

# 机器学习/算法工程师面试题库

牛客网出品



# 算法工程师/机器学习校招面试题库导读

## 一、学习说明

本题库均来自海量真实校招面试题目大数据进行的整理，后续也会不断更新，可免费在线观看，如需下载，也可直接在页面 <https://www.nowcoder.com/interview/center> 进行下载（下载需要用牛币兑换，一次兑换可享受永久下载权限，因为后续会更新）

需要严肃说明的是：面试题库作为帮助同学准备面试的辅助资料，但是绝对不能作为备考唯一途径，因为面试是一个考察真实水平的，不是背会了答案就可以的，需要你透彻理解的，否则追问问题答不出来反而减分，毕竟技术面试中面试官最痛恨的就是背答案这个事情了。

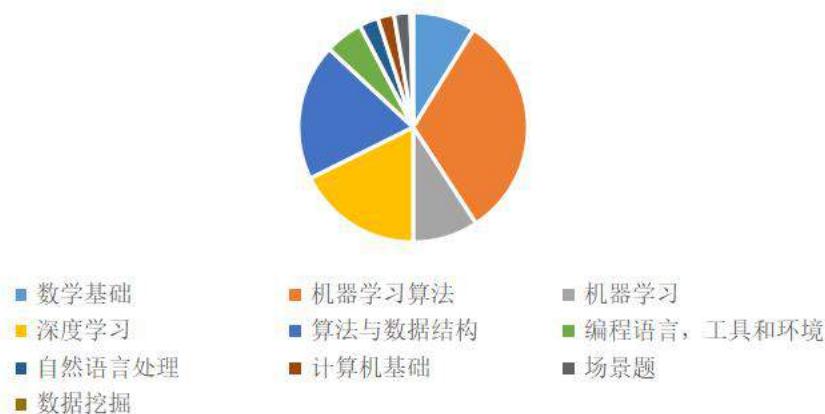
学完这个题库，把此题库都理解透彻应对各家企业面试完全没有问题。（当然，要加上好的项目以及透彻掌握）

另外，此面试题库中不包括面试中问到的项目，hr 面以及个人技术发展类。

- 项目是比较个性化的，没办法作为一个题库来给大家参考，但是如果你有一个非常有含金量的项目的话，是非常加分的，而且你的项目可能也会被问的多一些；
- hr 面的话一般来说技术面通过的话个人没有太大的和公司不符合的问题都能通过；
- 技术发展类的话这个就完全看自己啦，主要考察的会是你技术的热爱和学习能力，比如会问一些你是如何学习 xxx 技术的，或者能表达出你对技术的热爱的地方等等。此处不做赘述。

那么抛开这些，算法工程师/机器学习中技术面中考察的占比如下：

算法工程师/机器学习校招面试考点分布图  
牛客网出品



**需要注意的是：**此图不绝对，因为实际面试中面试官会根据你的简历去问，比如你的项目多

可能就问的项目多一些，或者你说哪里精通可能面试官就多去问你这些。而且此图是根据题库数据整理出来，并不是根据实际单场面试整理，比如基础部分不会考那么多，会从中抽着考

但是面试中必考的点且占比非常大的有机器学习算法，机器学习和**算法**。  
决定你是否能拿 sp offer（高薪 offer）以及是否进名企的是项目和**算法**。

可以看出，算法除了是面试必过门槛以外，更是决定你是否能进名企或高薪 offer 的决定性因素。

另外关于算法部分，想要系统的学习算法思想，实现高频面试题最优解等详细讲解的话可以报名[算法名企校招冲刺班](#)或[算法高薪校招冲刺班](#)，你将能学到更先进的算法思想以及又一套系统的校招高频题目的解题套路和方法论。

多出来的服务如下：

## 算法名企校招冲刺班

- ✓ 体系化直播教学
- ✓ 全程学习委员跟班
- ✓ 金牌助教一对一答疑
- ✓ 班级群讨论



**《程序员代码面试指南》**作者亲自讲解，前亚马逊，IBM，百度，GrowingIO 技术大牛，十年算法刷题经验

前100名报名可免费赠送签名书

如果有什么问题，也可以加 qq 咨询 1440073724，如果是早鸟，还可以领取早鸟优惠哦

## 二、面试技巧

面试一般分为技术面和 hr 面，形式的话很少有群面，少部分企业可能会有一个交叉面，不过总的来说，技术面基本就是考察你的专业技术水平的，hr 面的话主要是看这个人的综合素质以及家庭情况是否符合公司要求，一般来讲，技术的话只要通过了技术面 hr 面基本上是没有问题（也有少数企业 hr 面会刷很多人）

那我们主要来说技术面，技术面的话主要是考察专业技术知识和水平，我们是可以有一定的技巧的，但是一定是基于有一定的能力水平的。

所以也慎重的告诉大家，技巧不是投机取巧，是起到辅助效果的，技术面最主要的还是要有实力，这里是基于实力水平之上的技巧。

这里告诉大家面试中的几个技巧：

1、简历上做一个引导：

在词汇上做好区分，比如熟悉 Java，了解 python，精通 c 语言

这样的话对自己的掌握程度有个区分，也好让面试官有个着重去问，python 本来写的也只是了解，自然就不会多问你深入的一些东西了。

2、在面试过程中做一个引导：

面试过程中尽量引导到自己熟知的一个领域，比如问到你来说一下 DNS 寻址，然后你简单回答（甚至这步也可以省略）之后，可以说一句，自己对这块可能不是特别熟悉，对计算机网络中的运输层比较熟悉，如果有具体的，甚至可以再加一句，比如 TCP 和 UDP

这样的话你可以把整个面试过程往你熟知的地方引导，也能更倾向于体现出你的优势而不是劣势，但是此方法仅限于掌握合适的度，比如有的知识点是必会的而你想往别处引就有点说不过去了，比如让你说几个机器学习算法，你一个也说不上来，那可能就真的没辙了。

3、在自我介绍中做一个引导：

一般面试的开头都会有一个自我介绍，在这个位置你也可以尽情的为自己的优势方面去引导。

4、面试过程中展示出自信：

面试过程中的态度也要掌握好，不要自卑，也不要傲娇，自信的回答出每个问题，尤其遇到不会的问题，要么做一些引导，实在不能引导也可以先打打擦边球，和面试官交流一下问题，看起来像是没听懂题意，这个过程也可以再自己思考一下，如果觉得这个过程可以免了的话也直接表明一下这个地方不太熟悉或者还没有掌握好，千万不要强行回答。

面试前的准备：

最重要的肯定是系统的学习了，有一个知识的框架，基础知识的牢靠程度等。

其中算法尤其重要，越来越多公司还会让你现场或者视频面试中手写代码；

另一大重要的和加分项就是项目，在面试前，一定要练习回答自己项目的三个问题：

- 这是一个怎样的项目
- 用到了什么技术，为什么用这项技术（以及每项技术很细的点以及扩展）
- 过程中遇到了什么问题，怎么解决的。

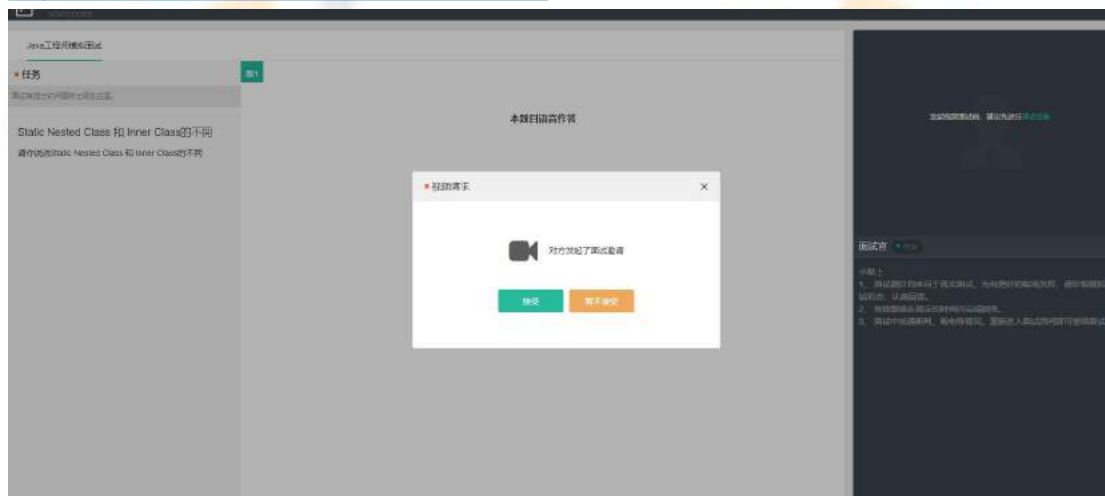
那么话说回来，这个的前提是你要有一个好的项目，牛客网 CEO 叶向宇有带大家做项目，感兴趣的可以去了解一下

- 竞争力超过 70% 求职者的项目：<https://www.nowcoder.com/courses/semester/medium>  
（专属优惠码：DjPgy3x，每期限量前 100 个）
- 竞争力超过 80% 求职者的项目：<https://www.nowcoder.com/courses/semester/senior>  
（专属优惠码：DMVSexJ，每期限量前 100 个）

知识都掌握好后，剩下的就是一个心态和模拟练习啦，因为你面试的少的话现场难免紧张，而且没在那个环境下可能永远不知道自己回答的怎么样。

因为哪怕当你都会了的情况下，你的表达和心态就显得更重要了，会了但是没有表达的很清晰就很吃亏了，牛客网这边有 AI 模拟面试，完全模拟了真实面试环境，正好大家可以真正的去练习一下，还能收获一份面试报告：

