻近度  
C、密度  
D、聚类

80. 下面关于Jarvis－Patrick（JP）聚类算法的说法不正确的是（ D ）。  
A、JP聚类擅长处理噪声和离群点，并且能够处理不同大小、形状和密度的簇。  
B、JP算法对高维数据效果良好，尤其擅长发现强相关对象的紧致簇。  
C、JP聚类是基于SNN相似度的概念。  
D、JP聚类的基本时间复杂度为O(m)。

### 二、 多选题(1-30)

1. 通过数据挖掘过程所推倒出的关系和摘要经常被称为：(A B)  
A. 模型  
B. 模式  
C. 模范  
D. 模具

2 寻找数据集中的关系是为了寻找精确、方便并且有价值地总结了数据的某一特征的表示，这个过程包括了以下哪些步骤？ (A B C D)  
A. 决定要使用的表示的特征和结构  
B. 决定如何量化和比较不同表示拟合数据的好坏  
C. 选择一个算法过程使评分函数最优  
D. 决定用什么样的数据管理原则以高效地实现算法。

3. 数据挖掘的预测建模任务主要包括哪几大类问题？ (A B)  
A. 分类  
B. 回归  
C. 模式发现  
D. 模式匹配

4. 数据挖掘算法的组件包括：(A B C D)  
A. 模型或模型结构  
B. 评分函数  
C. 优化和搜索方法  
D. 数据管理策略

5. 以下哪些学科和数据挖掘有密切联系？(A D)  
A. 统计  
B. 计算机组成原理  
C. 矿产挖掘  
D. 人工智能

6. 在现实世界的数据中，元组在某些属性上缺少值是常有的。描述处理该问题的各种方法有： (ＡＢＣＤＥ)  
A忽略元组  
B使用属性的平均值填充空缺值  
C使用一个全局常量填充空缺值  
D使用与给定元组属同一类的所有样本的平均值  
E使用最可能的值填充空缺值

7.下面哪些属于可视化高维数据技术 (ＡＢＣＥ)  
A 矩阵  
B 平行坐标系  
C星形坐标 D散布图  
E Chernoff脸

8. 对于数据挖掘中的原始数据，存在的问题有： (ＡＢＣＤＥ)  
A 不一致  
B重复  
C不完整  
D 含噪声  
E 维度高

9.下列属于不同的有序数据的有：(ＡＢＣＥ)  
A 时序数据  
B 序列数据  
C时间序列数据  
D事务数据  
E空间数据

10.下面属于数据集的一般特性的有：( B C D)  
A 连续性  
B 维度  
C 稀疏性  
D 分辨率  
E 相异性

11. 下面属于维归约常用的线性代数技术的有： (A C)  
A 主成分分析  
B 特征提取  
C 奇异值分解  
D 特征加权  
E 离散化

12. 下面列出的条目中，哪些是数据仓库的基本特征： (ACD)  
A. 数据仓库是面向主题的  
B. 数据仓库的数据是集成的  
C. 数据仓库的数据是相对稳定的  
D. 数据仓库的数据是反映历史变化的  
E. 数据仓库是面向事务的

13. 以下各项均是针对数据仓库的不同说法，你认为正确的有（BCDE ）。  
A．数据仓库就是数据库  
B．数据仓库是一切商业智能系统的基础  
C．数据仓库是面向业务的，支持联机事务处理（OLTP）  
D．数据仓库支持决策而非事务处理  
E．数据仓库的主要目标就是帮助分析，做长期性的战略制定

14. 数据仓库在技术上的工作过程是： (ABCD)  
A. 数据的抽取  
B. 存储和管理  
C. 数据的表现  
D. 数据仓库设计  
E. 数据的表现

15. 联机分析处理包括以下哪些基本分析功能？ (BCD)  
A. 聚类  
B. 切片  
C. 转轴  
D. 切块  
E. 分类

16. 利用Apriori算法计算频繁项集可以有效降低计算频繁集的时间复杂度。在以下的购物篮中产生支持度不小于3的候选3-项集，在候选2-项集中需要剪枝的是（BD）  
ID 项集  
1 面包、牛奶  
2 面包、尿布、啤酒、鸡蛋  
3 牛奶、尿布、啤酒、可乐  
4 面包、牛奶、尿布、啤酒  
5 面包、牛奶、尿布、可乐

A、啤酒、尿布  
B、啤酒、面包  
C、面包、尿布  
D、啤酒、牛奶

17. 下表是一个购物篮，假定支持度阈值为40%，其中\_\_(A D)\_\_是频繁闭项集。  
TID 项  
1 abc  
2 abcd  
3 bce  
4 acde  
5 de  
A、abc  
B、ad  
C、cd  
D、de

18. Apriori算法的计算复杂度受\_\_(ABCD)?\_\_影响。  
A、支持度阀值  
B、项数（维度）  
C、事务数  
D、事务平均宽度

19. 非频繁模式\_\_(AD)\_\_  
A、其支持度小于阈值  
B、都是不让人感兴趣的  
C、包含负模式和负相关模式  
D、对异常数据项敏感

20. 以下属于分类器评价或比较尺度的有: (ACD)

A,预测准确度  
B,召回率  
C,模型描述的简洁度  
D,计算复杂度

21. 在评价不平衡类问题分类的度量方法有如下几种，(ABCD)  
A,F1度量  
B,召回率（recall）  
C,精度（precision）  
D,真正率（ture positive rate,TPR）

22. 贝叶斯信念网络(BBN)有如下哪些特点， （AB）  
A,构造网络费时费力  
B,对模型的过分问题非常鲁棒  
C,贝叶斯网络不适合处理不完整的数据  
D,网络结构确定后，添加变量相当麻烦

23. 如下哪些不是最近邻分类器的特点， (C)  
A,它使用具体的训练实例进行预测，不必维护源自数据的模型  
B,分类一个测试样例开销很大  
C,最近邻分类器基于全局信息进行预测  
D,可以生产任意形状的决策边界

24. 如下那些不是基于规则分类器的特点，（AC）  
A,规则集的表达能力远不如决策树好  
B,基于规则的分类器都对属性空间进行直线划分，并将类指派到每个划分  
C,无法被用来产生更易于解释的描述性模型  
D,非常适合处理类分布不平衡的数据集

25. 以下属于聚类算法的是（ ABD ）。  
A、K均值  
B、DBSCAN  
C、Apriori  
D、Jarvis-Patrick（JP）

26.（ CD ）都属于簇有效性的监督度量。  
A、轮廓系数  
B、共性分类相关系数  
C、熵  
D、F度量

27. 簇有效性的面向相似性的度量包括（ BC ）。  
A、精度  
B、Rand统计量  
C、Jaccard系数  
D、召回率

28.（ ABCD ）这些数据特性都是对聚类分析具有很强影响的。  
A、高维性  
B、规模  
C、稀疏性  
D、噪声和离群点

29. 在聚类分析当中，（ AD ）等技术可以处理任意形状的簇。  
A、MIN（单链）  
B、MAX（全链）  
C、组平均  
D、Chameleon

30. （ AB ）都属于分裂的层次聚类算法。  
A、二分K均值  
B、MST  
C、Chameleon  
D、组平均

### 三、 判断题(1-39)

1. 数据挖掘的主要任务是从数据中发现潜在的规则，从而能更好的完成描述数据、预测数据等任务。 (对)

2. 数据挖掘的目标不在于数据采集策略，而在于对于已经存在的数据进行模式的发掘。（对）3. 图挖掘技术在社会网络分析中扮演了重要的角色。（对）

4. 模式为对数据集的全局性总结，它对整个测量空间的每一点做出描述；模型则对变量变化空间的一个有限区域做出描述。（错）

5. 寻找模式和规则主要是对数据进行干扰，使其符合某种规则以及模式。（错）

6. 离群点可以是合法的数据对象或者值。　　　　（对）

7. 离散属性总是具有有限个值。　　　　　　　　（错）

8. 噪声和伪像是数据错误这一相同表述的两种叫法。　　　　　（错）

9. 用于分类的离散化方法之间的根本区别在于是否使用类信息。　　　（对）

10. 特征提取技术并不依赖于特定的领域。　　　　　　（错）

11. 序列数据没有时间戳。　　　　　　（对）

12. 定量属性可以是整数值或者是连续值。　　　　　（对）

13. 可视化技术对于分析的数据类型通常不是专用性的。　　　　（错）

14. DSS主要是基于数据仓库.联机数据分析和数据挖掘技术的应用。（对）

15. OLAP技术侧重于把数据库中的数据进行分析、转换成辅助决策信息，是继数据库技术发展之后迅猛发展起来的一种新技术。 （对）

16. 商业智能系统与一般交易系统之间在系统设计上的主要区别在于：后者把结构强加于商务之上，一旦系统设计完毕，其程序和规则不会轻易改变；而前者则是一个学习型系统，能自动适应商务不断变化的要求。 （对）

17. 数据仓库中间层OLAP服务器只能采用关系型OLAP （错）

18．数据仓库系统的组成部分包括数据仓库，仓库管理，数据抽取，分析工具等四个部分. (错)

19. Web数据挖掘是通过数据库仲的一些属性来预测另一个属性,它在验证用户提出的假设过程中提取信息. （错）

21. 关联规则挖掘过程是发现满足最小支持度的所有项集代表的规则。（错）

22. 利用先验原理可以帮助减少频繁项集产生时需要探查的候选项个数（对）。

23. 先验原理可以表述为：如果一个项集是频繁的，那包含它的所有项集也是频繁的。（错）

24. 如果规则 不满足置信度阈值，则形如 的规则一定也不满足置信度阈值，其中 是X的子集。（对）

25. 具有较高的支持度的项集具有较高的置信度。（错）

26. 聚类（clustering）是这样的过程：它找出描述并区分数据类或概念的模型(或函数)，以便能够使用模型预测类标记未知的对象类。 （错）

27. 分类和回归都可用于预测，分类的输出是离散的类别值，而回归的输出是连续数值。(对)

28. 对于SVM分类算法，待分样本集中的大部分样本不是支持向量，移去或者减少这些样本对分类结果没有影响。 （对）

29. Bayes法是一种在已知后验概率与类条件概率的情况下的模式分类方法，待分样本的分类结果取决于各类域中样本的全体。 (错)

30.分类模型的误差大致分为两种：训练误差（training error）和泛化误差（generalization error）. (对)

31. 在决策树中，随着树中结点数变得太大，即使模型的训练误差还在继续减低，但是检验误差开始增大，这是出现了模型拟合不足的问题。 （错）

32. SVM是这样一个分类器，他寻找具有最小边缘的超平面，因此它也经常被称为最小边缘分类器（minimal margin classifier） (错)

33. 在聚类分析当中，簇内的相似性越大，簇间的差别越大，聚类的效果就越差。（错）

34. 聚类分析可以看作是一种非监督的分类。（对）

35. K均值是一种产生划分聚类的基于密度的聚类算法，簇的个数由算法自动地确定。（错）

36. 给定由两次运行K均值产生的两个不同的簇集，误差的平方和最大的那个应该被视为较优。（错）

37. 基于邻近度的离群点检测方法不能处理具有不同密度区域的数据集。（对）

38. 如果一个对象不强属于任何簇，那么该对象是基于聚类的离群点。（对）

39. 从点作为个体簇开始，每一步合并两个最接近的簇，这是一种分裂的层次聚类方法。（错）40. DBSCAN是相对抗噪声的，并且能够处理任意形状和大小的簇。（对）