<https://www.nowcoder.com/interview/ai/index>



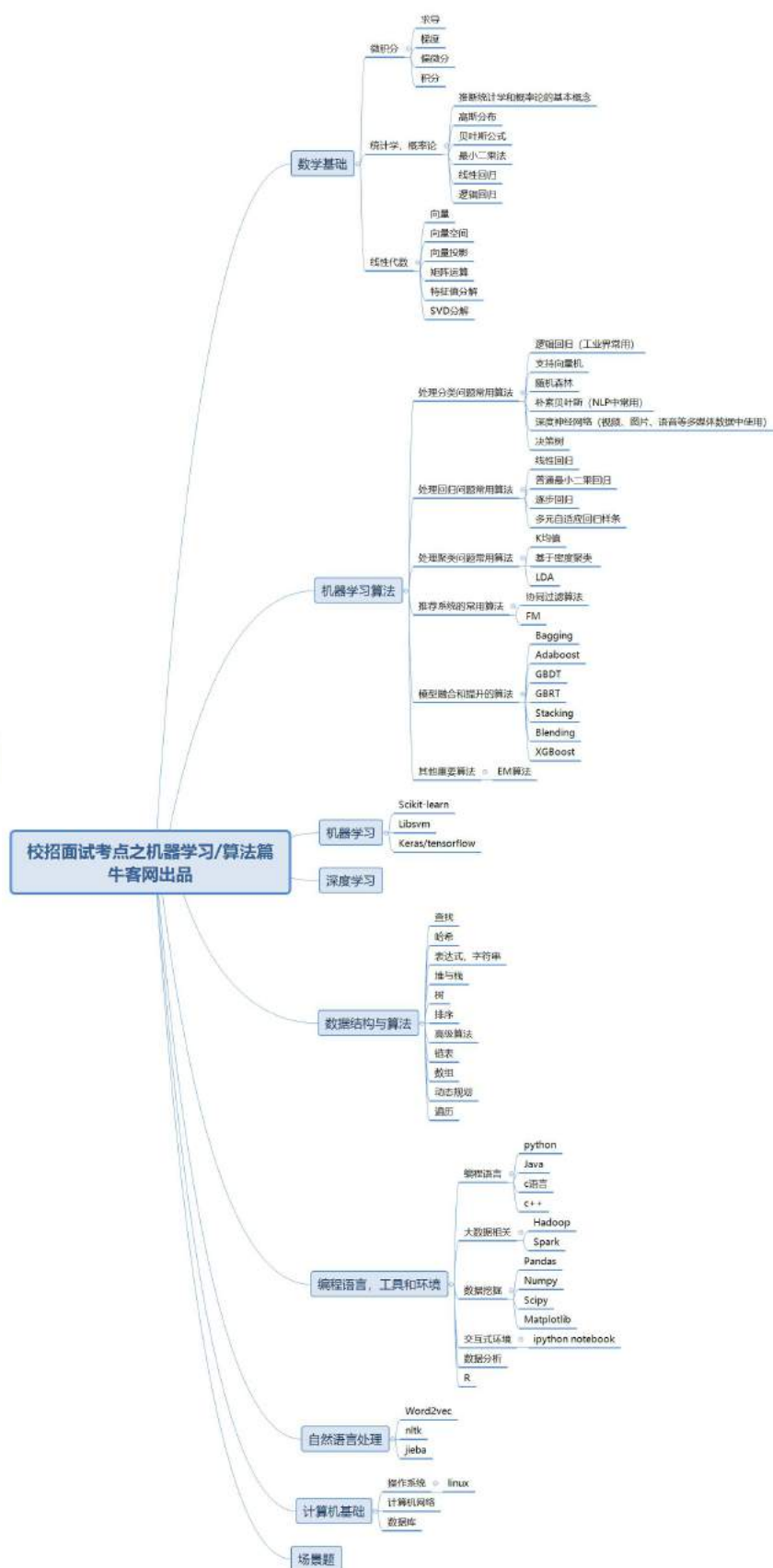
### 面试后需要做的：

面试完了的话就不用太在意结果了，有限的时间就应该做事半功倍的事情，当然，要保持电话邮箱畅通，不然别给你发 offer 你都不知道。

抛开这些，我们需要做的是及时将面试中的问题记录下来，尤其是自己回答的不够好的问题，一定要花时间去研究，并解决这些问题，下次面试再遇到相同的问题就能很好的解决，当然，即使不遇到，你这个习惯坚持住，后面也可以作为一个经历去跟面试官说，能表现出你对技术的喜爱和钻研的一个态度，同时，每次面试后你会发现自己的不足，查缺补漏的好机会，及时调整，在不断的调整和查缺补漏的过程中，你会越来越好。



### 三、面试考点导图



#### 四、一对一答疑讲解戳这里

如果你对校招求职或者职业发展很困惑，欢迎与牛客网专业老师沟通，老师会帮你一对一讲解答疑哦（可以扫下方二维码或者添加微信号：niukewang985）

## 专业老师，在线答疑

互联网校招求职全解惑



互联网校招求职如何准备，如何规划...  
测试自己校招中求职竞争力，适合公司...

扫码或添加老师微信：niukewang985

## 目录

### 一、数学基础：

- 1、微积分
- 2、统计学，概率论
- 3、线性代数

### 二、机器学习算法

- 1、处理分类问题常用算法
- 2、处理回归问题常用算法
- 3、处理聚类问题常用算法
- 4、推荐系统的常用算法
- 5、模型融合和提升的算法
- 6、其他重要算法

### 三、机器学习

- 1、Scikit-learn
- 2、Libsvm
- 3、Keras/tensorflow

### 四、深度学习

### 五、数据结构与算法

- 1、查找
- 2、哈希
- 3、表达式、字符串
- 4、堆与栈
- 5、树
- 6、排序
- 7、高级算法
- 8、链表
- 9、数组
- 10、动态规划
- 11、遍历

### 六、编程语言，工具和环境

- 1、编程语言
- 2、大数据相关

### 七、自然语言处理

- 1、Word2vec

### 八、计算机基础

- 1、linux



2、计算机网络

3、操作系统

4、数据库

九、场景题

十、项目

十一、hr 面

十二、数据挖掘

十三、学习与发展

更多名企历年笔试真题可点击直接进行练习：

<https://www.nowcoder.com/contestRoom>

## 一、数学基础：

### 1、微积分

#### 1、SGD,Momentum,Adagard,Adam 原理

考点:优化方法

详情:梯度

参考回答:

SGD 为随机梯度下降,每一次迭代计算数据集的 mini-batch 的梯度,然后对参数进行更新。

Momentum 参考了物理中动量的概念,前几次的梯度也会参与到当前的计算中,但是前几轮的梯度叠加在当前计算中会有一定的衰减。

Adagard 在训练的过程中可以自动变更学习速率,设置一个全局的学习率,而实际的学习率与以往的参数模和的开方成反比。

Adam 利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率,在经过偏置的校正后,每一次迭代后的学习率都有个确定的范围,使得参数较为平稳。

#### 2、L1 不可导的时候该怎么办

考点:函数不可导

标签:求导

参考回答:当损失函数不可导,梯度下降不再有效,可以使用坐标轴下降法,梯度下降是沿着当前点的负梯度方向进行参数更新,而坐标轴下降法是沿着坐标轴的方向,假设有  $m$  个特征个数,坐标轴下降法进行参数更新的时候,先固定  $m-1$  个值,然后再求另外一个的局部最优解,从而避免损失函数不可导问题。

使用 Proximal Algorithm 对 L1 进行求解,此方法是去优化损失函数上界结果。

#### 3、sigmoid 函数特性

考点:sigmoid 函数基础

标签:求导

参考回答:



定义域为 $(-\infty, +\infty)$

值域为 $(-1, 1)$

函数在定义域内为连续和光滑的函数

处处可导, 导数为 $f'(x) = f(x)(1 - f(x))$

## 2、统计学，概率论

### 1、 $a, b \sim U[0, 1]$ ，互相独立, 求 $\text{Max}(a, b)$ 期望

考点: 概率论

详情: 推断统计学和概率论的基本概念

参考回答: 略

### 2、一个活动, $n$ 个女生手里拿着长短不一的玫瑰花, 无序的排成一排, 一个男生从头走到尾, 试图拿更长的玫瑰花, 一旦拿了一朵就不能再拿其他的, 错过了就不能回头, 问最好的策略?

考点: 博弈论, 概率论

详情: 推断统计学和概率论基本概念

参考回答:

选择的策略为不选取前  $r-1$  个女生, 只从剩下的  $n-r+1$  个女生开始选取, 若任何一个女生比之前的女生玫瑰花都长则选取这个女生, 假设从第  $r$  个女生开始选, 则第  $k$  个被选中的女生拥有最长玫瑰花的概率为:

$$\begin{aligned} P &= \sum_{k=r}^n P(\text{第 } k \text{ 个女生被选中且拥有最长的玫瑰花}) \\ &= \sum_{k=r}^n P(\text{第 } k \text{ 个女生的玫瑰最长})P(\text{第 } k \text{ 个女生被选中} | \text{第 } k \text{ 个女生的玫瑰最长}) \\ &= \sum_{k=r}^n \frac{1}{n} P(\text{拥有最长玫瑰的女生出现在前 } r-1 \text{ 个女生中}) \end{aligned}$$

$$= \sum_{k=r}^n \frac{1}{n} \frac{r-1}{k-1} = \frac{r-1}{n} \sum_{k=r}^n \frac{1}{k-1}$$

当第  $r$  个为玫瑰最长的女生, 那么她被选中概率比第  $r+1$  个女生大, 则  $P_{r+1} \leq P_r$ , 即  $r^* = \min \{r | \sum_{k=r+1}^n \frac{1}{k-1} \leq 1\}$

因  $\sum_{k=r+1}^n \frac{1}{k-1} \approx \ln\left(\frac{n}{r^*}\right) = 1, r^* = n/e$ , 所以  $P_{r^*} = 1/e \approx 0.368$

在此策略下, 玫瑰最长女生被选中概率为 0.368

**3、问题：**某大公司有这么一个规定：只要有一个员工过生日，当天所有员工全部放假一天。但在其余时候，所有员工都没有假期，必须正常上班。这个公司需要雇用多少员工，才能让公司一年内所有员工的总工作时间期望值最大？

知识点：概率论

详情：推断统计学和概率论的基本概念

参考回答：

假设一年有 365 天，每个员工的生日都概率均等地分布在这 365 天里。

$$E = n * (1 - 1/365)^n$$

$$n = 365$$

#### 4、切比雪夫不等式

考点：大数定律

详情：推断统计学和概率论的基本概念

参考回答：

$$P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2} \quad k > 0, \mu \text{ 为期望}, \sigma \text{ 为标准差}$$

#### 5、一根绳子,随机截成 3 段,可以组成一个三角形的概率有多大

考点：数学



参考回答：

设绳子长为  $a$ ，折成三段的长度为  $x, y, a-x-y$  从而得到  $x > 0, y > 0, a-x-y > 0$ ，满足这三个约束条件在平面直角坐标系中的可行域为一个直角三角形，面积为  $\frac{1}{2}a^2$ 。而构成三角形的条件，任意两边和大于第三边的条件  $x+y > a-x-y, a-y > y, a-x > x$  同时成立。满足以上不等式在平面直角坐标系中也是一个直角三角形，面积为  $\frac{1}{8}a^2$ ，所以构成三角形概率为  $\frac{\frac{1}{8}a^2}{\frac{1}{2}a^2} = 0.25$ 。

## 6、最大似然估计和最大后验概率的区别？

考点：概率论

参考回答：

最大似然估计提供了一种给定观察数据来评估模型参数的方法，而最大似然估计中的采样满足所有采样都是独立同分布的假设。最大后验概率是根据经验数据获难以观察量的点估计，与最大似然估计最大的不同是最大后验概率融入了要估计量的先验分布在其中，所以最大后验概率可以看做规则化的最大似然估计。

## 7、什么是共轭先验分布

考点：概率论

参考回答：

假设  $\theta$  为总体分布中的参数， $\theta$  的先验密度函数为  $\pi(\theta)$ ，而抽样信息算得的后验密度函数与  $\pi(\theta)$  具有相同的函数形式，则称  $\pi(\theta)$  为  $\theta$  的共轭先验分布。

## 8、概率和似然的区别

考点：概率论

参考回答：

概率是指在给定参数  $\theta$  的情况下，样本的随机向量  $X=x$  的可能性。而似然表示的是在给定样本  $X=x$  的情况下，参数  $\theta$  为真实值的可能性。一般情况，对随机变量的取值用概率表示。而在非贝叶斯统计的情况下，参数为一个实数而不是随机变量，一般用似然来表示。

## 9、频率学派和贝叶斯学派的区别

考点：概率论

参考回答：





频率派认为抽样是无限的, 在无限的抽样中, 对于决策的规则可以很精确。贝叶斯派认为世界无时无刻不在改变, 未知的变量和事件都有一定的概率, 即后验概率是先验概率的修正。频率派认为模型参数是固定的, 一个模型在无数次抽样后, 参数是不变的。而贝叶斯学派认为数据才是固定的而参数并不是。频率派认为模型不存在先验而贝叶斯派认为模型存在先验。

#### 10、0~1 均匀分布的随机器如何变化成均值为 0，方差为 1 的随机器

标签：推断统计学和概率论的基本概念

参考回答：

0~1 的均匀分布是均值为 1/2，方差为 0. 转成均值为 0，方差为 1. 概率论题目

#### 11、Lasso 的损失函数

考点：损失函数

标签：逻辑回归

参考回答：

$$J(\theta) = \frac{1}{2} \|y - Xw\|^2 + \lambda \sum |\theta|$$

#### 12、Sift 特征提取和匹配的具体步骤

考点：特征点检测

标签：高斯分布

参考回答：

生成高斯差分金字塔, 尺度空间构建, 空间极值点检测, 稳定关键点的精确定位, 稳定关键点方向信息分配, 关键点描述, 特征点匹配。

### 3、线性代数

#### 1、求 $m \times k$ 矩阵 A 和 $n \times k$ 矩阵的欧几里得距离？

考点：矩阵的距离

详情：矩阵运算



参考回答：

先得到矩阵  $AB^T$ ，然后对矩阵 A 和矩阵  $B^T$  分别求出其中每个向量的模平方，并扩展为两个  $m \times k$  的矩阵  $A'$  和  $B'$ 。最终求得新的矩阵  $A' + B' - 2AB^T$ ，并将此矩阵开平方得到 A, B 向量集的欧几里得距离。

## 2、PCA 中第一主成分是第一的原因？

考点：降维技术

详情：特征值分解

参考回答：

略

## 3、欧拉公式

考点：数学

详情：向量

参考回答：

$$e^{ix} = \cos x + i \sin x$$

## 4、矩阵正定性的判断,Hessian 矩阵正定性在梯度下降中的应用

考点：矩阵正定性

详情：矩阵运算

参考回答：

若矩阵所有特征值均不小于 0，则判定为半正定。若矩阵所有特征值均大于 0，则判定为正定。在判断优化算法的可行性时 Hessian 矩阵的正定性起到了很大的作用，若 Hessian 正定，则函数的二阶偏导恒大于 0，函数的变化率处于递增状态，在牛顿法等梯度下降的方法中，Hessian 矩阵的正定性可以很容易的判断函数是否可收敛到局部或全局最优解。

## 5、概率题：抽蓝球红球，蓝结束红放回继续，平均结束游戏抽取次数

参考回答：

根据红球和蓝球的个数，依据概率公式，求出平均抽取次数。

## 6、讲一下 PCA

考点:降维技术

参考回答:

PCA 是比较常见的线性降维方法,通过线性投影将高维数据映射到低维数据中,所期望的是在投影的维度上,新特征自身的方差尽量大,方差越大特征越有效,尽量使产生的新特征间的相关性越小。

PCA 算法的具体操作为对所有的样本进行中心化操作,计算样本的协方差矩阵,然后对协方差矩阵做特征值分解,取最大的  $n$  个特征值对应的特征向量构造投影矩阵。

## 7、拟牛顿法的原理

考点:优化算法

标签: 矩阵运算

参考回答:

牛顿法的收敛速度快,迭代次数少,但是 Hessian 矩阵很稠密时,每次迭代的计算量很大,随着数据规模增大,Hessian 矩阵也会变大,需要更多的存储空间以及计算量。拟牛顿法就是在牛顿法的基础上引入了 Hessian 矩阵的近似矩阵,避免了每次都计算 Hessian 矩阵的逆,在拟牛顿法中,用 Hessian 矩阵的逆矩阵来代替 Hessian 矩阵,虽然不能像牛顿法那样保证最优化的方向,但其逆矩阵始终是正定的,因此算法始终朝最优化的方向搜索。

## 8、编辑距离

标签: 向量空间

参考回答:

概念

编辑距离的作用主要是用来比较两个字符串的相似度的

编辑距离, 又称 Levenshtein 距离 (莱文斯坦距离也叫做 Edit Distance), 是指两个字符串之间, 由一个转成另一个所需的最少编辑操作次数, 如果它们的距离越大, 说明它们越是不同。许可的编辑操作包括将一个字符替换成另一个字符, 插入一个字符, 删除一个字符。

在概念中, 我们可以看出一些重点那就是, 编辑操作只有三种。插入, 删除, 替换这三种操作, 我们有两个字符串, 将其中一个字符串经过上面的这三种操作之后, 得到两个完全相同的字符串付出的代价是什么就是我们要讨论和计算的。



例如：

我们有两个字符串：kitten 和 sitting：

现在我们要将 kitten 转换成 sitting

我们可以做如下的一些操作：

kitten -> sitten 将 K 替换成 S    sitten -> sittin 将 e 替换成 i

sittin -> sitting 添加 g

在这里我们设置每经过一次编辑，也就是变化（插入，删除，替换）我们花费的代价都是 1。

例如：

如果 str1="ivan"，str2="ivan"，那么经过计算后等于 0。没有经过转换。相似度  $= 1 - 0 / \text{Math.Max}(\text{str1.length}, \text{str2.length}) = 1$

如果 str1="ivan1"，str2="ivan2"，那么经过计算后等于 1。str1 的 "1" 转换 "2"，转换了一个字符，所以距离是 1，相似度  $= 1 - 1 / \text{Math.Max}(\text{str1.length}, \text{str2.length}) = 0.8$

算法过程

1. str1 或 str2 的长度为 0 返回另一个字符串的长度。 if(str1.length==0) return str2.length; if(str2.length==0) return str1.length;

2. 初始化  $(n+1) \times (m+1)$  的矩阵 d，并让第一行和列的值从 0 开始增长。扫描两字符串（n\*m 级的），如果：str1[i] == str2[j]，用 temp 记录它，为 0。否则 temp 记为 1。然后在矩阵 d[i, j] 赋予 d[i-1, j]+1、d[i, j-1]+1、d[i-1, j-1]+temp 三者的最小值。

3. 扫描完后，返回矩阵的最后一个值 d[n][m] 即是它们的距离。

计算相似度公式：1-它们的距离/两个字符串长度的最大值。

我们用字符串 "ivan1" 和 "ivan2" 举例来看看矩阵中值的状况：

1、第一行和第一列的值从 0 开始增长

		i	v	a	n	1
	0	1	2	3	4	5
i	1					
v	2					
a	3					
n	4					
2	5					

图一

首先我们先创建一个矩阵，或者说是我们的二维数列，假设有两个字符串，我们的字符串的长度分别是  $m$  和  $n$ ，那么，我们矩阵的维度就应该是  $(m+1)*(n+1)$ 。

注意，我们先给数列的第一行第一列赋值，从 0 开始递增赋值。我们就得到了图一的这个样子

之后我们计算第一列，第二列，依次类推，算完整个矩阵。

我们的计算规则就是：

$d[i, j] = \min(d[i-1, j]+1, d[i, j-1]+1, d[i-1, j-1]+temp)$  这三个当中的最小值。

其中： $str1[i] == str2[j]$ ，用  $temp$  记录它，为 0。否则  $temp$  记为 1

我们用  $d[i-1, j]+1$  表示增加操作

$d[i, j-1]+1$  表示我们的删除操作

$d[i-1, j-1]+temp$  表示我们的替换操作

2、举证元素的产生  $Matrix[i-1, j] + 1$  ;  $Matrix[i, j-1] + 1$  ;  $Matrix[i-1, j-1] + t$  三者当中的最小值

		i	v	a	n	1
	$0+t=0$	$1+1=2$	2	3	4	5
i	$1+1=2$	取三者最小值=0				
v	2	依次类推：1				
a	3	2				
n	4	3				
2	5	4				

3. 依次类推直到矩阵全部生成



		i	v	a	n	1
	0	1	2			
i	1	0	1			
v	2	1	0			
a	3	2	1			
n	4	3	2			
2	5	4	3			

		i	v	a	n	1
	0	1	2	3	4	5
i	1	0	1	2	3	4
v	2	1	0	1	2	3
a	3	2	1	0	1	2
n	4	3	2	1	0	1
2	5	4	3	2	1	1

这个就得到了我们的整个完整的矩阵。

## 二、机器学习算法

### 1、处理分类问题常用算法

#### 1、交叉熵公式

标签：决策树

参考回答：

交叉熵：设  $p(x)$ 、 $q(x)$  是  $X$  中取值的两个概率分布，则  $p$  对  $q$  的相对熵是：

$$D(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = E_{p(x)} \log \frac{p(x)}{q(x)}$$

在一定程度上，相对熵可以度量两个随机变量的“距离”，且有  $D(p \parallel q) \neq D(q \parallel p)$ 。另外，值得一提的是， $D(p \parallel q)$  是必然大于等于 0 的。

互信息：两个随机变量  $X, Y$  的互信息定义为  $X, Y$  的联合分布和各自独立分布乘积的相对熵，

用  $I(X, Y)$  表示：

$$I(X, Y) = \sum_{x, y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

且有  $I(X, Y) = D(P(X, Y) \parallel P(X)P(Y))$ 。下面，咱们来计算下  $H(Y) - I(X, Y)$  的结果，如下：



$$\begin{aligned}
H(Y) - I(X, Y) &= -\sum_y p(y) \log p(y) - \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\
&= -\sum_y \left( \sum_x p(x, y) \right) \log p(y) - \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\
&= -\sum_{x,y} p(x, y) \log p(y) - \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\
&= -\sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)} \\
&= -\sum_{x,y} p(x, y) \log p(y | x) \\
&= H(Y | X)
\end{aligned}$$

## 2、LR 公式

标签：逻辑回归

参考回答：

逻辑回归本质上是线性回归，只是在特征到结果的映射中加入了一层逻辑函数  $g(z)$ ，即先把特征线性求和，然后使用函数  $g(z)$  作为假设函数来预测。 $g(z)$  可以将连续值映射到 0 和 1。 $g(z)$  为 sigmoid function.

$$g(z) = \frac{1}{1 + e^{-z}}$$

则

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

sigmoid function 的导数如下：

$$\begin{aligned}
g'(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\
&= \frac{1}{(1 + e^{-z})^2} (e^{-z}) \\
&= \frac{1}{(1 + e^{-z})} \cdot \left( 1 - \frac{1}{(1 + e^{-z})} \right) \\
&= g(z)(1 - g(z)).
\end{aligned}$$

逻辑回归用来分类 0/1 问题，也就是预测结果属于 0 或者 1 的二值分类问题。这里假设了二值满足伯努利分布，也就是

$$\begin{aligned}P(y = 1 \mid x; \theta) &= h_{\theta}(x) \\P(y = 0 \mid x; \theta) &= 1 - h_{\theta}(x)\end{aligned}$$

其也可以写成如下的形式：

$$p(y \mid x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$

对于训练数据集，特征数据  $x = \{x_1, x_2, \dots, x_m\}$  和对应的分类标签  $y = \{y_1, y_2, \dots, y_m\}$ ，假设  $m$  个样本是相互独立的，那么，极大似然函数为：

$$\begin{aligned}L(\theta) &= p(\vec{y} \mid X; \theta) \\&= \prod_{i=1}^m p(y^{(i)} \mid x^{(i)}; \theta) \\&= \prod_{i=1}^m (h_{\theta}(x^{(i)}))^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}}\end{aligned}$$

$\log$  似然为：

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\&= \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))\end{aligned}$$

如何使其最大呢？与线性回归类似，我们使用梯度上升的方法（求最小使用梯度下降），那么  $\theta := \theta + \alpha \nabla_{\theta} \ell(\theta)$ 。

$$\begin{aligned}\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\&= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x) (1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\&= (y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x)) x_j \\&= (y - h_{\theta}(x)) x_j\end{aligned}$$