四、 自然语言处理面试题

1. XX

2. XX

3. XX

4. XX

负责人：徐英然

# 五、 知名企业面试题汇总

## **腾讯**

问：

方差和偏差的区别。

答：

偏差形容的是在训练过程中输出的预测结果与真实样本的差距，简单的说，就是形容模型在样本的拟合程度好不好，一般情况下，模型越复杂，也就是模型参数越多，偏差会越低，但是与此同时会带来过拟合的现象，一般情况下的解决办法有几种：（1）增加数据集，使得模型从数据集中学习到更多的特征，减少噪声的影响；（2）降低模型的复杂程度，比如减少神经网络中隐藏层的层数或者神经元个数（3）正则化手段，比如普通机器学习中，常常采用L1L2正则对训练的参数进行稀疏化处理，神经网络中则是加入droupout，目的减少训练参数。（4）集成的方法，比如bagging、随机森林等；方差则是描述模型在测试集中的表现，一般情况下，模型参数越少，测试集在方差上表现越好，但是也容易发生过欠拟合的现象。欠拟合的处理方式跟过拟合有些相反，主要的手段有：（1）增加新的特征 （2）增加正则化系数 （3）增加模型复杂度。

问：

怎么处理缺失值。

答：

缺失值得处理主要看缺失值的样本数占总样本数的比例大小 （1）占比小时，一般会直接剔除 （2）占比大时，有两种方式去处理，一种是基于统计量，比如均值、中位数、众数，另一种是模型预测的方法，选择未缺失的字段将其作为训练样本，缺失的字段作为预测值，进行模型训练后，对缺失的部分进行预测

## **百度**

问：

**new 和 malloc的区别**

答：

1,malloc与free是C++/C语言的标准库函数，new/delete是C++的运算符。它们都可用于申请动态内存和释放内存。

2,对于非内部数据类型的对象而言，光用maloc/free无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。由于malloc/free是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free。

3,因此C++语言需要一个能完成动态内存分配和初始化工作的运算符new，以一个能完成清理与释放内存工作的运算符delete。注意new/delete不是库函数。

4,C++程序经常要调用C函数，而C程序只能用malloc/free管理动态内存

问：

**hash 冲突及解决办法**

答：

关键字值不同的元素可能会映象到哈希表的同一地址上就会发生哈希冲突。解决办法：

1）开放定址法：当冲突发生时，使用某种探查(亦称探测)技术在散列表中形成一个探查(测)序列。沿此序列逐个单元地查找，直到找到给定 的关键字，或者碰到一个开放的地址(即该地址单元为空)为止（若要插入，在探查到开放的地址，则可将待插入的新结点存人该地址单元）。查找时探查到开放的 地址则表明表中无待查的关键字，即查找失败。

2） 再哈希法：同时构造多个不同的哈希函数。

3）链地址法：将所有哈希地址为i的元素构成一个称为同义词链的单链表，并将单链表的头指针存在哈希表的第i个单元中，因而查找、插入和删除主要在同义词链中进行。链地址法适用于经常进行插入和删除的情况。

4）建立公共溢出区：将哈希表分为基本表和溢出表两部分，凡是和基本表发生冲突的元素，一律填入溢出表。

问：

**请列出几种文本特征提取算法**

答：

文档频率、信息增益、互信息、X^2统计、TF-IDF

问：

简述几种自然语言处理开源工具包

答：

LingPipe、FudanNLP、OpenNLP、CRF++、Standord CoreNLP、IKAnalyzer

问：

简述无监督和有监督算法的区别

答：

（1）有监督学习：对具有概念标记（分类）的训练样本进行学习，以尽可能对训练样本集外的数据进行标记（分类）预测。这里，所有的标记（分类）是已知的。因此，训练样本的岐义性低。

无监督学习：对没有概念标记（分类）的训练样本进行学习，以发现训练样本集中的结构性知识。这里，所有的标记（分类）是未知的。因此，训练样本的岐义性高。聚类就是典型的无监督学习

（2）有监督学习的样本全部带标记，无监督学习的样本全部不带标记。

PS:部分带标记的是半监督学习

（3）训练集有输入有输出是有监督，包括所有的回归算法分类算法，比如线性回归、决策树、神经网络、KNN、SVM等；训练集只有输入没有输出是无监督，包括所有的聚类算法，比如k-means 、PCA、 GMM等

问：

请简述几种熟悉的分类算法

答：

kNN，kMeans，决策树，随机森林等

## **搜狐**

问：

下面哪项技巧可用于关键词归一化（keyword normalization），即把关键词转化为其基本形式？

1. 词形还原（Lemmatization）

2.Levenshtein

3. 词干提取（Stemming）

4. 探测法（Soundex）

A) 1 和 2

B) 2 和 4

C) 1 和 3

D) 1、2 和 3

E) 2、3 和 4

F) 1、2、3 和 4

答：

C

词形还原和词干提取是用于关键词归一化的技术；Levenshtein 和探测法是用于字符串匹配的技术。

问：

N-gram 被定义为 N 个关键词组合在一起。从给定的句子可以产生多少二元组短语（Bigram）：

「Analytics Vidhya is a great source to learn data science」

A) 7

B) 8

C) 9

D) 10

E) 11

答：

C

二元组短语: Analytics Vidhya, Vidhya is, is a, a great, great source, source to, To learn, learn data, data science

问：

在执行了以下的文本清理步骤之后，可从下面的语句中生成多少三元组短语（trigram）：

* 停用词移除
* 使用单一空格替换标点符号

「#Analytics-vidhya is a great source to learn @data\_science.」

A) 3

B) 4

C) 5

D) 6

E) 7

答：

C

在执行了停用词移除和标点符号替换之后，文本变成：「Analytics vidhya great source learn data science」

三元组短语——Analytics vidhya great, vidhya great source, great source learn, source learn data, learn data science

问：

以下哪个正则表达式可用于标识文本对象中存在的日期：「The next meetup on data science will be held on 2017-09-21, previously it happened on 31/03, 2016」

A) \d{4}-\d{2}-\d{2}

B) (19|20)\d{2}-(0[1-9]|1[0-2])-[0-2][1-9]

C) (19|20)\d{2}-(0[1-9]|1[0-2])-([0-2][1-9]|3[0-1])

D) 没有一个

答：

D

问题背景 5-6：

你已经搜集了 10,000 行推特文本的数据并且没有其他信息。你想要创建一个推特分类模型，可以把每条推特分为三类：积极、消极、中性。

问：

下面哪个模型可以执行问题背景中提及的推特分类问题？

A) 朴素贝叶斯

B) 支持向量机

C) 以上都不是

答：

C

由于你被给定了推特数据并且没有其他信息，这意味着不存在目标变量，所以不可能训练一个监督学习模型，支持向量机和朴素贝叶斯都是监督学习技巧。

问：

通过把每个推特视为一个文档，你已经创建了一个数据的文档词矩阵。关于文件词矩阵以下哪项是正确的？

1. 从数据中移除停用词（stopwords）将会影响数据的维度

2. 数据中词的归一化将会减少数据的维度

3. 转化所有的小写单词将不会影响数据的维度

A) 只有 1

B) 只有 2

C) 只有 3

D) 1 和 2

E) 2 和 3

F) 1、2 和 3

答：

D

1 和 2 是正确的，因为停用词移除将会减少矩阵中特征的数量，词的归一化也将会减少不相关的特征，并且把所有的词变成小写也会减少数据维度。

问：

下面哪个特征可以用来提升分类模型的精度？

A) 词频计数

B) 语句的向量符号

C) 语音标签部分

D) 依赖度（Dependency）语法

E) 以上所有

答：

E

问：

关于主题建模，总体语句占比多少才是正确的？

1. 它是一个监督学习技巧

2. 线性判别分析（LDA）可用于执行主题建模

3. 模型中主题数量的选择不取决于数据的大小

4. 主题术语的数量与数据的大小成正比

A) 0

B) 25

C) 50

D) 75

E) 100

答：

A

LDA 是无监督学习模型，但 LDA 代表的是隐狄利克雷分布，而不是线性判别分析。模型中主题数量的选择直接与数据的大小成正比，而主题词条的数量并不直接与数据大小成正比。因此没有一个陈述是正确的。

## **阿里**

问：

在用于文本分类的隐狄利克雷分布（LDA）模型中，α 和 β 超参数表征什么？

A) α ：文档中的主题数量，β：假主题中的词条数量

B) α ：主题内生成的词条密度，β：假词条中生成的主题密度

C) α ：文档中的主题数量，β：假主题中的词条数量

D) α ：文档中生成的主题密度，β：真主题内生成的词密度

答：

D

问：

根据语句「I am planning to visit New Delhi to attend Analytics Vidhya Delhi Hackathon」解方程。

A = (名词作为语音标签的一部分)

B = (动词作为语音标签的一部分)

C = (频率计数大于 1)

下面哪个是A、B、 C 的正确值？

A) 5、5、2

B) 5、5、0

C) 7、5、1

D) 7、4、2

E) 6、4、3

答：

D

名词：I, New, Delhi, Analytics, Vidhya, Delhi, Hackathon (7)

动词：am, planning, visit, attend (4)

频率计数大于 1 的词：to, Delhi (2)

因此，选项 D 正确。

问：

在包含 N 个文档的语料库中，随机选择一个文档。该文件总共包含 T 个词，词条「数据」出现 K 次。如果词条「数据」出现在文件总数的数量接近三分之一，则 TF（词频）和 IDF（逆文档频率）的乘积的正确值是多少？

A) KT \* Log(3)

B) K \* Log(3) / T

C) T \* Log(3) / K

D) Log(3) / KT

答：

B

 TF 的公式是 K/T

IDF 的公式是 log

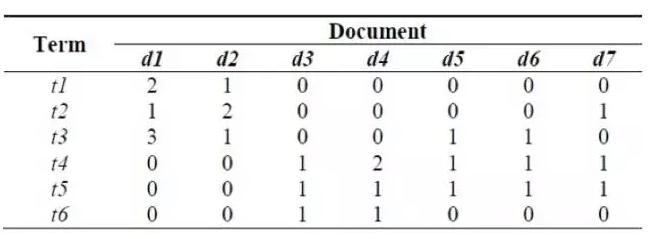
= log(1 / (⅓))

= log (3)