如果只用一个训练样例  $(x, y)$ ，采用随机梯度上升规则，那么随机梯度上升更新规则为：

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

### 3、LR 的推导，损失函数

标签：逻辑回归

参考回答：

逻辑回归本质上是线性回归，只是在特征到结果的映射中加入了一层逻辑函数  $g(z)$ ，即先把特征线性求和，然后使用函数  $g(z)$  作为假设函数来预测。 $g(z)$  可以将连续值映射到 0 和 1。 $g(z)$  为 sigmoid function.

$$g(z) = \frac{1}{1 + e^{-z}}$$

则

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

sigmoid function 的导数如下：

$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\ &= \frac{1}{(1 + e^{-z})^2} (e^{-z}) \\ &= \frac{1}{(1 + e^{-z})} \cdot \left(1 - \frac{1}{(1 + e^{-z})}\right) \\ &= g(z)(1 - g(z)). \end{aligned}$$

逻辑回归用来分类 0/1 问题，也就是预测结果属于 0 或者 1 的二值分类问题。这里假设了二值满足伯努利分布，也就是

$$\begin{aligned} P(y = 1 | x; \theta) &= h_{\theta}(x) \\ P(y = 0 | x; \theta) &= 1 - h_{\theta}(x) \end{aligned}$$

其也可以写成如下的形式：

$$p(y | x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$

对于训练数据集，特征数据  $x = \{x_1, x_2, \dots, x_m\}$  和对应的分类标签  $y = \{y_1, y_2, \dots, y_m\}$ ，假设  $m$  个样本是相互独立的，那么，极大似然函数为：

$$\begin{aligned} L(\theta) &= p(\vec{y} | X; \theta) \\ &= \prod_{i=1}^m p(y^{(i)} | x^{(i)}; \theta) \\ &= \prod_{i=1}^m (h_{\theta}(x^{(i)}))^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

log 似然为：



$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))\end{aligned}$$

如何使其最大呢？与线性回归类似，我们使用梯度上升的方法（求最小使用梯度下降），那么  $\theta := \theta + \alpha \nabla_{\theta} \ell(\theta)$ 。

$$\begin{aligned}\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x) (1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\ &= (y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x)) x_j \\ &= (y - h_{\theta}(x)) x_j\end{aligned}$$

如果只用一个训练样例  $(x, y)$ ，采用随机梯度上升规则，那么随机梯度上升更新规则为：

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

损失函数：

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m (-y \log(\hat{y}^{(i)}) - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$

#### 4、逻辑回归怎么实现多分类

考点：逻辑回归的应用

详情：逻辑回归

参考回答：

方式一：修改逻辑回归的损失函数，使用 softmax 函数构造模型解决多分类问题，softmax 分类模型会有相同于类别数的输出，输出的值为对于样本属于各个类别的概率，最后对于样本进行预测的类型为概率值最高的那个类别。

方式二：根据每个类别都建立一个二分类器，本类别的样本标签定义为 0，其它分类样本标签定义为 1，则有多少个类别就构造多少个逻辑回归分类器

若所有类别之间有明显的互斥则使用 softmax 分类器，若所有类别不互斥有交叉的情况则构造相应类别个数的逻辑回归分类器。

#### 5、SVM 中什么时候用线性核什么时候用高斯核？

考点：SVM 应用





详情：支持向量机

公司：阿里菜鸟

参考回答：

当数据的特征提取的较好，所包含的信息量足够大，很多问题是线性可分的那么可以采用线性核。若特征数较少，样本数适中，对于时间不敏感，遇到的问题是线性不可分的时候可以使用高斯核来达到更好的效果。

## 6、什么是支持向量机, SVM 与 LR 的区别？

考点：SVM 基础

详情：支持向量机

参考回答：支持向量机为一个二分类模型，它的基本模型定义为特征空间上的间隔最大的线性分类器。而它的学习策略为最大化分类间隔，最终可转化为凸二次规划问题求解。

LR 是参数模型，SVM 为非参数模型。LR 采用的损失函数为 logistical loss，而 SVM 采用的是 hinge loss。在学习分类器的时候，SVM 只考虑与分类最相关的少数支持向量点。LR 的模型相对简单，在进行大规模线性分类时比较方便。

## 7、监督学习和无监督学习的区别

考点：机器学习基础

详情：随机森林

参考回答：

输入的数据有标签则为监督学习，输入数据无标签为非监督学习。

## 8、机器学习中的距离计算方法？

考点：距离测量方法

详情：支持向量机

参考回答：

设空间中两个点为  $A(x_1, y_1), B(x_2, y_2)$

欧式距离： $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

曼哈顿距离： $|x_1 - x_2| + |y_1 - y_2|$

$$\text{余弦距离: } \cos \theta = \frac{x_1 * x_2 + y_1 * y_2}{\sqrt{x_1^2 + y_1^2} * \sqrt{x_2^2 + y_2^2}}$$

$$\text{切比雪夫距离: } \max(x_1 - x_2, y_1 - y_2)$$

**9、问题：朴素贝叶斯（naive Bayes）法的要求是？**

知识点：NB

详情：朴素贝叶斯（NLP 中常用）

参考回答：： 贝叶斯定理、特征条件独立假设

解析：朴素贝叶斯属于生成式模型，学习输入和输出的联合概率分布。给定输入  $x$ ，利用贝叶斯概率定理求出最大的后验概率作为输出  $y$ 。

**10、问题：训练集中类别不均衡，哪个参数最不准确？**

知识点：类别不平衡、评价指标的了解

详情：随机森林

参考回答：准确度（Accuracy）

解析：举例，对于二分类问题来说，正负样例比相差较大为 99:1，模型更容易被训练成预测较大占比的类别。因为模型只需要对每个样例按照 0.99 的概率预测正类，该模型就能达到 99% 的准确率。

**11、问题：你用的模型，最有挑战性的项目**

知识点：

详情：支持向量机

参考回答：在回答自己的模型时，必须要深入了解自己的模型细节以及其中用到知识（如：Bi-LSTM 的优点以及与 rnn 和 lstm 的对比）的原理。

**12、问题：SVM 的作用，基本实现原理；**

知识点：SVM

详情：支持向量机

参考回答：SVM 可以用于解决二分类或者多分类问题，此处以二分类为例。SVM 的目标是寻找一个最优化超平面在空间中分割两类数据，这个最优化超平面需要满足的条件是：离其最近的点到其的距离最大化，这些点被称为支持向量。

解析：建议练习推导 SVM，从基本式的推导，到拉格朗日对偶问题。

13、问题：SVM 的硬间隔，软间隔表达式；

知识点：SVM

详情：支持向量机

$$\min_{w,b} \frac{1}{2} \|w\|^2$$
$$st. \quad y^{(i)}(w^T x^{(i)} + b) \geq 1$$

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i$$
$$st. \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

参考回答：左边为硬间隔；右边为软间隔

解析：不同点在于有无引入松弛变量

14、问题：SVM 使用对偶计算的目的是什么，如何推出来的，手写推导；

知识点：SVM

详情：支持向量机

答案：目的有两个：一是方便核函数的引入；二是原问题的求解复杂度与特征的维数相关，而转成对偶问题后只与问题的变量个数有关。由于 SVM 的变量个数为支持向量的个数，相较于特征位数较少，因此转对偶问题。通过拉格朗日算子使带约束的优化目标转为不带约束的优化函数，使得  $w$  和  $b$  的偏导数等于零，带入原来的式子，再通过转成对偶问题。

解析：

15、问题：SVM 的物理意义是什么；

知识点：SVM

详情：支持向量机

答案：构造一个最优化的超平面在空间中分割数据

16、问题：如果给你一些数据集，你会如何分类（我是分情况答的，从数据的大小，特征，是否有缺失，分情况分别答的）；

知识点：分类



详情：逻辑回归（工业界常用）

答案：根据数据类型选择不同的模型，如 Lr 或者 SVM，决策树。假如特征维数较多，可以选择 SVM 模型，如果样本数量较大可以选择 LR 模型，但是 LR 模型需要进行数据预处理；假如缺失值较多可以选择决策树。选定完模型后，相应的目标函数就确定了。还可以在考虑正负样例比比，通过上下集采样平衡正负样例比。

解析：需要了解多种分类模型的优缺点，以及如何构造分类模型的步骤

#### 17、问题：如果数据有问题，怎么处理；

知识点：实践问题

详情：随机森林

答案：1. 上下采样平衡正负样例比；2. 考虑缺失值；3. 数据归一化

解析：发散问题需要自己展现自己的知识面

#### 18、分层抽样的适用范围

考点：抽样调查

详情：随机森林

参考回答：

分层抽样利用事先掌握的信息，充分考虑了保持样本结构和总体结构的一致性，当总体由差异明显的几部分组成的时候，适合用分层抽样。

#### 19、LR 的损失函数

考点：逻辑回归基础

详情：逻辑回归（工业界常用）

参考回答：

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y_i \log(h_{\theta}(x_i)) + (1 - y_i) \log(1 - h_{\theta}(x_i))]$$

$m$  为样本个数， $h_{\theta}(x_i)$  为模型对样本  $i$  的预测结果， $y_i$  为样本  $i$  的真实标签。



## 20、LR 和线性回归的区别

考点:模型比较

详情: 逻辑回归 (工业界常用)

参考回答:

线性回归用来做预测, LR 用来做分类。线性回归是来拟合函数, LR 是来预测函数。线性回归用最小二乘法来计算参数, LR 用最大似然估计来计算参数。线性回归更容易受到异常值的影响, 而 LR 对异常值有较好的稳定性。

## 21、生成模型和判别模型基本形式, 有哪些?

参考回答:

生成式: 朴素贝叶斯、HMM、Gaussians、马尔科夫随机场

判别式: LR, SVM, 神经网络, CRF, Boosting

详情: 支持向量机

## 22、核函数的种类和应用场景。

参考回答:

线性核、多项式核、高斯核。

特征维数高选择线性核

样本数量可观、特征少选择高斯核 (非线性核)

样本数量非常多选择线性核 (避免造成庞大的计算量)

详情: 支持向量机

## 23、分类算法列一下有多少种? 应用场景。

参考回答:

单一的分类方法主要包括: LR 逻辑回归, SVM 支持向量机, DT 决策树、NB 朴素贝叶斯、NN 人工神经网络、K-近邻; 集成学习算法: 基于 Bagging 和 Boosting 算法思想, RF 随机森林, GBDT, Adaboost, XGboost。

详情: 随机森林

## 24、给你一个检测的项目, 检测罐装的可口可乐, 瓶装的可口可乐作为负样本, 怎么弄?



参考回答：

二分类问题，两类标签：罐装和瓶装，特征工程、构建特征；选择模型（LR, SVM, 决策树等）、输入模型训练。考虑下正负样例比。

## 25、SVM 核函数的选择

考点:SVM

参考回答：

当样本的特征很多且维数很高时可考虑用 SVM 的线性核函数。当样本的数量较多, 特征较少时, 一般手动进行特征的组合再使用 SVM 的线性核函数。当样本维度不高且数量较少时, 且不知道该用什么核函数时一般优先使用高斯核函数, 因为高斯核函数为一种局部性较强的核函数, 无论对于大样本还是小样本均有较好的性能且相对于多项式核函数有较少的参数。

## 26、SVM 的损失函数

考点:SVM

参考回答：

$$J(\theta) = \frac{1}{2} \|\theta\|^2 + c \sum_i \max(0, 1 - y_i(\theta^T x_i + b))$$

## 27、核函数的作用

考点:SVM

参考回答：

核函数隐含着一个从低维空间到高维空间的映射, 这个映射可以把低维空间中线性不可分的两类点变成线性可分的。

## 28、SVM 为什么使用对偶函数求解

考点:SVM 基础

参考回答：



对偶将原始问题中的约束转为了对偶问题中的等式约束, 而且更加方便了核函数的引入, 同时也改变了问题的复杂度, 在原始问题下, 求解问题的复杂度只与样本的维度有关, 在对偶问题下, 只与样本的数量有关。

### 29、ID3,C4.5 和 CART 三种决策树的区别

考点: 决策树基础

参考回答:

ID3 决策树优先选择信息增益大的属性来对样本进行划分, 但是这样的分裂节点方法有一个很大的缺点, 当一个属性可取值数目较多时, 可能在这个属性对应值下的样本只有一个或者很少个, 此时它的信息增益将很高, ID3 会认为这个属性很适合划分, 但实际情况下叫多属性的取值会使模型的泛化能力较差, 所以 C4.5 不采用信息增益作为划分依据, 而是采用信息增益率作为划分依据。但是仍不能完全解决以上问题, 而是有所改善, 这个时候引入了 CART 树, 它使用 gini 系数作为节点的分裂依据。

### 30、SVM 和全部数据有关还是和局部数据有关?

考点: SVM 基础

参考回答:

SVM 只和分类界限上的支持向量点有关, 换言之只和局部数据有关。

### 31、为什么高斯核能够拟合无穷维度

考点: SVM 基础

参考回答:

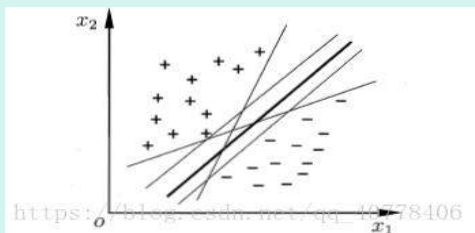
因为将泰勒展开式代入高斯核, 将会得到一个无穷维度的映射。

### 32、完整推导了 svm 一遍, 还有强化学习问的很多, dqn 的各种 trick 了解多少, 怎么实现知不知道。

参考回答:

SVM 推导:

支持向量机是一种二分类模型, 他的基本想法就是基于训练集和样本空间中找到一个最好的划分超平面, 将两类样本分割开来, 首先你就要知道什么样的划分发才能称为“最”好划分



看上图，二维平面上有两类样本，一类是用‘+’表示，另一类用‘-’表示，那么中间那几条划分线每条都能将两类样本分割开来，但我们一眼就注意到中间那条加粗的划分超平面，似乎他是最好的，因为两类的样本点都离他挺远的，专业点说就是该划分超平面对训练样本局部扰动的‘容忍’性最好。好，这还只是个二维平面，我们可以通过可视化大概寻找这样一个超平面，但如果三维，四维，五维呢，我们必须用我们擅长的数学去描述它，推导它。

在样本空间中，划分超平面可用  $w^T x + b = 0$  表示，记为  $(w, b)$ ，样本点  $(x_i, y_i)$  到划分超平面的函数间隔为  $\hat{\gamma}_i = y_i(w \cdot x_i + b)$ ，几何间隔为：
$$\gamma = \frac{\hat{\gamma}}{\|w\|}$$

若  $\|w\|=1$ ，可知函数间隔和几何间隔相等，若超平面参数  $w, b$  成比例的改变（超平面没有变），则函数间隔也是成比例的改变，而几何间隔不变。

支持向量机的基本想法就是求解能够正确划分训练数据集并且几何间隔最大的分离超平面，表达为数学公式即为：

$$\begin{aligned} \max_{w, b} \quad & \frac{\hat{\gamma}}{\|w\|} \\ \text{s.t.} \quad & y_i(w \cdot x_i + b) \geq \hat{\gamma}, \quad i=1, 2, \dots, N \end{aligned}$$

其实函数间隔  $\hat{\gamma}$  的取值并不影响最优化问题的解，假设将  $w$  和  $b$  成倍的改变为  $aw, ab$ ，那么函数间隔也会相应变成  $a\hat{\gamma}$ ，函数间隔的对上面最优化问题的不等式没有影响，也对目标函数没有影响，因此为简便，取  $\hat{\gamma}=1$ ，而且我们注意到最大化  $\frac{1}{\|w\|}$  等价于最小化  $\frac{1}{2}\|w\|^2$ （为啥取平方呢，因为后面好求导），便可得到下面支持线性可分（线性不可分的情况后面会提到）的支持向量机的最优化问题

$$\begin{aligned} \min_{w, b} \quad & \frac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y_i(w \cdot x_i + b) - 1 \geq 0, \quad i=1, 2, \dots, N \end{aligned}$$

这是一个凸二次优化的问题，可以直接求解，但是为了简便呢，我们要应用拉格朗日对偶性，求解他的对偶问题

其实求解对偶问题相比于原问题有以下几点好处(1). 对偶问题更容易求解，因为不用求  $w$  了 (2)我们可以自然引入核函数，这样可以推广到线性不可分分类问题上

建立拉格朗日函数，引进拉格朗日乘子  $\alpha_i \geq 0, i=1,2,\dots,N$ ，定义拉格朗日函数：

$$L(w,b,\alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i y_i (w \cdot x_i + b) + \sum_{i=1}^N \alpha_i$$

根据原始问题的对偶性，原始问题的对偶性是极大极小问题，即  $\max_{\alpha} \min_{w,b} L(w,b,\alpha)$

首先我们来求最小，零  $L(w,b,a)$  分别对  $w$  和  $b$  求导为零可得

$$\begin{aligned} w &= \sum_{i=1}^m \alpha_i y_i x_i, \\ 0 &= \sum_{i=1}^m \alpha_i y_i. \end{aligned}$$

将其代入对偶问题，可得

$$\max_{\alpha} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$\begin{aligned} \text{s.t. } & \sum_{i=1}^m \alpha_i y_i = 0, \\ & \alpha_i \geq 0, \quad i = 1, 2, \dots, m. \end{aligned}$$

解出  $\alpha$  之后，那么  $w, b$  也相应得到啦

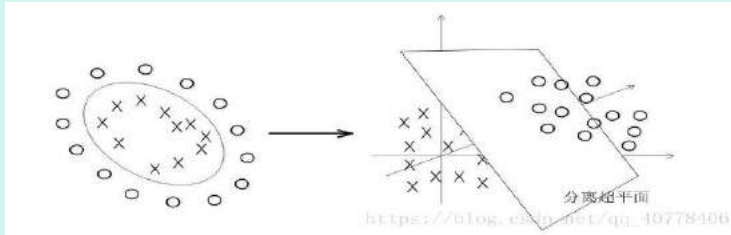
$$\begin{aligned} f(x) &= w^T x + b \\ &= \sum_{i=1}^m \alpha_i y_i x_i^T x + b. \end{aligned}$$

接下来，我们看看很有意思的上式不等式约束的 kkt 条件（不懂请百度）带给我们的信息

$$\begin{cases} \alpha_i \geq 0; \\ y_i f(x_i) - 1 \geq 0; \\ \alpha_i (y_i f(x_i) - 1) = 0. \end{cases}$$

咦，对于任意训练样本  $(x_i, y_i)$ ，总有  $\alpha_i = 0$  或者  $y_i f(x_i) = 1$ ，也就是说最终与模型有关的样本点都位于最大间隔的边界上，我们称之为支持向量，其余的样本点与模型无关

在前面的讨论中，我们都是聊的线性可分的情况，那么大多数情况下都线性不可分怎么办，比如这样（如左）



山人自有妙计，我们可以将样本先映射到高维特征空间，然后就可以继续分割了（如右）

前面我们说到对偶问题是

$$\max_{\alpha} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

$$\text{s.t.} \quad \sum_{i=1}^m \alpha_i y_i = 0,$$

$$\alpha_i \geq 0, \quad i = 1, 2, \dots, m.$$

公式中涉及到计算  $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ ， $\mathbf{x}_i, \mathbf{x}_j$  是映射到特征空间之后的内积，由于特征维数可能会很高，甚至是无穷多维，直接计算很困难，所以我们引入了核函数：

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

这样我们就可以不用麻烦的计算内积了

dqn 的各种 trick:

第一个 Trick。DQN 引入卷积层。模型通过 Atari 游戏视频图像了解环境信息并学习策略。DQN 需要理解接收图像，具有图像识别能力。卷积神经网络，利用可提取空间结构信息卷积层抽取特征。卷积层提取图像中重要目标特征传给后层做分类、回归。DQN 用卷积层做强化学习训练，根据环境图像输出决策。

第二个 Trick。Experience Replay。深度学习需要大量样本，传统 Q-Learning online update 方法（逐一对新样本学习）不适合 DQN。增大样本，多个 epoch 训练，图像反复利用。Experience Replay，储存 Agent Experience 样本，每次训练随机抽取部分样本供网络学习。稳定完成学习任务，避免短视只学习最新接触样本，综合反复利用过往大量样本学习。创建储存 Experience 缓存 buffer，储存一定量较新样本。容量满了，用新样本替换最旧样本，保证大部分样本相近概率被抽到。不替换旧样本，训练过程被抽到概率永远比新样本高很多。每次需要训练样本，直接从 buffer 随机抽取一定量给 DQN 训练，保持样本高利用率，让模型学习到较新样本。

第三个 Trick。用第二个 DQN 网络辅助训练，target DQN，辅助计算目标 Q 值，提供学习目标公式里的  $\max_a Q(s_{t+1}, a)$ 。两个网络，一个制造学习目标，一个实际训练，让 Q-Learning 训练目标保持平稳。强化学习 Q-Learning 学习目标每次变化，学习目标分部是模型本身输出，每次更新模型参数会导致学习目标变化，更新频繁幅度大，训练过程会非常不稳定、失控，DQN 训



练会陷入目标 Q 值与预测 Q 值反馈循环(陷入震荡发散，难收敛)。需要稳定 target DQN 辅助网络计算目标 Q 值。target DQN，低频率、缓慢学习，输出目标 Q 值波动较小，减小训练过程影响。

第四个 Trick。Double DQN。传统 DQN 高估 Action Q 值，高估不均匀，导致次优 Action 被高估超过最优 Action。targetDQN 负责生成目标 Q 值，先产生  $Q(st+1, a)$ ，再通过  $\max_a$  选择最大 Q 值。Double DQN，在主 DQN 上通过最大 Q 值选择 Action，再获取 Action 在 target DQN Q 值。主网选择 Action，targetDQN 生成 Action Q 值。被选择 Q 值，不一定总是最大，避免被高估次优 Action 总是超过最优 Action，导致发现不了真正最好 Action。学习目标公式： $Target = rt + \gamma \cdot Q_{target}(st+1, \arg\max_a(Q_{main}(st+1, a)))$ 。

第五个 Trick。Dueling DQN。Dueling DQN，Q 值函数  $Q(st, at)$  拆分，一部分静态环境状态具有价值  $V(st)$ ，Value；另一部分动态选择 Action 额外带来价值  $A(at)$ ，Advantage。公式， $Q(st, at) = V(st) + A(at)$ 。网络分别计算环境 Value 和选择 Action Advantage。Advantage，Action 与其他 Action 比较，零均值。网络最后，不再直接输出 Action 数量 Q 值，输出一个 Value，及 Action 数量 Advantage 值。V 值分别加到每个 Advantage 值上，得最后结果。让 DQN 学习目标更明确，如果当前期望价值主要由环境状态决定，Value 值大，所有 Advantage 波动不大；如果期望价值主要由 Action 决定，Value 值小，Advantage 波动大。分解让学习目标更稳定、精确，DQN 对环境状态估计能力更强。