因此正确答案是 Klog(3)/T

下面三个问题的背景：

参阅以下的文档词矩阵



问：

下面哪个文档包含相同数量的词条，并且在整个语料库中其中一个文档的词数量不等同于其他任何文档的最低词数量。

A) d1 和 d4

B) d6 和 d7

C) d2 和 d4

D) d5 和 d6

答：

C

文档 d2 和 d4 包含 4 个词条并且不是词条最低数量 3。

问：

语料库中最常见和最稀少的词条分别是什么？

A) t4、t6

B) t3、t5

C) t5、t1

D) t5、t6

答：

A

T5 是最常见的词条，出现在 7 个文档中的 5 个，T6 是最稀疏的词条，只在 d3 和 d4 中出现。

问：

在该文件中使用最多次数的词条的频次是多少？

A) t6 – 2/5

B) t3 – 3/6

C) t4 – 2/6

D) t1 – 2/6

答：

B

t3 在整个语料库中的使用的最大次数是 3，t3 的 tf 是 3/6

问：

下列哪种方法不是灵活文本匹配的一部分？

A）字符串语音表示（Soundex）

B）语音发声散列（Metaphone）

C）编辑距离算法（Edit Distance）

D）关键词哈希算法（Keyword Hashing）

答：

D

除了关键词哈希算法，其它所有方法都用于灵活字串匹配

## **新浪**

问：

正误判断：Word2vec 模型是一种用于给文本目标创建矢量标记的机器学习模型。Word2vec 包含多个深度神经网络

A）正确

B）错误

答：

B

Word2vec 也包含预处理模型（preprocessing mode），它不属于深度神经网络。

问：

下列关于 Word2vec 的说法哪些是正确的（不定项选择）？

A）word2vec 的结构仅包括两层——连续性词包和 skip-gram 模型

B）连续性词包是浅层神经网络模型

C）Skip-gram 是深度神经网络模型

D）CBOW 和 Skip-gram 都是深度神经网络模型

E）以上表述全部正确

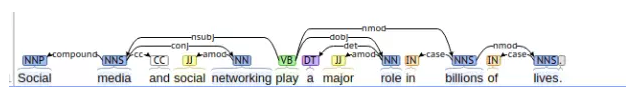
答：

D

Word2vec 包含连续性词包和 skip-gram 模型，都是深度神经网络

问：

关于无语境依赖关系图（context-free dependency graph），句子里有多少子决策树（sub-trees）？

 A）3

B）4

C）5

D）6

答：

D

依赖关系图中的子决策树可以被看做是拥有外部连接的节点，例如：Media, networking, play, role, billions, 和 lives 是子决策树的根。

问：

文本分类模型组成部分的正确顺序是：

1. 文本清理（Text cleaning）

2. 文本标注（Text annotation）

3. 梯度下降（Gradient descent）

4. 模型调优（Model tuning）

5. 文本到预测器（Text to predictors）

A) 12345

B) 13425

C) 12534

D) 13452

答：

C

正确的文本分类模型包含——文本清理以去除噪声，文本标注以创建更多特征，将基于文本的特征转换为预测器，使用梯度下降学习一个模型，并且最终进行模型调优。

问：

多义现象可以被定义为在文本对象中一个单词或短语的多种含义共存。下列哪一种方法可能是解决此问题的最好选择？

A）随机森林分类器

B）卷积神经网络

C）梯度爆炸

D）上述所有方法

答：

B

CNN 是文本分类问题中比较受欢迎的选择，因为它们把上下文的文本当作特征来考虑，这样可以解决多义问题。

问：

下列那种模型可以被用于文本相似度（document similarity）问题？

A）在语料中训练一个由词到向量（word 2 vector）的模型来对文本中呈现的上下文语境进行学习

B）训练一个词包模型（a bag of words model）来对文本中的词的发生率（occurrence）进行学习

C）创建一个文献检索词矩阵（document-term matrix）并且对每一个文本应用余弦相似性

D）上述所有方法均可

答：

D

word2vec 模型可在基于上下文语境的情况下用于测量文本相似度。词包模型（Bag Of Words）和文献检索词矩阵（document term matrix）可以在基于词条的情况下用来测量相似度。

问：

下列哪些是语料库的可能性特征？

1. 文本中词的总数

2. 布尔特征——文本中词的出现

3. 词的向量标注

4. 语音标注部分

5. 基本依赖性语法

6. 整个文本作为一个特征

A) 1

B) 12

C) 123

D) 1234

E) 12345

F) 123456

答：

E

除了全部文本作为特征这个选项，其余均可被用作文本分类特征，从而来对模型进行学习。

问：

当在文本数据中创建一个机器学习模型时，你创建了一个输入数据为 100K 的文献检索词矩阵（document-term matrix）。下列哪些纠正方法可以用来减少数据的维度——

1. 隐狄利克雷分布（Latent Dirichlet Allocation）

2. 潜在语义索引（Latent Semantic Indexing）

3. 关键词归一化（Keyword Normalization）

A）只有 1

B）2、3

C）1、3

D）1、2、3

答：

D

所有的这些方法都可用于减少数据维度。

问：

谷歌搜索特征——「Did you mean」，是不同方法相混合的结果。下列哪种方法可能是其组成部分？

1. 用协同过滤模型（Collaborative Filtering model）来检测相似用户表现（查询）

2. 在术语中检查 Levenshtein 距离的模型

3. 将句子译成多种语言

A）1

B）2

C）1、2

D）1、2、3

答：

C

协同过滤可以用于检测人们使用的是何种模式，Levenshtein 用来测量术语间的距离。

## **华为**

问：

**什么是向量空间模型**

答：

向量空间模型就是把对文本 的 内容经过处理后转换到向量空间 中的向量运算， 文档被表示为文档空间的向量，就可以通过计算向量之间的相似性来度量文档间的相似性

问：

在处理自然结构的新闻性句子的时候，哪种基于语法的文本句法分析方法可以用于名词短语检测、动词短语检测、主语检测和宾语检测。

A）部分语音标注

B）依存句法分析（Dependency Parsing）和选取句法分析（Constituency Parsing）

C）Skip Gram 和 N-Gram 提取

D）连续性词包

答：

B

依存句法分析和选取句法分析可从文本中提取这些关系。

问：

社交媒体平台是文本数据最直观的呈现形式。假设你有一个推特社交媒体完整语料库，你会如何创建一个建议标签的模型？

A）完成一个主题模型掌握语料库中最重要的词汇；

B）训练一袋 N-gram 模型捕捉顶尖的 n-gram：词汇和短语

C）训练一个词向量模型学习复制句子中的语境

D）以上所有

答：

D

上面所有的技术都可被用于提取语料库中最重要的词条。

问：

在从文本数据中提取语境时，你遇到两个不同的句子：The tank is full of soldiers. The tank is full of nitrogen。下面哪种措施可被用于句子中词意模糊的问题？

A）对比模糊词汇与近义词在词典上的定义

B）同指（Co-reference) 解决方案，使用先前句子中包含的正确词意解决模糊单词的含义。

C）使用句子的依存解析理解含义

答：

A

A 选项被称为 Lesk 算法，被用在词意模糊问题上，其他选择不对。

问：

协同过滤和基于内容的模型是两种流行的推荐引擎，在建立这样的算法中 NLP 扮演什么角色？

A 从文本中提取特征

B 测量特征相似度

C 为学习模型的向量空间编程特征

D 以上都是

答：

D

NLP 可用于文本数据相关的任何地方：特征提取、测量特征相似度、创造文本的向量特征。

问：

基于检索的模型和生成式模型是建立聊天机器人的两个主流技术，下面那个选项分别包含检索模型和生成式模型例子？

A. 基于辞典的学习和词向量模型

B. 基于规则的学习和序列到序列模型

C. 词向量和句子到向量模型

D. 循环神经网络和卷积神经网络

答：

B

选项 B 最佳诠释了基于检索的模型和生成式模型的例子。

问：

CRF（条件随机场）和 HMM（隐马尔可夫模型）之间的主要区别是什么？

A.CRF 是生成式的，而 HMM 是判别式模型；

B.CRF 是判别式模型，HMM 是生成式模型。

C.CRF 和 HMM 都是生成式模型；

D.CRF 和 HMM 都是判别式模型。

答：

B

# 2018年6月份

# 你是如何解决Ajax跨越问题—作者：XX项目经理

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 95% | 3星 | .Net | 跨域问题 | CORS/JSONP |

## 通常解法

在请求页面上使用Access-Control-AllowOigin标头，即CROS

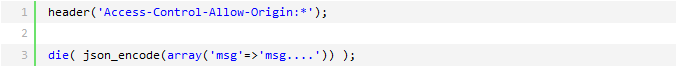
数使用如下标头可以接受全部网站请求：

heaer('Access-Control-Allow-Origin:\*')

使用如下标头可以接受指定网站请求：

header('Access-Control-Allow-Origin:http://www.abc.com')

所以，服务器写法：



Ajax写法和平常一样：



## 通用大牛级解法

把上面的解法说一遍………

另外也可以使用jsonp跨域请求。

服务器：



Ajax请求：



在ajax请求中，使用jsonp数据格式，所以将dataType参数设置为jsonp，平常，还加到了jsonp和jsonpCallback两个参数，结合代码，不难看出这两个参数作用：

jsonp的作用是设置服务器获取回调函数名称参数的下标参数，jsonpCallback的作用就是设置回调函数，相当于input标签中name和value，jsonp对应name，value对应jsonpCallback。

当然，对已一些简单的GET请求，并不需要回调函数处理，可以使用script、img、iframe 等html标签设置的src属性产生GET请求，这里不在讨论情况了。

## 解法对比及优缺点

JSONP的优缺点

JSONP的优点是：它不像XMLHttpRequest对象实现的Ajax请求那样受到同源策略的限制；它的兼容性更好，在更加古老的浏览器中都可以运行，不需要XMLHttpRequest或ActiveX的支持；并且在请求完毕后可以通过调用callback的方式回传结果。

JSONP的缺点则是：它只支持GET请求而不支持POST等其它类型的HTTP请求；它只支持跨域HTTP请求这种情况，不能解决不同域的两个页面之间如何进行JavaScript调用的问题

1. CORS与JSONP相比，无疑更为先进、方便和可靠。

* JSONP只能实现GET请求，而CORS支持所有类型的HTTP请求。
* 使用CORS，开发者可以使用普通的XMLHttpRequest发起请求和获得数据，比起JSONP有更好的错误处理。
* JSONP主要被老的浏览器支持，它们往往不支持CORS，而绝大多数现代浏览器都已经支持了CORS）。

**综上所述**：使用CORS要比JSONP要好的多，解法一相对比较好，但如果了解JSONP，也可以在客观上加分，尤其当你说出了JSONP的优缺点

## 延伸及扩展问题回答参考

**问题**： 针对跨域问题，服务（WebAPI、WebService）的安全性如何去做，你有几种解决方案

**解答**： 1、TOKEN+签名认证，使用json web token

2、基于Owin OAuth，进行授权

3、使用签名，给每个http请求添加一个签名，服务端来验证签名的合法性，如果签名合法则执行响应的操作，如果签名非法则直接拒绝请求。

（签名算法一般都使用Hash散列算法，常用的有MD5，SHA系列算法。）

## 项目中体现经验的点

做过跨越，主要体现接触过客户端服务端分离开发的经验，适合大多中级工程师必须掌握的知识点

## 论坛参考

http://www.cnblogs.com/pandang/p/5341250.html

http://www.cnblogs.com/landeanfen/p/5177176.html

http://www.cnblogs.com/JerryTian/p/4194900.html

## XX项目经理音频文件XX时长

# 什么是内存泄漏？

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 70% | 4星 | .Net | C# | 内存流 |

## 通常解法

，**内存泄漏**指由于疏忽或错误造成程序未能释放已经不再使用的[内存](https://zh.wikipedia.org/wiki/%E5%86%85%E5%AD%98)。内存泄漏并非指内存在物理上的消失，而是应用程序分配某段内存后，由于设计错误，导致在释放该段内存之前就失去了对该段内存的控制，从而造成了内存的浪费。

，重起计算机可以解决，但是也有可能再次发生内存泄露，内存泄露和硬件没有关系，它是由软件设计缺陷引起的

从用户使用程序的角度来看，内存泄漏本身不会产生什么危害，作为一般的用户，根本感觉不到内存泄漏的存在。真正有危害的是内存泄漏的堆积，这会最终 消耗尽系统所有的内存

一般我们常说的内存泄漏是指堆内存的泄漏。堆内存是指程序从堆中分配的

内存泄漏可以分为4类：   
  
1. 常发性内存泄漏。发生内存泄漏的代码会被多次执行到，每次被执行的时候都会导致一块内存泄漏。   
2. 偶发性内存泄漏。发生内存泄漏的代码只有在某些特定环境或操作过程下才会发生。常发性和偶发性是相对的。对于特定的环境，偶发性的也许就变成了常发性的。所以测试环境和测试方法对检测内存泄漏至关重要。   
3. 一次性内存泄漏。发生内存泄漏的代码只会被执行一次，或者由于算法上的缺陷，导致总会有一块仅且一块内存发生泄漏。比如，在类的构造函数中分配内存，在析构函数中却没有释放该内存，所以内存泄漏只会发生一次。   
4. 隐式内存泄漏。程序在运行过程中不停的分配内存，但是直到结束的时候才释放内存。严格的说这里并没有发生内存泄漏，因为最终程序释放了所有申请的内存。但是对于一个服务器程序，需要运行几天，几周甚至几个月，不及时释放内存也可能导致最终耗尽系统的所有内存。所以，我们称这类内存泄漏为隐式内存泄漏

## 通用大牛级解法

加上通用解法后

当怀疑应用程序发生内存泄漏时，必须首先确定是哪种类型的内存发生泄漏，以便您将调试工作的重点放在合适的区域。使用 PerfMon 来检查用于应用程序的下列性能计数器：Process/Private Bytes、.NET CLR Memory/# Bytes in All Heaps 和 .NET CLR LocksAndThreads/# of current logical Threads。Process/Private Bytes 计数器用于报告系统中专门为某一进程分配而无法与其他进程共享的所有内存。.NET CLR Memory/# Bytes in All Heaps 计数器报告第 0 代、第 1 代、第 2 代和大型对象堆的合计大小。.NET CLR LocksAndThreads/# of current logical Threads 计数器报告 AppDomain 中逻辑线程的数量。如果应用程序的逻辑线程计数出现意想不到的增大，则表明线程堆栈发生泄漏。如果 Private Bytes 增大，而 # Bytes in All Heaps 保持不变，则表明非托管内存发生泄漏。如果上述两个计数器均有所增加，则表明托管堆中的内存消耗在增长。

大多数时候，主要关注Process 里的以下几个指标就能得出结论，如果这些量整体来看是持续上升的，基本可以判断是有泄露情况存在的。

　　A.Handle Count

　　B.Thread Count

　　C.Private Bytes

　　D.Virtual Bytes

　　E.Working Set

　　F.另外.NET CLR Memory下的Bytes in all heeps也是比较关注的。

　　通过观察，如果发现这些参数是在一个区间内震荡的，应该是没有大的问题，但如果是一个持续上涨的状态，那就得注意，很可能存在内存泄露

**内存泄漏检测工具：**

1. SciTech Software AB .NET Memory Profiler

2. YourKit .NET & Java Profiler

3. AutomatedQA AQTime

3.使用[**LoadRunner**](http://blog.csdn.net/fhzh520/article/details/7872305)，

4.使用 .Net Memory Profiler 工具

5.在单元测试时，在代码中检测，如.net 下   使用Console.WriteLine("Total memory: {0:###,###,###,##0} bytes", GC.GetTotalMemory(true));代码可以查看当前使用的内存。

**常见的内存泄漏场景及解决方案：**

在.NET中，虽然CLR的GC垃圾回收器帮我们自动回收托管堆对象，释放内存，最大程度避免了"内存泄漏"(应用程序所占用的内存没有得到及时释放)，但.NET应用程序"内存泄漏"的问题还是会存在，如果不加以注意，"内存泄漏"时有发生。

*有关流以及Reader或Writer引起的内存泄漏*

比如，把文件读取到流中：

public static string ReadFile()

{

var filePath = @"硬盘地址";

var sr = new StreamReader(filePath);

return sr.ReadToEnd();

}

以上，StreamReader在读取数据后没有解释销毁，存在"内存泄漏"。正确的做法是在使用完后及时关闭。

public static string ReadFile()

{

var filePath = @"硬盘地址";

using(var sr = new StreamReader(filePath))

{

return sr.ReadToEnd();

}

}

或者

public static string ReadFile()

{

var filePath = @"硬盘地址";

var sr = new StreamReader(filePath);

var result = sr.ReadToEnd();

sr.Close();

return result;

}

以上，需要我们注意的是：当通过某种流的构造函数创建的对象实例，注意及时关闭。

有时候，通过某个方法返回某种流的对象实例，也会忘记关闭。比如以下：

//创建字节数组

var data = new byte[1024];

var client = new TcpClient();

//从TCP实例方法返回流

var stream = client.GetStream();

//把流读到字节数组中

int bytesLength = stream.Read(data, 0, data.Length);

//字节数组转换城字符串

var result = System.Text.Encoding.ASCII.GetString(data, 0, bytesLength);

正确的写法应该是：

//创建字节数组

var data = new byte[1024];

var client = new TcpClient();

//从TCP实例方法返回流

var stream = client.GetStream();

//把流读到字节数组中

int bytesLength = stream.Read(data, 0, data.Length);

stream.Close();

//字节数组转换城字符串

var result = System.Text.Encoding.ASCII.GetString(data, 0, bytesLength);

同理，其它与流有关的类，我们也需要注意在用完后及时关闭：

● FileStream  
● MemoryStream  
● StreamReader  
● TextWriter  
......

*静态引用引起的内存泄漏*

对于静态实例来说，除非应用程序关闭，对应的内存一直得不到释放。比如有如下遵循"Siingleton"模式的类(没考虑线程安全)。

public class MySingletonClass

{

private static MySingletonClass myInstance;

private static List<IAmBig> bigObjects = new List<IAmBig>();

private MySingletonClass(){}

public static MySingletonClass MyInstance

{

get

{

if(myInstance == null)

{

myInstance = new MySingletonClass();

}

return myInstance;

}

}

public static IAmBig CreateBigObject()

{

var bigObject = new IAmBig();

bigobject.AllocateMemory(4096);

bigObjects.add(bigObject);

return bigObject;

}

}

public class IAmBig

{

}

以上，每次调用CreateBigObject静态方法，都往List<IAmBig>类型集合中添加，由于MySingletonClass静态类实例一直存在于应用程序的生命周期，再加上GC不会释放bigObjects这个集合对象实例，于是就出现了"内存泄漏"。解决办法是避免让静态实例引用其它实例对象，避免出现静态实例的链式引用。

*委托引起的内存泄漏*

比如有2个委托形成的委托链。

var objectOne = new ObjectOne();

var objectTwo = new ObjectTwo();

objectOne.StateChanged += objectTwo.StateChangedEventHandler;

objectTwo.Dispose();

以上，把objectTwo的委托注册给了objectOne，这样objectOne和objectTwo有依赖关系，形成了依赖链。只有当objectOne被释放，才能释放objectTwo。如果objectOne恰巧是全局静态实例，那在应用程序的生命周期内，objectTwo一直得不到内存释放，造成了"内存泄漏"。

解决办法是在调用objectTwo的Dispose方法之前，先解开两者的依赖关系。修改如下：

var objectOne = new ObjectOne();

var objectTwo = new ObjectTwo();

objectOne.StateChanged += objectTwo.StateChangedEventHandler;

......

objectOne.StateChanged -= objectTwo.StateChangedEventHandler;

objectTwo.Dispose()

*非托管资源引起的内存泄漏*

public class MyUnManagedExample

{

public void Allocate()

{

IntPtr pointer = Marshal.AllocHGlobal(1024);

}

}

对于创建的非托管类型的实例ponter，需要显式释放。

Marshal.FreeGlobal(pointer);

*实现IDisposable接口的类引起的内存泄漏*

所有实现IDisposable接口的类都有一个Dispose方法，如果忘记调用，就造成"内存泄漏"。

**如何防止内存泄露**

**(1) Dispose()的使用**

　　如果使用的对象提供Dispose（）方法，那么当你使用完毕或在必要的地方（比如Exception）调用该方法，特别是对非托管对象,一定要加以调 用，以达到防止泄露的目的。另外很多时候程序提供对Dispose()的扩展，比如Form，在这个扩展的Dispose方法中你可以把大对象的引用什么 的在退出前释放。

　　对于DB连接，COM组件（比如OLE组件）等必须调用其提供的Dispose方法，没有的话最好自己写一个。

**(2) using的使用**

using除了引用Dll的功用外，还可以限制对象的适用范围，当超出这个界限后对象自动释放，比如

using语句的用途

定义一个范围，将在此范围之外释放一个或多个对象。

可以在 using 语句中声明对象：  
using (Font font1 = new Font("Arial", 10.0f))

{  
   // use font1

}

或者在 using 语句之前声明对象：

Font font2 = new Font("Arial", 10.0f);

using (font2)

{

// use font2

}

可以有多个对象与 using 语句一起使用，但是必须在 using 语句内部声明这些对象：  
using (Font font3 = new Font("Arial", 10.0f),font4 = new Font("Arial", 10.0f))

{

// Use font3 and font4.

}

**(3) 事件的卸载**

　　这个不是必须的，推荐这样做。之前注册了的事件，关闭画面时应该手动注销，有利于GC回收资源。

**(4) API的调用**

　　一般的使用API了就意味着使用了非托管资源，需要根据情况手动释放所占资源，特别是在处理大对象时。 4.5继承 IDisposable实现自己内存释放接口　Net 如何继承IDisposable接口，实现自己的Dispose()函数

**(5)弱引用（WeakReference ）**

　　通常情况下，一个实例如果被其他实例引用了，那么他就不会被GC回收，而弱引用的意思是，如果一个实例没有被其他实例引用（真实引用），而仅仅是被弱引 用，那么他就会被GC回收。

**(6)析构函数(Finalize())**

　　使用了非托管资源的时候，可以自定义析构函数使得对象结束时释放所占资源；

　　对仅使用托管资源的对象，应尽可能使用它自身的Dispose方法，一般不推荐自定义析构函数。

根据普遍意义上的内存泄漏定义，大多数的.NET内存对象在不再被使用后都会有短暂的一段时间的内存泄漏，因为要等待下一个GC时才有可能会被释放。但这种情况并不会对系统造成大的危害。

## 解法对比及优缺点

通常解法，仅就问题本身做简要解答，未能够对问题做详尽分析及解决。通用级解法，详细阐述产生内存泄漏的原因，并通过工具诊断，排除内存泄漏问题，针对产生内存泄漏的问题给出有效的解决方案进行防止。

## 延伸及扩展问题回答参考

**问题**： 针对内存泄漏的问题，对CLR的内存分配机和垃圾回收机制

**解答**： 1、NET Framework 的垃圾回收器管理应用程序的内存分配和释放。每次使用 **new**运算符创建对象时，运行库都从托管堆为该对象分配内存。只要托管堆中有地址空间可用，运行库就会继续为新对象分配空间。

...

object obj = new object();

但是，内存不是无限大的。

public void FillMemory()

{

ArrayList memory = new ArrayList();

    // 输出填充前所占内存大小

Console.WriteLine("used memory:" + GC.GetTotalMemory(false));

for (int i = 0; i < 100000; i++)

{

memory.Add(new object());

}

    // 输出填充后所占的内存大小

Console.WriteLine("used memory:" + GC.GetTotalMemory(false));

}

最终，垃圾回收器必须执行回收以释放一些内存。垃圾回收器优化引擎根据正在进行的分配情况确定执行回收的最佳时间。当垃圾回收器执行回收时，它检查托管堆中不再被应用程序使用的对象并执行必要的操作来回收它们占用的内存。

2、**CLR的内存回收机制**

一般我们在程序中创建的对象大部分都是托管对象，可依靠GC自动进行内存的回收，但是对于封装了非托管资源的对象，就需要我们显式重载object.Finalize()接口来实现非托管资源的释放。

using System;

using System.IO;

public class Foo

{

private SomeComObject \_com;

public Foo()

{

\_com = new SomeComObject();

}

// some other operation here...