### 33、SVM 所有核函数的了解应用，SVM 的损失函数

参考回答：

SVM 核函数：

#### 1 核函数本质

核函数的本质可以概括为如下三点：

1) 实际应用中，常常遇到线性不可分的情况。针对这种情况，常用做法是把样例特征映射到高维空间中，转化为线性可分问题。

2) 将样例特征映射到高维空间，可能会遇到维度过高的问题。

3) 针对可能的维灾难，可以利用核函数。核函数也是将特征从低维到高维的转换，但避免了直接进行高维空间中的复杂计算，可以在低维上进行计算，却能在实质上将分类效果表现在高维上。

当然，SVM 也能处理线性可分问题，这时使用的就是线性核了。

常用的核函数包括如下几个：线性核函数，多项式核函数，RBF 核函数(高斯核)，Sigmoid 核函数

#### 2 线性核

##### 2.1 线性核



SVM 肯定是可以处理线性问题的，这个就是斯坦福课程里讲 SVM 时候，最开始讲解的部分，以线性问题入手进行讲解。

线性核计算为  $k(x_1, x_2) = \langle x_1, x_2 \rangle$ ，其实就是在原始空间中的内积。

## 2.2 线性 SVM 和逻辑回归

当 SVM 使用线性核时，在处理分类问题时，和逻辑回归 比较有哪些异同，如何选择？

1) 线性核 SVM 和逻辑回归本质上没有区别

2)  $n$ =特征数， $m$ =训练样本数目

如果  $n$  相对  $m$  比较大，使用逻辑回归或者线性 SVM，例如  $n=10000$ ,  $m=10-1000$

如果  $n$  比较小， $m$  数值适中，使用高斯核的 SVM，例如  $n=1-1000$ ,  $m=10-10000$

如果  $n$  比较小， $m$  很大，构建更多特征，然后使用逻辑回归或者线性 SVM

## 3 非线性核

常用的非线性核包括多项式核，RBF 核（高斯核），sigmoid 核。

RBF 核通常是首选，实践中往往能表现出良好的性能。计算方法如下：

$$K(x, x_i) = \exp\left(-\frac{\|x - x_i\|^2}{\sigma^2}\right)$$

其中，如果  $\sigma$  选得很大的话，高次特征上的权重实际上衰减得非常快，所以实际上（数值上近似一下）相当于一个低维的子空间；反过来，如果  $\sigma$  选得很小，则可以将任意的数据映射为线性可分——当然，这并不一定是好事，因为随之而来的可能是非常严重的过拟合问题。不过，总的来说，通过调控参数，高斯核实际上具有相当高的灵活性，也是使用最广泛的核函数之一。

多项式核计算方法如下：
$$\kappa(x_1, x_2) = (\langle x_1, x_2 \rangle + R)^d$$

sigmoid 核函数计算方法如下：
$$K(x, x_i) = \tanh(\kappa(x, x_i) - \delta)$$

采用 Sigmoid 函数作为核函数时，支持向量机实现的就是一种多层感知器神经网络，应用 SVM 方法，隐含层节点数目（它确定神经网络的结构）、隐含层节点对输入节点的权值都是在设计（训练）的过程中自动确定的。而且支持向量机的理论基础决定了它最终求得的是全局最优值而不是局部最小值，也保证了它对于未知样本的良好泛化能力而不会出现过学习现象。

## 4 如何选择

1) 可以利用专家先验知识余弦选定核函数，例如已经知道问题是线性可分的，就可以使用线性核，不必选用非线性核

2) 利用交叉验证，试用不同的核函数，误差最小的即为效果最好的核函数



### 3) 混合核函数方法， 将不同的核函数结合起来

我觉得问题是线性可分和线性不可分其中之一，那在选择核函数的时候，如果不清楚问题属于哪一类，就两类核都尝试一下，所以可以主要尝试线性核以及 RBF 核。

1) Linear 核：主要用于线性可分的情形。参数少，速度快，对于一般数据，分类效果已经很理想了。

2) RBF 核：主要用于线性不可分的情形。参数多，分类结果非常依赖于参数。有很多人是通过训练数据的交叉验证来寻找合适的参数，不过这个过程比较耗时。

至于到底该采用哪种核，要根据具体问题，有的数据是线性可分的，有的不可分，需要多尝试不同核不同参数。如果特征的提取的好，包含的信息量足够大，很多问题都是线性可分的。当然，如果有足够的时间去寻找 RBF 核参数，应该能达到更好的效果。

没有一种黄金规则可以确定哪种核函数将推导出最准确的 SVM，在实践中，核函数的选择一般并不导致准确率的很大差别。SVM 总是发现全局解，而不是局部解。

### 5 其他

SVM 的复杂度主要由支持向量数刻画，而不是数据的维度，因此相比其他方法，SVM 不太容易过拟合 。

SVM 的损失函数：

#### 1、Hinge 损失函数

首先我们来看什么是合页损失函数 (hinge loss function)：

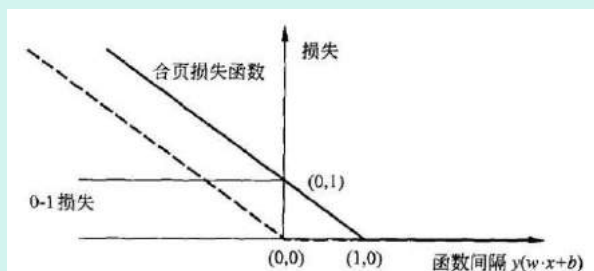
$$L(y \cdot (w \cdot x + b)) = [1 - y(w \cdot x + b)]_+$$

hinge loss function

下标 “+” 表示以下取正值的函数，我们用  $z$  表示中括号中的部分：

$$[z]_+ = \begin{cases} z, & z > 0 \\ 0, & z \leq 0 \end{cases}$$

也就是说，数据点如果被正确分类，损失为 0，如果没有被正确分类，损失为  $z$ 。合页损失函数如下图所示：





## 2、SVM 损失函数

SVM 的损失函数就是合页损失函数加上正则化项：

$$\sum_i^N [1 - y_i(w \cdot x_i + b)]_+ + \lambda \|w\|^2$$

损失函数

## 34、LR 和 SVM 区别

参考回答：

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。

## 35、朴素贝叶斯基本原理和预测过程

标签：朴素贝叶斯

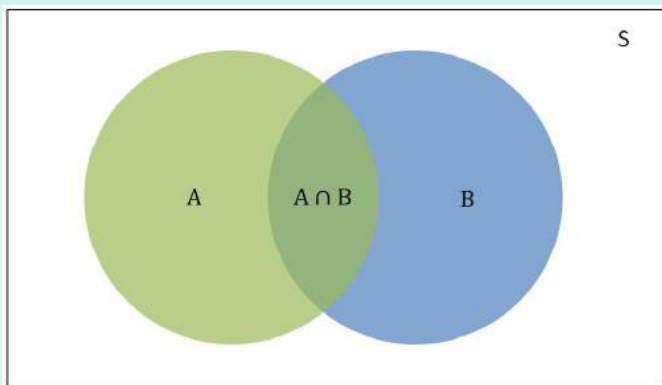
参考回答：

朴素贝叶斯分类和预测算法的原理

决策树和朴素贝叶斯是最常用的两种分类算法，本篇文章介绍朴素贝叶斯算法。贝叶斯定理是以英国数学家贝叶斯命名，用来解决两个条件概率之间的关系问题。简单的说就是在已知  $P(A|B)$  时如何获得  $P(B|A)$  的概率。朴素贝叶斯 (Naive Bayes) 假设特征  $P(A)$  在特定结果  $P(B)$  下是独立的。

### 1. 概率基础：

在开始介绍贝叶斯之前，先简单介绍下概率的基础知识。概率是某一结果出现的可能性。例如，抛一枚匀质硬币，正面向上的可能性多大？概率值是一个 0-1 之间的数字，用来衡量一个事件发生可能性的大小。概率值越接近 1，事件发生的可能性越大，概率值越接近 0，事件越不可能发生。我们日常生活中听到最多的是天气预报中的降水概率。概率的表示方法叫维恩图。下面我们通过维恩图来说明贝叶斯公式中常见的几个概率。



在维恩图中：

S：S 是样本空间，是所有可能事件的总和。

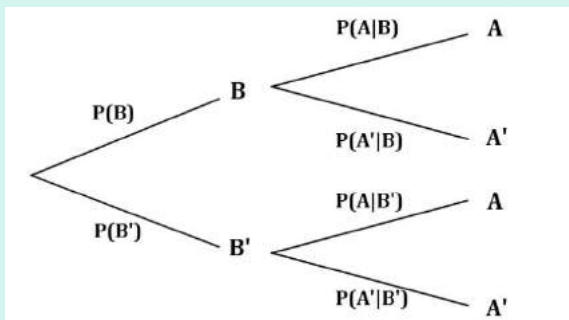
$P(A)$ ：是样本空间 S 中 A 事件发生的概率，维恩图中绿色的部分。

$P(B)$ ：是样本空间 S 中 B 事件发生的概率，维恩图中蓝色的部分。

$P(A \cap B)$ ：是样本空间 S 中 A 事件和 B 事件同时发生的概率，也就是 A 和 B 相交的区域。

$P(A|B)$ ：是条件概率，是 B 事件已经发生时 A 事件发生的概率。

对于条件概率，还有一种更清晰的表示方式叫概率树。下面的概率树表示了条件概率  $P(A|B)$ 。与维恩图中的  $P(A \cap B)$  相比，可以发现两者明显的区别。 $P(A \cap B)$  是事件 A 和事件 B 同时发现的情况，因此是两者相交区域的概率。而事件概率  $P(A|B)$  是事件 B 发生时事件 A 发生的概率。这里有一个先决条件就是  $P(B)$  要首先发生。



因为条件概率  $P(A|B)$  是在事件 B 已经发生的情况下，事件 A 发生的概率，因此  $P(A|B)$  可以表示为事件 A 与 B 的交集与事件 B 的比率。

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

该公式还可以转换为以下形式，以便我们下面进行贝叶斯公式计算时使用。

$$P(A \cap B) = P(A|B) \times P(B)$$

2. 贝叶斯公式：

贝叶斯算法通过已知的  $P(A|B)$ ,  $P(A)$ , 和  $P(B)$  三个概率计算  $P(B|A)$  发生的概率。假设我们现在已知  $P(A|B)$ ,  $P(A)$  和  $P(B)$  三个概率, 如何计算  $P(B|A)$  呢? 通过前面的概率树及  $P(A|B)$  的概率可知,  $P(B|A)$  的概率是在事件  $A$  发生的前提下事件  $B$  发生的概率, 因此  $P(B|A)$  可以表示为事件  $B$  与事件  $A$  的交集与事件  $A$  的比率。

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$


该公式同样可以转化为以下形式:

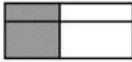
$$P(B \cap A) = P(B|A) \times P(A)$$

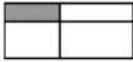
到这一步, 我们只需要证明  $P(A \cap B) = P(B \cap A)$  就可以证明在已知  $P(A|B)$  的情况下可以通过计算获得  $P(B|A)$  的概率。我们将概率树转化为下面的概率表, 分别列出  $P(A|B)$ ,  $P(B|A)$ ,  $P(A)$ , 和  $P(B)$  的概率。

Relative size	Case B	Case $\bar{B}$	Total
Condition A	w	x	w+x
Condition $\bar{A}$	y	z	y+z
Total	w+y	x+z	w+x+y+z


  