~Foo()

{

// release the unmanaged resource

\_com.Close();

}

}

**\* 析构函数会在编译时会被翻译为protected void Finalize()，这是C#的析构函数的语法**

GC在回收对象之前会调用Finalize()来实现非托管资源的释放，不过按照MSDN的说法，Finalize()会导致性能的降低。

更加建议实现Sytem.IDisposable.Dispose()接口，用来实现对非托管资源的释放，这也是.Net Framework中常见的设计模式。那该怎么实现Dispose呢？

1、首先，Dispose接口应该释放自身对象所占用的资源，还应该调用基类的Dispose方法，释放基类部分所占用的资源。

public void Dispose()

{

// do something to release my unmanaged resource

ReleaseMyResource();

base.Dispose();

}

2、前面说过Finalize()会导致性能问题，那么在执行Dispose以后就应该告诉GC不用在调用Finalize()了

public void Dispose()

{

// do something to release my unmanaged resource

ReleaseMyResource();

base.Dispose();

// tell gc not to call Finalize()

GC.SuppressFinalize(this);

}

当然我们完全可以定义一个MySpace.IClosable.Close()，通过实现这个接口来进行非托管资源的释放，不过这实在没有必要。

## 项目中体现经验的点

主要体现在文件流读取及Office PIA Com组件的调用经验，适合大多中级工程师必须掌握的知识点

## 论坛参考

<http://www.360doc.com/content/17/0925/20/47820059_690133713.shtml>

<http://blog.csdn.net/changtianshuiyue/article/details/52443821>

<https://www.codeproject.com/articles/42721/best-practices-no-detecting-net-application-memo>

<https://www.codeproject.com/articles/19490/memory-leak-detection-in-net>

<http://www.cnblogs.com/sufei/archive/2010/01/18/1650257.html>

<http://www.cnblogs.com/qianyz/archive/2010/10/07/1844920.html>

<http://www.cnblogs.com/kissdodog/p/3672460.html>

<http://www.cnblogs.com/rohelm/archive/2012/04/20/2458704.html>

<https://www.zhihu.com/question/39047173>

<http://blog.csdn.net/fhzh520/article/details/7872305>

<http://blog.csdn.net/jelink/article/details/1338810>

# 什么是事务？什么是锁？

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 90% | 4星 | .Net | 事务和锁 | 数据库 |

## 通常解法

**事务**

所谓事务是用户定义的一个数据库操作系列，这些操作要么全部执行，要么全部不执行，是一个不可分割的工作单位。例如在关系数据库中，一个事务可以是一条sql语句、一组sql语句或整个程序。

给个例子：

小IT在网上购物，其付款过程至少包括以下几步数据库操作：

1. 更新客户所购商品的库存信息；
2. 生成订单并且保存到数据库；
3. 更新用户相关信息，例如购物数量等；

正常情况下，操作顺利进行，最终交易成功，那么与交易相关的所有数据库信息也成功更新。但是，如果在这一系列过程中任何一个环节出了差错，例如在更新商品库存信息时发生异常、该顾客银行帐户存款不足等，都将导致交易失败。一旦交易失败，数据库中所有信息都必须保持交易前的状态不变，比如最后一步更新用户信息时失败而导致交易失败，那么必须保证这笔失败的交易不影响数据库的状态--库存信息没有被更新、用户也没有付款，订单也没有生成。否则，数据库的信息将会一片混乱而不可预测。

数据库事务正是用来保证这种情况下交易的平稳性和可预测性的技术。

**锁**

数据库和操作系统一样，是一个多用户使用的共享资源。当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性。加锁是实现数据库并 发控制的一个非常重要的技术。在实际应用中经常会遇到的与锁相关的异常情况，当两个事务需要一组有冲突的锁，而不能将事务继续下去的话，就会出现死锁，严重影响应用的正常执行。   
在数据库中有两种基本的锁类型：排它锁（Exclusive Locks，即X锁）和共享锁（Share Locks，即S锁）。当数据对象被加上排它锁时，其他的事务不能对它读取和修改。加了共享锁的数据对象可以被其他事务读取，但不能修改。数据库利用这两 种基本的锁类型来对数据库的事务进行并发控制。

## 通用大牛级解法

把上面的解法说一遍………

**事务的ACID特性**

**A(Atomicity)原子性**

事务必须是原子工作单元；对于其数据修改，要么全都执行，要么全都不执行。通常，与某个事务关联的操作具有共同的目标，并且是相互依赖的。如果系统只执行这些操作的一个子集，则可能会破坏事务的总体目标。原子性消除了系统处理操作子集的可能性。

**C(Consistency)一致性**

事务在完成时，必须使所有的数据都保持一致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构（如 B 树索引或双向链表）都必须是正确的。某些维护一致性的责任由应用程序开发人员承担，他们必须确保应用程序已强制所有已知的完整性约束。例如，当开发用于转帐的应用程序时，应避免在转帐过程中任意移动小数点。

**I(Isolation)隔离性**

指的是在并发环境中，当不同的事务同时操纵相同的数据时，每个事务都有各自的完整数据空间。由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。事务查看数据更新时，数据所处的状态要么是另一事务修改它之前的状态，要么是另一事务修改它之后的状态，**事务不会查看到中间状态的数据**。

**D(Durability)持久性**

指的是只要事务成功结束，它对数据库所做的更新就必须永久保存下来。即使发生系统崩溃，重新启动数据库系统后，数据库还能恢复到事务成功结束时的状态。

事务的ACID特性是由关系数据库管理系统（RDBMS，数据库系统）来实现的。数据库管理系统采用日志来保证事务的原子性、一致性和持久性。日志记录了事务对数据库所做的更新，如果某个事务在执行过程中发生错误，就可以根据日志，撤销事务对数据库已做的更新，使数据库退回到执行事务前的初始状态。数据库管理系统采用锁机制来实现事务的隔离性。当多个事务同时更新数据库中相同的数据时，只允许持有锁的事务更新数据，其他事务必须等待，直到前一个事务释放了锁，其他事务才有机会更新该数据。

完整的事务结构

[复制代码](javascript:void(0);)

BEGIN a transaction;//设置事务的起始点

COMMIT a transaction;//提交事务，使事务提交的数据成为持久不可更改的部分

ROLLBACK a transaction;//撤销一个事务，回滚，使之成为事务开始前的状态

SAVE a transaction;//建立标签，用作部分回滚，使之恢复到标签初的状态

[复制代码](javascript:void(0);)

给个例子：

下面整个示例，先来建一张表（使用SqlServer）如下：



[复制代码](javascript:void(0);)

BEGIN TRAN Tran\_Money --开始事务

DECLARE @tran\_error int;

SET @tran\_error = **0**;

BEGIN TRY

UPDATE tb\_Money SET MyMoney = MyMoney - **30** WHERE Name = '刘备';

SET @tran\_error = @tran\_error + **@@ERROR**;

--测试出错代码，看看刘备的钱减少，关羽的钱是否会增加

--SET @tran\_error = 1;

UPDATE tb\_Money SET MyMoney = MyMoney + **30** WHERE Name = '关羽';

SET @tran\_error = @tran\_error + **@@ERROR**;

END TRY

BEGIN CATCH

PRINT '出现异常，错误编号：' + convert(varchar,error\_number()) + ',错误消息：' + error\_message()

SET @tran\_error = @tran\_error + **1**

END CATCH

IF(@tran\_error > **0**)

BEGIN

--执行出错，回滚事务

ROLLBACK TRAN;

PRINT '转账失败，取消交易!';

END

ELSE

BEGIN

--没有异常，提交事务

COMMIT TRAN;

PRINT '转账成功!';

END

[复制代码](javascript:void(0);)

详细参照[SQL Server 事务语法](http://www.cnblogs.com/kissdodog/p/3169788.html)

**冲突问题**

**1、脏读**

某个事务读取的数据是另一个事务正在处理的数据。而另一个事务可能会回滚，造成第一个事务读取的数据是错误的。

**2、不可重复读**

在一个事务里两次读入数据，但另一个事务已经更改了第一个事务涉及到的数据，造成第一个事务读入旧数据。

**3、幻读**

幻读是指当事务不是独立执行时发生的一种现象。例如第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。

**4、更新丢失**

多个事务同时读取某一数据，一个事务成功处理好了数据，被另一个事务写回原值，造成第一个事务更新丢失。

事务隔离级别

数据库事务的隔离级别有4个，由低到高依次为Read uncommitted、Read committed、Repeatable read、Serializable，这四个级别可以逐个解决脏读、不可重复读、幻读这几类问题。

**Read UnCommitted - 读未提交**

Read UnCommitted事务可以读取事务已修改，但未提交的的记录。

Read UnCommitted事务会产生脏读（Dirty Read）。

Read UnCommitted事务与select语句加nolock的效果一样，它是所有隔离级别中限制最少的。

例如：

公司发工资了，领导把5000元打到singo的账号上，但是该事务并未提交，而singo正好去查看账户，发现工资已经到账，是5000元整，非常高兴。可是不幸的是，领导发现发给singo的工资金额不对，是2000元，于是迅速回滚了事务，修改金额后，将事务提交，最后singo实际的工资只有2000元，singo空欢喜一场。

出现上述情况，即我们所说的脏读，两个并发的事务，“事务A：领导给singo发工资”、“事务B：singo查询工资账户”，事务B读取了事务A尚未提交的数据。

当隔离级别设置为Read uncommitted时，就可能出现脏读，如何避免脏读，请看下一个隔离级别。

Read Committed 读提交

一旦创建共享锁的语句执行完成，该锁顶便释放。

Read Committed是SQL Server的预设隔离等级。

Read Committed只可以防止脏读。

[复制代码](javascript:void(0);)

--先创建表:

CREATE TABLE tb(id int,val int)

INSERT tb VALUES(**1**,**10**)

INSERT tb VALUES(**2**,**20**)

然后在连接1中,执行:

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

SELECT \* FROM tb; --这个SELECT结束后,就会释放掉共享锁

WAITFOR DELAY '00:00:05' --模拟事务处理,等待5秒

SELECT \* FROM tb; --再次SELECT tb表

ROLLBACK --回滚事务

在连接2中,执行

UPDATE tb SET

val = val + **10**

WHERE id = **2**;

--------

回到连接1中.可以看到.两次SELECT的结果是不同的.

因为在默认的READ COMMITTED隔离级别下,SELECT完了.就会马上释放掉共享锁.

[复制代码](javascript:void(0);)

singo拿着工资卡去消费，系统读取到卡里确实有2000元，而此时她的老婆也正好在网上转账，把singo工资卡的2000元转到另一账户，并在 singo之前提交了事务，当singo扣款时，系统检查到singo的工资卡已经没有钱，扣款失败，singo十分纳闷，明明卡里有钱，为何......

出现上述情况，即我们所说的不可重复读 ，两个并发的事务，“事务A：singo消费”、“事务B：singo的老婆网上转账”，事务A事先读取了数据，事务B紧接了更新了数据，并提交了事务，而事务A再次读取该数据时，数据已经发生了改变。

当隔离级别设置为Read committed 时，避免了脏读，但是可能会造成不可重复读。

大多数数据库的默认级别就是Read committed，比如Sql Server , Oracle。如何解决不可重复读这一问题，请看下一个隔离级别。

**Repeatable Read -重复读**

REPEATABLE READ事务不会产生脏读，并且在事务完成之前，任何其它事务都不能修改目前事务已读取的记录。

其它事务仍可以插入新记录，但必须符合当前事务的搜索条件——这意味着当前事务重新查询记录时，会产生幻读（Phantom Read）。

当隔离级别设置为Repeatable read 时，可以避免不可重复读。当singo拿着工资卡去消费时，一旦系统开始读取工资卡信息（即事务开始），singo的老婆就不可能对该记录进行修改，也就是singo的老婆不能在此时转账。

虽然Repeatable read避免了不可重复读，但还有可能出现幻读 。

singo的老婆工作在银行部门，她时常通过银行内部系统查看singo的信用卡消费记录。有一天，她正在查询到singo当月信用卡的总消费金额 （select sum(amount) from transaction where month = 本月）为80元，而singo此时正好在外面胡吃海塞后在收银台买单，消费1000元，即新增了一条1000元的消费记录（insert transaction ... ），并提交了事务，随后singo的老婆将singo当月信用卡消费的明细打印到A4纸上，却发现消费总额为1080元，singo的老婆很诧异，以为出 现了幻觉，幻读就这样产生了。

注：Mysql的默认隔离级别就是Repeatable read。

**Serializable – 序列化**

SERIALIZABLE可以防止除更新丢失外所有的一致性问题,即:

1.语句无法读取其它事务已修改但未提交的记录。

2.在当前事务完成之前，其它事务不能修改目前事务已读取的记录。

3.在当前事务完成之前，其它事务所插入的新记录，其索引键值不能在当前事务的任何语句所读取的索引键范围中。

**SnapShot**

Snapshot事务中任何语句所读取的记录，都是事务启动时的数据。

这相当于事务启动时，数据库为事务生成了一份专用“快照”。在当前事务中看到不其它事务在当前事务启动之后所进行的数据修改。

Snapshot事务不会读取记录时要求锁定，读取记录的Snapshot事务不会锁住其它事务写入记录，写入记录的事务也不会锁住Snapshot事务读取数据。

**锁模式**

**1、共享锁**

共享锁（S 锁）允许并发事务在封闭式并发控制下读取 (SELECT)资源。有关详细信息，请参阅[并发控制的类型](http://msdn.microsoft.com/zh-cn/library/ms189132.aspx)（悲观锁和乐观锁）。资源上存在共享锁（S锁）时，任何其他事务都不能修改数据。读取操作一完成，就立即释放资源上的共享锁（S锁），除非将事务隔离级别设置为可重复读或更高级别，或者在事务持续时间内用锁定提示保留共享锁（S锁）。

**2、更新锁（U锁）**

更新锁在共享锁和排他锁的结合。更新锁意味着在做一个更新时，一个共享锁在扫描完成符合条件的数据后可能会转化成排他锁。

这里面有两个步骤：

1) 扫描获取Where条件时。这部分是一个更新查询，此时是一个更新锁。

2) 如果将执行写入更新。此时该锁升级到排他锁。否则，该锁转变成共享锁。

更新锁可以防止常见的死锁。

**3、排他锁**

排他锁（X 锁）可以防止并发事务对资源进行访问。排他锁不与其他任何锁兼容。使用排他锁（X锁）时，任何其他事务都无法修改数据；仅在使用 NOLOCK提示或未提交读隔离级别时才会进行读取操作。

**悲观锁**

悲观锁是指假设并发更新冲突会发生，所以不管冲突是否真的发生，都会使用锁机制。  
悲观锁会完成以下功能：锁住读取的记录，防止其它事务读取和更新这些记录。其它事务会一直阻塞，直到这个事务结束.  
悲观锁是在使用了数据库的事务隔离功能的基础上，独享占用的资源，以此保证读取数据一致性，避免修改丢失。

悲观锁可以使用Repeatable Read事务，它完全满足悲观锁的要求。

**乐观锁**

乐观锁不会锁住任何东西，也就是说，它不依赖数据库的事务机制，乐观锁完全是应用系统层面的东西。

如果使用乐观锁，那么数据库就必须加版本字段，否则就只能比较所有字段，但因为浮点类型不能比较，所以实际上没有版本字段是不可行的。

## 解法对比及优缺点

常见解法，简要说明事务及死锁的概念及案例应用。通用解法，详尽举例说明死锁的发生场景及产生原因，针对不同场景下事务的应用及各种锁类型应用做详尽阐述。

## 延伸及扩展问题回答参考

**问题**： 死锁的几种情况及死锁避免

解答

**死锁的几种情况：**

死锁的第一种情况   
一个用户A 访问表A(锁住了表A),然后又访问表B；另一个用户B 访问表B(锁住了表B)，然后企图访问表A；这时用户A由于用户B已经锁住表B，它必须等待用户B释放表B才能继续，同样用户B要等用户A释放表A才能继续，这就死锁就产生了。

解决方法：   
这种死锁比较常见，是由于程序的BUG产生的，除了调整的程序的逻辑没有其它的办法。仔细分析程序的逻辑，对于数据库的多表操作时，尽量按照相同的顺序进 行处理，尽量避免同时锁定两个资源，如操作A和B两张表时，总是按先A后B的顺序处理， 必须同时锁定两个资源时，要保证在任何时刻都应该按照相同的顺序来锁定资源。

死锁的第二种情况   
用户A查询一条纪录，然后修改该条纪录；这时用户B修改该条纪录，这时用户A的事务里锁的性质由查询的共享锁企图上升到独占锁，而用户B里的独占锁由于A 有共享锁存在所以必须等A释放掉共享锁，而A由于B的独占锁而无法上升的独占锁也就不可能释放共享锁，于是出现了死锁。这种死锁比较隐蔽，但在稍大点的项目中经常发生。如在某项目中，页面上的按钮点击后，没有使按钮立刻失效，使得用户会多次快速点击同一按钮，这样同一段代码对数据库同一条记录进行多次操作，很容易就出现这种死锁的情况。

解决方法：   
1、对于按钮等控件，点击后使其立刻失效，不让用户重复点击，避免对同时对同一条记录操作。   
2、使用乐观锁进行控制。乐观锁大多是基于数据版本（Version）记录机制实现。即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是 通过为数据库表增加一个“version”字段来实现。读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数 据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。乐观锁机制避免了长事务中的数据库加锁开销（用户A和用户B操作过程中，都没有对数据库数据加锁），大大提升了大并发量下的系统整体性能表现。Hibernate 在其数据访问引擎中内置了乐观锁实现。需要注意的是，由于乐观锁机制是在我们的系统中实现，来自外部系统的用户更新操作不受我们系统的控制，因此可能会造 成脏数据被更新到数据库中。   
3、使用悲观锁进行控制。悲观锁大多数情况下依靠数据库的锁机制实现，如Oracle的Select … for update语句，以保证操作最大程度的独占性。但随之而来的就是数据库性能的大量开销，特别是对长事务而言，这样的开销往往无法承受。如一个金融系统， 当某个操作员读取用户的数据，并在读出的用户数据的基础上进行修改时（如更改用户账户余额），如果采用悲观锁机制，也就意味着整个操作过程中（从操作员读 出数据、开始修改直至提交修改结果的全过程，甚至还包括操作员中途去煮咖啡的时间），数据库记录始终处于加锁状态，可以想见，如果面对成百上千个并发，这样的情况将导致灾难性的后果。所以，采用悲观锁进行控制时一定要考虑清楚。

死锁的第三种情况   
如果在事务中执行了一条不满足条件的update语句，则执行全表扫描，把行级锁上升为表级锁，多个这样的事务执行后，就很容易产生死锁和阻塞。类似的情 况还有当表中的数据量非常庞大而索引建的过少或不合适的时候，使得经常发生全表扫描，最终应用系统会越来越慢，最终发生阻塞或死锁。   
解决方法：   
SQL语句中不要使用太复杂的关联多表的查询；使用“执行计划”对SQL语句进行分析，对于有全表扫描的SQL语句，建立相应的索引进行优化。

总体上来说，产生内存溢出与锁表都是由于代码写的不好造成的，因此提高代码的质量是最根本的解决办法。有的人认为先把功能实现，有BUG时再在测试阶段进 行修正，这种想法是错误的。正如一件产品的质量是在生产制造的过程中决定的，而不是质量检测时决定的，软件的质量在设计与编码阶段就已经决定了，测试只是对软件质量的一个验证，因为测试不可能找出软件中所有的BUG。

**如何避免死锁**

1 使用事务时，尽量缩短事务的逻辑处理过程，及早提交或回滚事务；   
2 设置死锁超时参数为合理范围，如：3分钟-10分种；超过时间，自动放弃本次操作，避免进程悬挂；   
3 所有的SP都要有错误处理（通过@error）   
4 一般不要修改SQL SERVER事务的默认级别。不推荐强行加锁   
5 优化程序，检查并避免死锁现象出现；   
1）合理安排表访问顺序   
2）在事务中尽量避免用户干预，尽量使一个事务处理的任务少些。   
3）采用脏读技术。脏读由于不对被访问的表加锁，而避免了锁冲突。在客户机/服务器应用环境中，有些事务往往不允许读脏数据，但在特定的条件下，我们可以用脏读。   
4）数据访问时域离散法。数据访问时域离散法是指在客户机/服务器结构中，采取各种控制手段控制对数据库或数据库中的对象访问时间段。主要通过以下方式实 现: 合理安排后台事务的执行时间，采用工作流对后台事务进行统一管理。工作流在管理任务时，一方面限制同一类任务的线程数（往往限制为1个），防止资源过多占 用; 另一方面合理安排不同任务执行时序、时间，尽量避免多个后台任务同时执行，另外，避免在前台交易高峰时间运行后台任务   
5）数据存储空间离散法。数据存储空间离散法是指采取各种手段，将逻辑上在一个表中的数据分散到若干离散的空间上去，以便改善对表的访问性能。主要通过以下方法实现: 第一，将大表按行或列分解为若干小表; 第二，按不同的用户群分解。   
6）使用尽可能低的隔离性级别。隔离性级别是指为保证数据库数据的完整性和一致性而使多用户事务隔离的程度，SQL92定义了4种隔离性级别：未提交读、 提交读、可重复读和可串行。如果选择过高的隔离性级别，如可串行，虽然系统可以因实现更好隔离性而更大程度上保证数据的完整性和一致性，但各事务间冲突而死锁的机会大大增加，大大影响了系统性能。   
7）使用Bound Connections。Bound connections 允许两个或多个事务连接共享事务和锁，而且任何一个事务连接要申请锁如同另外一个事务要申请锁一样，因此可以允许这些事务共享数据而不会有加锁的冲突。   
8）考虑使用乐观锁定或使事务首先获得一个独占锁定。