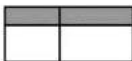


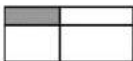




$$P(A|B) \times P(B) = \frac{w}{w+y} \times \frac{w+y}{w+x+y+z} = \frac{w}{w+x+y+z}$$
  







$$P(B|A) \times P(A) = \frac{w}{w+x} \times \frac{w+x}{w+x+y+z} = \frac{w}{w+x+y+z}$$

通过计算可以证明  $P(A|B) * P(B)$  和  $P(B|A) * P(A)$  最后求得的结果是概率表中的同一个区域的值, 因此:

$$P(A \cap B) = P(B \cap A)$$

我们通过  $P(A \cap B) = P(B \cap A)$  证明了在已知  $P(A|B)$ ,  $P(A)$ , 和  $P(B)$  三个概率的情况下可以计算出  $P(B|A)$  发生的概率。整个推导和计算过程可以说得通。但从统计学的角度来看,  $P(A|B)$  和  $P(B|A)$  两个条件概率之间存在怎样的关系呢? 我们从贝叶斯推断里可以找到答案。

### 3. 贝叶斯推断:

贝叶斯推断可以说明贝叶斯定理中两个条件概率之间的关系。换句话说就是我们为什么可以通过  $P(A|B)$ ,  $P(A)$ , 和  $P(B)$  三个概率计算出  $P(B|A)$  发生的概率。



$$P(B | A) = \frac{P(A | B) \times P(B)}{P(A)}$$

在贝叶斯推断中，每一种概率都有一个特定的名字：

$P(B)$  是”先验概率” (Prior probability)。

$P(A)$  是”先验概率” (Prior probability)，也作标准化常量(normalized constant)。

$P(A|B)$  是已知  $B$  发生后  $A$  的条件概率，叫做似然函数(likelihood)。

$P(B|A)$  是已知  $A$  发生后  $B$  的条件概率，是我们要求的值，叫做后验概率。

$P(A|B)/P(A)$  是调整因子，也被称作标准似然度 (standardised likelihood)。

$$P(B | A) = P(B) \times \frac{P(A | B)}{P(A)}$$

贝叶斯推断中有几个关键的概念需要说明下：

第一个是先验概率，先验概率是指我们主观通过事件发生次数对概率的判断。

第二个是似然函数，似然函数是对某件事发生可能性的判断，与条件概率正好相反。通过事件已经发生的概率推算事件可能性的概率。

维基百科中对似然函数与概率的解释：

概率：是给定某一参数值，求某一结果的可能性。

例如，抛一枚匀质硬币，抛 10 次，6 次正面向上的可能性多大？

似然函数：给定某一结果，求某一参数值的可能性。

例如，抛一枚硬币，抛 10 次，结果是 6 次正面向上，其是匀质的可能性多大？

第三个是调整因子：调整因子是似然函数与先验概率的比值，这个比值相当于一个权重，用来调整后验概率的值，使后验概率更接近真实概率。调整因子有三种情况，大于 1，等于 1 和小于 1。

调整因子  $P(A|B)/P(A) > 1$ ：说明事件可能发生的概率要大于事件已经发生次数的概率。

调整因子  $P(A|B)/P(A) = 1$ ：说明事件可能发生的概率与事件已经发生次数的概率相等。

调整因子  $P(A|B)/P(A) < 1$ ：说明事件可能发生的概率与事件小于已经发生次数的概率。

因此，贝叶斯推断可以理解为通过先验概率和调整因子来获得后验概率。其中调整因子是根据事件已经发生的概率推断事件可能发生的概率(通过硬币正面出现的次数来推断硬币均匀的可

能性)，并与已经发生的先验概率（硬币正面出现的概率）的比值。通过这个比值调整先验概率来获得后验概率。

后验概率 = 先验概率  $\times$  调整因子

### 36、LR 推导

标签：逻辑回归

参考回答：

逻辑回归本质上是线性回归，只是在特征到结果的映射中加入了一层逻辑函数  $g(z)$ ，即先把特征线性求和，然后使用函数  $g(z)$  作为假设函数来预测。 $g(z)$  可以将连续值映射到 0 和 1。 $g(z)$  为 sigmoid function.

$$g(z) = \frac{1}{1 + e^{-z}}$$

则

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

sigmoid function 的导数如下：

$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\ &= \frac{1}{(1 + e^{-z})^2} (e^{-z}) \\ &= \frac{1}{(1 + e^{-z})} \cdot \left(1 - \frac{1}{(1 + e^{-z})}\right) \\ &= g(z)(1 - g(z)). \end{aligned}$$

逻辑回归用来分类 0/1 问题，也就是预测结果属于 0 或者 1 的二值分类问题。这里假设了二值满足伯努利分布，也就是

$$\begin{aligned} P(y = 1 | x; \theta) &= h_{\theta}(x) \\ P(y = 0 | x; \theta) &= 1 - h_{\theta}(x) \end{aligned}$$

其也可以写成如下的形式：

$$p(y | x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$

对于训练数据集，特征数据  $x = \{x_1, x_2, \dots, x_m\}$  和对应的分类标签  $y = \{y_1, y_2, \dots, y_m\}$ ，假设  $m$  个样本是相互独立的，那么，极大似然函数为：



$$\begin{aligned} L(\theta) &= p(\vec{y} | X; \theta) \\ &= \prod_{i=1}^m p(y^{(i)} | x^{(i)}; \theta) \\ &= \prod_{i=1}^m (h_{\theta}(x^{(i)}))^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

log 似然为：

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})) \end{aligned}$$

如何使其最大呢？与线性回归类似，我们使用梯度上升的方法（求最小使用梯度下降），那么  $\theta := \theta + \alpha \nabla_{\theta} \ell(\theta)$ 。

$$\begin{aligned} \frac{\partial}{\partial \theta_j} \ell(\theta) &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x) (1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\ &= (y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x)) x_j \\ &= (y - h_{\theta}(x)) x_j \end{aligned}$$

如果只用一个训练样例  $(x, y)$ ，采用随机梯度上升规则，那么随机梯度上升更新规则为：

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

### 37、交叉熵

标签：决策树

参考回答：

为了更好的理解，需要了解的概率必备知识有：

大写字母  $X$  表示随机变量，小写字母  $x$  表示随机变量  $X$  的某个具体的取值；

$P(X)$  表示随机变量  $X$  的概率分布， $P(X, Y)$  表示随机变量  $X$ 、 $Y$  的联合概率分布， $P(Y|X)$  表示已知随机变量  $X$  的情况下随机变量  $Y$  的条件概率分布；

$p(X=x)$  表示随机变量  $X$  取某个具体值的概率，简记为  $p(x)$ ；

$p(X=x, Y=y)$  表示联合概率，简记为  $p(x, y)$ ， $p(Y=y|X=x)$  表示条件概率，简记为  $p(y|x)$ ，且有： $p(x, y) = p(x) * p(y|x)$ 。



熵：如果一个随机变量  $X$  的可能取值为  $X = \{x_1, x_2, \dots, x_k\}$ ，其概率分布为  $P(X=x_i)=p_i$  ( $i=1, 2, \dots, n$ )，则随机变量  $X$  的熵定义为：

$$H(X) = -\sum_x p(x) \log p(x)$$

把最前面的负号放到最后，便成了：

$$H(X) = \sum_x p(x) \log \frac{1}{p(x)}$$

上面两个熵的公式，无论用哪个都行，而且两者等价，一个意思（这两个公式在下文中都会用到）。

联合熵：两个随机变量  $X, Y$  的联合分布，可以形成联合熵 Joint Entropy，用  $H(X, Y)$  表示。

条件熵：在随机变量  $X$  发生的前提下，随机变量  $Y$  发生所新带来的熵定义为  $Y$  的条件熵，用  $H(Y|X)$  表示，用来衡量在已知随机变量  $X$  的条件下随机变量  $Y$  的不确定性。

且有此式子成立： $H(Y|X) = H(X, Y) - H(X)$ ，整个式子表示  $(X, Y)$  发生所包含的熵减去  $X$  单独发生包含的熵。至于怎么得来的请看推导：

$$\begin{aligned} H(X, Y) - H(X) &= -\sum_{x,y} p(x, y) \log p(x, y) + \sum_x p(x) \log p(x) \\ &= -\sum_{x,y} p(x, y) \log p(x, y) + \sum_x \left( \sum_y p(x, y) \right) \log p(x) \\ &= -\sum_{x,y} p(x, y) \log p(x, y) + \sum_{x,y} p(x, y) \log p(x) \\ &= -\sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)} \\ &= -\sum_{x,y} p(x, y) \log p(y|x) \end{aligned}$$

简单解释下上面的推导过程。整个式子共 6 行，其中

第二行推到第三行的依据是边缘分布  $p(x)$  等于联合分布  $p(x, y)$  的和；

第三行推到第四行的依据是把公因子  $\log p(x)$  乘进去，然后把  $x, y$  写在一起；

第四行推到第五行的依据是：因为两个  $\sigma$  都有  $p(x, y)$ ，故提取公因子  $p(x, y)$  放到外边，然后把里边的  $-\log p(x, y) - \log p(x)$  写成  $-\log(p(x, y)/p(x))$ ；

第五行推到第六行的依据是： $p(x, y) = p(x) * p(y|x)$ ，故  $p(x, y)/p(x) = p(y|x)$ 。

相对熵：又称互熵，交叉熵，鉴别信息，Kullback 熵，Kullback-Leibler 散度等。设  $p(x)$ 、 $q(x)$  是  $X$  中取值的两个概率分布，则  $p$  对  $q$  的相对熵是：

$$D(p \| q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = E_{p(x)} \log \frac{p(x)}{q(x)}$$

在一定程度上，相对熵可以度量两个随机变量的“距离”，且有  $D(p||q) \neq D(q||p)$ 。另外，值得一提的是， $D(p||q)$  是必然大于等于 0 的。

互信息：两个随机变量  $X, Y$  的互信息定义为  $X, Y$  的联合分布和各自独立分布乘积的相对熵，

用  $I(X, Y)$  表示：
$$I(X, Y) = \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

且有  $I(X, Y) = D(P(X, Y) || P(X)P(Y))$ 。下面，咱们来计算下  $H(Y) - I(X, Y)$  的结果，如下：

$$\begin{aligned} H(Y) - I(X, Y) &= -\sum_y p(y) \log p(y) - \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\ &= -\sum_y \left( \sum_x p(x, y) \right) \log p(y) - \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\ &= -\sum_{x,y} p(x, y) \log p(y) - \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\ &= -\sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)} \\ &= -\sum_{x,y} p(x, y) \log p(y|x) \\ &= H(Y|X) \end{aligned}$$

通过上面的计算过程，我们发现竟然有  $H(Y) - I(X, Y) = H(Y|X)$ 。故通过条件熵的定义，有： $H(Y|X) = H(X, Y) - H(X)$ ，而根据互信息定义展开得到  $H(Y|X) = H(Y) - I(X, Y)$ ，把前者跟后者结合起来，便有  $I(X, Y) = H(X) + H(Y) - H(X, Y)$ ，此结论被多数文献作为互信息的定义。

### 38、LR 公式

标签：逻辑回归

参考回答：

逻辑回归本质上是线性回归，只是在特征到结果的映射中加入了一层逻辑函数  $g(z)$ ，即先把特征线性求和，然后使用函数  $g(z)$  作为假设函数来预测。 $g(z)$  可以将连续值映射到 0 和 1。 $g(z)$  为 sigmoid function.