）

## 项目中体现经验的点

针对多任务并行查询处理，解除死锁，具体语句参照**select \* from t2 WITH(NOLOCK) where UserID = 2   
select \* from t2 WITH(READPAST) where UserID = 2**，

简单来说：   
NOLOCK 可能把没有提交事务的数据也显示出来．   
READPAST 会把被锁住的行不显示出来

适合大多中级工程师必须掌握的知识点

## 论坛参考

<http://www.cnblogs.com/zgx/archive/2009/04/28/1445292.html>

<http://www.cnblogs.com/tgycoder/p/5410537.html>

# 如何处理几十万条并发数据？

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 80% | 5星 | .Net | 高并发 | 数据库 |

## 通常解法

主要三部分设计及配置：

第一个部分是由Web服务器和应用服务器构成的负载均衡区。该区域的主要目标是分散用户的访问量，平衡各服务器的压力，提高各服务器的资源利用率，

第二个部分是数据存储区。包括了NoSql数据库和关系型数据库，NoSql数据库主要目的是实现快速缓存，通过主从结构提高处理能力，NoSql数据库可以存储需要经常访问的数据，可以选择redis、Memcache、berkelydb等。关系型数据库主要存储交易和行为数据，通过读写分离和业务拆分提高数据库的负载能力。同时，数据在写入关系型数据库时候同时更新读库和Nosql数据库，以保证数据的一致性。数据一致性符合CAP原理。同时数据库存储的大量交易、行为数据可以通过ETL的形式提供给数据仓库作为数据分析和报表使用

第三个部分是数据采集区。包括了多并发多线程的采集服务器集群，以提高数据更新的实时性。数据采集后存入分布式数据库，因存在大量非结构化数据，所以可以采用Monogodb、Hbase等文档或列表型数据库存储海量数据。Monogodb内置Map-reduce处理，Hbase可以结合Hadoop做Map-reduce大数据分析。生成的结果可以同步到数据存储区，以便于用户的查询。

对数据库级别案例如下：

用存储过程，Web Service，异步存取。

有关并发控制，推荐使用时间戳来解决。   
 比如我们在SQL Server中的表中定义一个字段为timestamp类型的字段ts,这个字段的值不需要我们进行控制的。   
 在Insert与Update时，数据库会自己进行ts值的更新，因此我们只要在Update时使用：   
 Update xxx where [key=@key](mailto:key=@key) and [ts=@ts](mailto:ts=@ts) 就可以了，根本不用考虑ts里的值更新。   
 Delete时也最好进行一下判断，用这种方式是可以控制数据并发操作的。   
只需要在Update与Delete时，判断"影响条数"就可以知道更新是否成功。   
 但是不是所有的数据库都支持timestampr的，如在Access里没有timestamp字段，我们不能完全信赖于数据库的控制，而应该采用自设的控制机制，这样可以适应系统的数据库移值

此外，可以采用的其他策略

* + 分库（选择“二叉树分库”）与分表
  + 数据库高可用
  + Nginx实现粗细管道

max\_conns参数可以限制一台服务器的TCP连接数量，根据服务器的容量来设置恰当的配置数值，尤其在高并发的场景下，可以达到过载保护的目的

* + FIFO队列思路
  + Redis缓存策略

## 通用大牛级解法

把上面的解法说一遍………

**海量数据的解决方案：**

1. 使用缓存；
2. 页面静态化技术；
3. 数据库优化；
4. 分离数据库中活跃的数据；
5. 批量读取和延迟修改；
6. 读写分离；
7. 使用NoSQL和Hadoop等技术；
8. 分布式部署数据库；
9. 应用服务和数据服务分离；
10. 使用搜索引擎搜索数据库中的数据；
11. 进行业务的拆分；

**高并发情况下的解决方案：**

1. 应用程序和静态资源文件进行分离；
2. 页面缓存；
3. 集群与分布式；
4. 反向代理；
5. CDN；

**案例参照：** <http://blog.csdn.net/bbaiggey/article/details/53103022>

## 解法对比及优缺点

通常解法，简要说明几十万条并发数据并发处理方式及策略。通用解法，对海量数据并发提出优化策略及解决方案，优化点全面更加深入。

## 延伸及扩展问题回答参考

**问题**：

一. QPS,TPS

**解答**：

1 QPS：Query per second，每秒查询量

2. TPS：Transaction per second，每秒事物量

）

二．Nginx优化配置

一般来说nginx 配置文件中对优化比较有作用的为以下几项：

worker\_processes 8;

nginx 进程数，建议按照cpu 数目来指定，一般为它的倍数。

worker\_cpu\_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000;

为每个进程分配cpu，上例中将8 个进程分配到8 个cpu，当然可以写多个，或者将一  
个进程分配到多个cpu。

worker\_rlimit\_nofile 102400;

这个指令是指当一个nginx 进程打开的最多文件描述符数目，理论值应该是最多打开文  
件数（ulimit -n）与nginx 进程数相除，但是nginx 分配请求并不是那么均匀，所以最好与ulimit -n 的值保持一致。

use epoll;

使用epoll 的I/O 模型

worker\_connections 102400;

每个进程允许的最多连接数， 理论上每台nginx 服务器的最大连接数为worker\_processes\*worker\_connections。

keepalive\_timeout 60;

keepalive 超时时间。

client\_header\_buffer\_size 4k;

客户端请求头部的缓冲区大小，这个可以根据你的系统分页大小来设置，一般一个请求  
头的大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令getconf PAGESIZE 取得。

open\_file\_cache max=102400 inactive=20s;

这个将为打开文件指定缓存，默认是没有启用的，max 指定缓存数量，建议和打开文件数一致，inactive 是指经过多长时间文件没被请求后删除缓存。

open\_file\_cache\_valid 30s;

这个是指多长时间检查一次缓存的有效信息。

open\_file\_cache\_min\_uses 1;

open\_file\_cache 指令中的inactive 参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在inactive 时间内一次没被使用，它将被移除。

关于内核参数的优化：

net.ipv4.tcp\_max\_tw\_buckets = 6000

timewait 的数量，默认是180000。

net.ipv4.ip\_local\_port\_range = 1024 65000

允许系统打开的端口范围。

net.ipv4.tcp\_tw\_recycle = 1

启用timewait 快速回收。

net.ipv4.tcp\_tw\_reuse = 1

开启重用。允许将TIME-WAIT sockets 重新用于新的TCP 连接。

net.ipv4.tcp\_syncookies = 1

开启SYN Cookies，当出现SYN 等待队列溢出时，启用cookies 来处理。

net.core.somaxconn = 262144

web 应用中listen 函数的backlog 默认会给我们内核参数的net.core.somaxconn 限制到128，而nginx 定义的NGX\_LISTEN\_BACKLOG 默认为511，所以有必要调整这个值。

net.core.netdev\_max\_backlog = 262144

每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许送到队列的数据包的最大数目。

net.ipv4.tcp\_max\_orphans = 262144

系统中最多有多少个TCP 套接字不被关联到任何一个用户文件句柄上。如果超过这个数字，孤儿连接将即刻被复位并打印出警告信息。这个限制仅仅是为了防止简单的DoS 攻击，不能过分依靠它或者人为地减小这个值，更应该增加这个值(如果增加了内存之后)。

net.ipv4.tcp\_max\_syn\_backlog = 262144

记录的那些尚未收到客户端确认信息的连接请求的最大值。对于有128M 内存的系统而言，缺省值是1024，小内存的系统则是128。

net.ipv4.tcp\_timestamps = 0

时间戳可以避免序列号的卷绕。一个1Gbps 的链路肯定会遇到以前用过的序列号。时间戳能够让内核接受这种“异常”的数据包。这里需要将其关掉。

net.ipv4.tcp\_synack\_retries = 1

为了打开对端的连接，内核需要发送一个SYN 并附带一个回应前面一个SYN 的ACK。也就是所谓三次握手中的第二次握手。这个设置决定了内核放弃连接之前发送SYN+ACK 包的数量。

net.ipv4.tcp\_syn\_retries = 1

在内核放弃建立连接之前发送SYN 包的数量。

net.ipv4.tcp\_fin\_timeout = 1

如果套接字由本端要求关闭，这个参数决定了它保持在FIN-WAIT-2 状态的时间。对端可以出错并永远不关闭连接，甚至意外当机。缺省值是60 秒。2.2 内核的通常值是180 秒，3你可以按这个设置，但要记住的是，即使你的机器是一个轻载的WEB 服务器，也有因为大量的死套接字而内存溢出的风险，FIN- WAIT-2 的危险性比FIN-WAIT-1 要小，因为它最多只能吃掉1.5K 内存，但是它们的生存期长些。

net.ipv4.tcp\_keepalive\_time = 30

当keepalive 起用的时候，TCP 发送keepalive 消息的频度。缺省是2 小时。

三．缓存系统: Redis

四．消息队列：Rabbitmq

## 项目中体现经验的点

做过SQL Server数据库分库，Redis缓存，数据库timestamp使用，Nginx负载均衡配置等，适合大多中级工程师必须掌握的知识点

## 论坛参考

<http://blog.csdn.net/xlgen157387/article/details/53230138>

<http://blog.csdn.net/bbaiggey/article/details/53103022>

<http://casey.blog.51cto.com/9994043/1741461>

# try {}里有一个return语句，那么紧跟在这个try后的finally {}里的code会不会被执行，什么时候被执行，在return前还是后?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 80% | 2星 | .Net | 一场处理 | C# |

## 通常解法

**把return的返回值临时存储起来后再执行finall代码，然后再将return值返回，**return都是最后执行.

## 通用大牛级解法

把上面的解法说一遍………

return语句并不是函数的最终出口，只是将值压入堆栈中，如果有finally语句，这在return之后还会执行finally（return的值会暂存在栈里面，等待finally执行后再返回）可以通过查看编译的IL代码查看代码执行路径。

## 解法对比及优缺点

通常解法简要说明代码调用顺序及执行方式，通用方式通过深入分析编译代码，了解底层IL代码调用顺序及操作步骤，更能透过表象看本质。

## 延伸及扩展问题回答参考

**问题**：堆栈的定义及区别

解答

**堆栈空间分批**

栈（操作系统）：由操作系统自动分配释放 ，存放函数的[参数值](http://baike.baidu.com/view/1703315.htm)，[局部变量](http://baike.baidu.com/view/552847.htm)的值等。其操作方式类似于数据结构中的栈。

堆（操作系统）： 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收，分配方式倒是类似于链表

**堆栈缓存方式**

栈使用的是[一级缓存](http://baike.baidu.com/view/16882.htm)， 他们通常都是被调用时处于存储空间中，调用完毕立即释放。

堆则是存放在[二级缓存](http://baike.baidu.com/view/27650.htm)中，生命周期由虚拟机的垃圾回收算法来决定（并不是一旦成为孤儿对象就能被回收）。所以调用这些对象的速度要相对来得低一些。

**堆栈数据结构区别：**

堆（数据结构）：堆可以被看成是一棵树，如：堆排序。

栈（数据结构）：一种先进后出的数据结构。

## 项目中体现经验的点

C#基础

## 论坛参考

<https://www.cnblogs.com/hhx626/p/6010465.html>

# 什么是反射?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 80% | 3星 | .Net | C#架构 | 反射 |

## 通常解法

简述概念:反射是.NET中的重要机制，通过反射，可以在运行时获得程序或程序集中每一个 类型（包括类、结构、委托、接口和枚举等）的成员和成员的信息。有了反射，即可对每一个类型了如指掌。另外我还可以直接创建对象，即使这个对象的类型在编译时还不知道。

## 通用大牛级解法

简述概念的基础上解析反射常用程序集及作用

（1）使用Assembly定义和加载程序集，加载在程序集清单中列出模块，以及从此程序集中查找类型并创建该类型的实例。

（2）使用Module了解包含模块的程序集以及模块中的类等，还可以获取在模块上定义的所有全局方法或其他特定的非全局方法。

（3）使用ConstructorInfo了解构造函数的名称、参数、访问修饰符（如pulic 或private）和实现详细信息（如abstract或virtual）等。

（4）使用MethodInfo了解方法的名称、返回类型、参数、访问修饰符（如pulic 或private）和实现详细信息（如abstract或virtual）等。

（5）使用FiedInfo了解字段的名称、访问修饰符（如public或private）和实现详细信息（如static）等，并获取或设置字段值。

（6）使用EventInfo了解事件的名称、事件处理程序数据类型、自定义属性、声明类型和反射类型等，添加或移除事件处理程序。

（7）使用PropertyInfo了解属性的名称、数据类型、声明类型、反射类型和只读或可写状态等，获取或设置属性值。

（8）使用ParameterInfo了解参数的名称、数据类型、是输入参数还是输出参数，以及参数在方法签名中的位置等。。

## 解法对比及优缺点

本题主要在考核关于C#中反射的概念，所以解题的首选是将概念能清晰的描述清楚，在描述清楚概念的基础上再针对概念中涉及到的常用程序集、类库及方法做详细的阐述，不存在好坏的区分，是知识点的一个由浅入深的过程，也是考核对反射熟悉程度的过程。

## 延伸及扩展问题回答参考

**问题一：**反射用到的命名空间有哪些？

解答: System.Reflection

System.Type

System.Reflection.Assembly

**问题二**：获取给定类型的Type引用有3种常用方式：

1、使用 C# typeof 运算符。

Type t = typeof(string);

2、使用对象GetType()方法。

string s = "grayworm";

Type t = s.GetType();

3、还可以调用Type类的静态方法GetType()。

Type t = Type.GetType("System.String");

## 项目中体现经验的点

反射+工厂+配置文件，在创建抽象工厂工程的时候，会根据在配置文件中描述的示例动态通过反射创建对象。

## 论坛参考

http://blog.csdn.net/qq826364410/article/details/68488786

http://www.cnblogs.com/zhaopei/p/reflection.html

# 什么叫SQL注入，如何防止?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 90% | 5星 | .Net | SQL注入 | 数据库 |

## 通常解法

**简述概念:** 所谓SQL注入式攻击，就是输入域或页面请求的查询字符串，欺骗服务器执行恶意的SQL命令。在某些表单中，用户输入的内容直接用来构造（或者影响）动态SQL命令，或作为存储过程的输入参数，这类表单特别容易受到SQL注入式攻击。

一、 尽量避免拼接sql语句执行

二、 使用sql参数化

三、 通过过滤器对输入的内容进行筛选过滤。

## 通用大牛级解法

**解法1：**检查变量数据类型和格式

如果你的SQL语句是类似where id={$id}这种形式，数据库里所有的id都是数字，那么就应该在SQL被执行前，检查确保变量id是int类型；如果是接受邮箱，那就应该检查并严格确保变量一定是邮箱的格式，其他的类型比如日期、时间等也是一个道理。总结起来：只要是有固定格式的变量，在SQL语句执行前，应该严格按照固定格式去检查，确保变量是我们预想的格式，这样很大程度上可以避免SQL注入攻击。

**方法2：**过滤特殊符号

对于无法确定固定格式的变量，一定要进行特殊符号过滤或转义处理。

**方法3：**绑定变量，使用预编译语句，用SqlParameter参数化传入的变量

**方法4：**使用正则表达式过滤传入的参数

## 解法对比及优缺点

本题主要在考核关于sql防注入，针对于数据库安全的处理，所以解题的首选是将sql注入的流程有一定了解，了解之后在针对注入阐出一系列的解决方案，不存在好坏的区分，各有其好处，了解了sql注入的原理过程后，有助于打开我们的思路，想出更多的防注入的方法。

## 延伸及扩展问题回答参考

**问题一：**何如进行sql注入，直接危害至数据库？

**…..where id=’3 or 1=1’**

**问题二**：sql语句进行参数化的方法：

**Select \* from tablename where id = @id**

## 项目中体现经验的点

登录、注册等基本信息的表单提交的都需要防止注入攻击

所有在系统中的查询语句如果涉及将参数传入拼接sql都需要考虑注入攻击 的问题

## 论坛参考

http://blog.csdn.net/u013953289/article/details/32371541

http://www.cnblogs.com/baizhanshi/p/6002898.html

# sql server中count(\*),count(col),count(1)的区别

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 70% | 1星 | .Net | 求总数 | 数据库 |

## 通常解法

**简述概念:** count(1)和count(\*)都是统计表的总行数，两者执行结果相同。

表上没有主键或者唯一键索引，两者都走全表扫描；

表上有主键或者唯一键索引，那么走主键或者唯一键索引。

count(col)则是统计col列中不为空的总行数,如果该列存在索引，那么自动走索引（INDEX FULL SCAN）, 否则走全表扫描。

## 通用大牛级解法

**解法：**count(\*)和count(1)执行的效率是完全一样的。count(\*)的执行效率比count(col)高，因此可以用count(\*)的时候就不要去用count(col)。count(col)的执行效率比count(distinct col)高，不过这个结论的意义不大，这两种方法也是看需要去用。如果是对特定的列做count的话建立这个列的非聚集索引能对count有很大的帮助。如果经常count(\*)的话则可以找一个最小的col建立非聚集索引以避免全表扫描而影响整体性能。

## 解法对比及优缺点

本题主要在考核count(\*),count(col),count(1)三种的效率，针对于数据查询时的效率，所以解题的首选是了解三个的意思，然后在对症下药，三者各有其用，没有优缺点之分，主要根据项目所产生的需求来进行选择。

## 延伸及扩展问题回答参考

**问题一：**SELECT COUNT(\*) FROM tablename where ……… where条件存在效率高还是不存在高？

**答：**尽量减少SELECT COUNT(\*) FROM tablename WHERE COL = ‘value’ 这种查询；因为在有WHERE限制的情况下，总是需要进行全表遍历

**问题二**：COUNT(\*)与COUNT(COL)谁的效率高

**答：**在不加WHERE限制条件的情况下，COUNT(\*)与COUNT(COL)基本可以认为是等价的；但是在有WHERE限制条件的情况下，COUNT(\*)会比COUNT(COL)快非常多；

## 项目中体现经验的点

当查询时有为NULL值并且可以不计算NULL的行数时此时可以采用COUNT(COL)

## 论坛参考

<http://blog.163.com/bj_whj/blog/static/132129412011128113856421/>

<http://www.cnblogs.com/l1pe1/p/7216788.html>

<http://blog.csdn.net/jianglei421/article/details/5630611>

# 如何提高SqlServer安全性?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 50% | 5星 | .Net | 安全性 | 数据库 |

## 通常解法

**简述概念:** 数据库的安全性指的是防止非法用户访问数据库，避免造成数据的泄露、更改或破坏，以达到保护数据库的目的。为提高SQL Server数据库的安全性，可考虑修改端口、使用强密码、灵活运用触发器等办法。

## 通用大牛级解法

**解法：**方法一一列举：

1.安装最新的服务包

2.使用Microsoft基线安全性分析器(MBSA)来评估服务器

3.使用Windows身份验证模式

4.定期进行备份,并隔离服务器

5.分配一个强健的sa密码

6.在防火墙上禁用端口,提高SqlServer安全性

7.限制 SQL Server服务的权限

8.使用最安全的文件系统

9.删除或保护旧的安装文件

10.审核指向 SQL Server 的连接

11.修改SqlServer端口

12.灵活运用触发器

13.身份验证模式。

## 解法对比及优缺点

本题主要是讲SqlServer的安全性,没有优缺点之分,都为了提高SqlServer的安全性。

## 延伸及扩展问题回答参考

**问题：那么如何提高MySQL的安全性?**

**答：**(1)避免从互联网访问MySQL数据库,确保特定主机才拥有访问特权

(2)定期备份数据库

(3)禁用或限制远程访问

(4)设置root用户的口令并改变其登录名

(5)移除测试(test)数据库

(6)禁用LOCAL INFILE

(7)移除匿名账号和废弃账号

(8)降低系统特权

(9)降低用户的数据库特权

(10)移除和禁用.mysql\_history文件

(11)安全补丁

(12)启用日志

(13)改变root目录

## 项目中体现经验的点

1、数据安全，及时做数据备份

2、连接安全控制数据库的账户连接权限，设置只读只写账户，控制增删改查的权限

## 论坛参考

http://www.yesky.com/471/1913471\_2.shtml

http://www.fx361.com/page/2017/0425/1625039.shtml

https://wenku.baidu.com/view/612ac835bcd126fff7050bb9.html

http://blog.csdn.net/listen\_for/article/details/53907270

# 当正在为表建立索引的时候，SQL Server 会禁止对表的访问吗?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 面试概率 | 级别 | 应用模块或方向 | 解决问题 | 考核点 |
| 80% | 4星 | .Net | 快速查询 | 数据库索引 |

## 通常解法

在你建立索引时，SQL Server 不会禁止你对表进行读访问，除非你正在建立一个簇索引，或者在此时试图更新表中的数据，SQL Server 会禁止更新操作。

## 通用大牛级解法

**解法：**当SQL Server 完成了建立索引的操作后，它必须修改系统表来反映数据表的变化。两个最主要被修改的系统表是sysindexes 和 sysobjects 表。因为你没有使用“normal”SQL来建立索引，所以你不能对SQL Server的操作进行跟踪。

在最后一个阶段， 独占锁（IX lock）并不出现在数据表中，而是出现在系统表正在建立索引的相应数据表项上。 一般来说，最后这个阶段比较短，因为SQL Server已经对数据进行了排序并抽取了行指针（物理行定位符或簇索引键）。唯一剩下的任务就是修改系统表，这个过程很快。如果系统表上的独占锁还在，你不能执行任何查询操作，因为你不能从sysindexes 和 sysobjects系统表中读取出所需要的信息。

## 解法对比及优缺点

本题主要在考核关于建立索引，对SQL进行读写访问的概念，所以解题的首选是将概念能清晰的描述清楚。

## 延伸及扩展问题回答参考

**问题：对于一个频繁增删的表,如何建立索引,提高查询效率?**

**答：**对于数据量大的表，最好是需要有索引的。频繁删除更新也许造成了锁表，造成阻塞。一个好的方法是添加个锁表检测线程,自动排除死锁。另一个方法是,普通的删除只做逻辑删除,设置个flag字段，添加个线程定时做物理删除

## 项目中体现经验的点

尽量避免在使用索引时对SQL Server的更新操作。

## 论坛参考

<http://www.cnblogs.com/kevinlzf/archive/2009/10/28/1591201.html>

<http://bbs.csdn.net/topics/390899005>