$$g(z) = \frac{1}{1 + e^{-z}}$$

则

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

sigmoid function 的导数如下：



$$\begin{aligned}
 g'(z) &= \frac{d}{dz} \frac{1}{1+e^{-z}} \\
 &= \frac{1}{(1+e^{-z})^2} (e^{-z}) \\
 &= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right) \\
 &= g(z)(1-g(z)).
 \end{aligned}$$

逻辑回归用来分类 0/1 问题，也就是预测结果属于 0 或者 1 的二值分类问题。这里假设了二值满足伯努利分布，也就是

$$\begin{aligned}
 P(y=1 | x; \theta) &= h_{\theta}(x) \\
 P(y=0 | x; \theta) &= 1 - h_{\theta}(x)
 \end{aligned}$$

其也可以写成如下的形式：

$$p(y | x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$

对于训练数据集，特征数据  $x=\{x_1, x_2, \dots, x_m\}$  和对应的分类标签  $y=\{y_1, y_2, \dots, y_m\}$ ，假设  $m$  个样本是相互独立的，那么，极大似然函数为：

$$\begin{aligned}
 L(\theta) &= p(\vec{y} | X; \theta) \\
 &= \prod_{i=1}^m p(y^{(i)} | x^{(i)}; \theta) \\
 &= \prod_{i=1}^m (h_{\theta}(x^{(i)}))^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}}
 \end{aligned}$$

log 似然为：

$$\begin{aligned}
 \ell(\theta) &= \log L(\theta) \\
 &= \sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))
 \end{aligned}$$

如何使其最大呢？与线性回归类似，我们使用梯度上升的方法（求最小使用梯度下降），那么  $\theta := \theta + \alpha \nabla_{\theta} \ell(\theta)$ 。

$$\begin{aligned}
 \frac{\partial}{\partial \theta_j} \ell(\theta) &= \left( y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\
 &= \left( y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) g(\theta^T x)(1-g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\
 &= (y(1-g(\theta^T x)) - (1-y)g(\theta^T x)) x_j \\
 &= (y - h_{\theta}(x)) x_j
 \end{aligned}$$



如果只用一个训练样例  $(x, y)$ ，采用随机梯度上升规则，那么随机梯度上升更新规则为：

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

### 39、交叉熵公式

标签：决策树

参考回答：

交叉熵：设  $p(x)$ 、 $q(x)$  是  $X$  中取值的两个概率分布，则  $p$  对  $q$  的相对熵是：

$$D(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = E_{p(x)} \log \frac{p(x)}{q(x)}$$

在一定程度上，相对熵可以度量两个随机变量的“距离”，且有  $D(p \parallel q) \neq D(q \parallel p)$ 。另外，值得一提的是， $D(p \parallel q)$  是必然大于等于 0 的。

互信息：两个随机变量  $X, Y$  的互信息定义为  $X, Y$  的联合分布和各自独立分布乘积的相对熵，

用  $I(X, Y)$  表示：
$$I(X, Y) = \sum_{x, y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

且有  $I(X, Y) = D(P(X, Y) \parallel P(X)P(Y))$ 。下面，咱们来计算下  $H(Y) - I(X, Y)$  的结果，如下：

$$\begin{aligned} H(Y) - I(X, Y) &= -\sum_y p(y) \log p(y) - \sum_{x, y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\ &= -\sum_y \left( \sum_x p(x, y) \right) \log p(y) - \sum_{x, y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\ &= -\sum_{x, y} p(x, y) \log p(y) - \sum_{x, y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\ &= -\sum_{x, y} p(x, y) \log \frac{p(x, y)}{p(x)} \\ &= -\sum_{x, y} p(x, y) \log p(y | x) \\ &= H(Y | X) \end{aligned}$$

## 2、处理回归问题常用算法

### 1、L1 和 L2 正则化的区别

考点:防过拟合

详情: 线性回归

参考回答:

L1 是模型各个参数的绝对值之和, L2 为各个参数平方和的开方值。L1 更趋向于产生少量的特征, 其它特征为 0, 最优的参数值很大概率出现在坐标轴上, 从而导致产生稀疏的权重矩阵, 而 L2 会选择更多的矩阵, 但是这些矩阵趋向于 0。

## 2、问题: Loss Function 有哪些, 怎么用?

知识点: 损失函数

详情: 线性回归

答案: 平方损失 (预测问题)、交叉熵 (分类问题)、hinge 损失 (SVM 支持向量机)、CART 回归树的残差损失

## 3、问题: 线性回归的表达式, 损失函数;

知识点: 线性回归

答案: 线性回归  $y=wx+b$ ,  $w$  和  $x$  可能是多维。线性回归的损失函数为平方损失函数。

解析: 一般会要求反向求导推导

## 4、线性回归的损失函数

考点: 损失函数

详情: 线性回归

参考回答:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2$$
$$\frac{\min}{\theta} J(\theta)$$

## 5、机器学习: 知道哪些传统机器学习模型

参考回答:





常见的机器学习算法：

1). 回归算法：回归算法是试图采用对误差的衡量来探索变量之间的关系的一类算法。回归算法是统计机器学习的利器。常见的回归算法包括：最小二乘法 (Ordinary Least Square)，逻辑回归 (Logistic Regression)，逐步式回归 (Stepwise Regression)，多元自适应回归样条 (Multivariate Adaptive Regression Splines) 以及本地散点平滑估计 (Locally Estimated Scatterplot Smoothing)。

2). 基于实例的算法：基于实例的算法常常用来对决策问题建立模型，这样的模型常常先选取一批样本数据，然后根据某些近似性把新数据与样本数据进行比较。通过这种方式来寻找最佳的匹配。因此，基于实例的算法常常也被称为“赢家通吃”学习或者“基于记忆的学习”。常见的算法包括 k-Nearest Neighbor (KNN)，学习矢量量化 (Learning Vector Quantization, LVQ)，以及自组织映射算法 (Self-Organizing Map, SOM)。深度学习的概念源于人工神经网络的研究。含多隐层的多层感知器就是一种深度学习结构。深度学习通过组合低层特征形成更加抽象的高层表示属性类别或特征，以发现数据的分布式特征表示。

3). 决策树学习：决策树算法根据数据的属性采用树状结构建立决策模型，决策树模型常常用来解决分类和回归问题。常见的算法包括：分类及回归树 (Classification And Regression Tree, CART)，ID3 (Iterative Dichotomiser 3)，C4.5，Chi-squared Automatic Interaction Detection (CHAID)，Decision Stump，随机森林 (Random Forest)，多元自适应回归样条 (MARS) 以及梯度推进机 (Gradient Boosting Machine, GBM)。

4). 贝叶斯方法：贝叶斯方法算法是基于贝叶斯定理的一类算法，主要用来解决分类和回归问题。常见算法包括：朴素贝叶斯算法，平均单依赖估计 (Averaged One-Dependence Estimators, AODE)，以及 Bayesian Belief Network (BBN)。

5). 基于核的算法：基于核的算法中最著名的莫过于支持向量机 (SVM) 了。基于核的算法把输入数据映射到一个高阶的向量空间，在这些高阶向量空间里，有些分类或者回归问题能够更容易的解决。常见的基于核的算法包括：支持向量机 (Support Vector Machine, SVM)，径向基函数 (Radial Basis Function, RBF)，以及线性判别分析 (Linear Discriminate Analysis, LDA) 等。

6). 聚类算法：聚类，就像回归一样，有时候人们描述的是一类问题，有时候描述的是一类算法。聚类算法通常按照中心点或者分层的方式对输入数据进行归并。所以的聚类算法都试图找到数据的内在结构，以便按照最大的共同点将数据进行归类。常见的聚类算法包括 k-Means 算法以及期望最大化算法 (Expectation Maximization, EM)。

7). 降低维度算法：像聚类算法一样，降低维度算法试图分析数据的内在结构，不过降低维度算法是以非监督学习的方式试图利用较少的信息来归纳或者解释数据。这类算法可以用于高维数据的可视化或者用来简化数据以便监督式学习使用。常见的算法包括：主成份分析 (Principle Component Analysis, PCA)，偏最小二乘回归 (Partial Least Square Regression, PLS)，Sammon 映射，多维尺度 (Multi-Dimensional Scaling, MDS)，投影追踪 (Projection Pursuit) 等。

8). 关联规则学习：关联规则学习通过寻找最能够解释数据变量之间关系的规则，来找出大量多元数据集中有用的关联规则。常见算法包括 Apriori 算法和 Eclat 算法等。



9). 集成算法: 集成算法用一些相对较弱的学习模型独立地就同样的样本进行训练, 然后把结果整合起来进行整体预测。集成算法的主要难点在于究竟集成哪些独立的较弱的学习模型以及如何把学习结果整合起来。这是一类非常强大的算法, 同时也非常流行。常见的算法包括:

Boosting, Bootstrapped Aggregation (Bagging), AdaBoost, 堆叠泛化 (Stacked Generalization, Blending), 梯度推进机 (Gradient Boosting Machine, GBM), 随机森林 (Random Forest)。

10). 人工神经网络: 人工神经网络算法模拟生物神经网络, 是一类模式匹配算法。通常用于解决分类和回归问题。人工神经网络是机器学习的一个庞大的分支, 有几百种不同的算法。(其中深度学习就是其中的一类算法, 我们会单独讨论), 重要的人工神经网络算法包括: 感知器神经网络 (Perceptron Neural Network), 反向传递 (Back Propagation), Hopfield 网络, 自组织映射 (Self-Organizing Map, SOM)。学习矢量量化 (Learning Vector Quantization, LVQ)。

### 3、处理聚类问题常用算法

#### 1、什么是 DBSCAN

考点: DBSCAN 的原理

详情: 基于密度聚类

擦考回答:

DBSCAN 是一种基于密度的空间聚类算法, 它不需要定义簇的个数, 而是将具有足够高密度的区域划分为簇, 并在有噪声的数据中发现任意形状的簇, 在此算法中将簇定义为密度相连的点的最大集合。

#### 2、k-means 算法流程

考点: 聚类算法

详情: K 均值

参考回答:

从数据集中随机选择  $k$  个聚类样本作为初始的聚类中心, 然后计算数据集中每个样本到这  $k$  个聚类中心的距离, 并将此样本分到距离最小的聚类中心所对应的类中。将所有样本归类后, 对于每个类别重新计算每个类别的聚类中心即每个类中所有样本的质心, 重复以上操作直到聚类中心不变为止。

#### 3、LDA 的原理

详情: LDA

考点: 降维技术





参考回答：

LDA 是一种基于有监督学习的降维方式,将数据集在低维度的空间进行投影,要使得投影后的同类别的数据点间的距离尽可能的靠近,而不同类别间的数据点的距离尽可能的远。

#### 4、介绍几种机器学习的算法，我就结合我的项目经理介绍了些 RF，Kmeans 等算法。

参考回答：

常见的机器学习算法：

1) . 回归算法：回归算法是试图采用对误差的衡量来探索变量之间的关系的一类算法。回归算法是统计机器学习的利器。常见的回归算法包括：最小二乘法（Ordinary Least Square），逻辑回归（Logistic Regression），逐步式回归（Stepwise Regression），多元自适应回归样条（Multivariate Adaptive Regression Splines）以及本地散点平滑估计（Locally Estimated Scatterplot Smoothing）。

2) . 基于实例的算法：基于实例的算法常常用来对决策问题建立模型，这样的模型常常先选取一批样本数据，然后根据某些近似性把新数据与样本数据进行比较。通过这种方式来寻找最佳的匹配。因此，基于实例的算法常常也被称为“赢家通吃”学习或者“基于记忆的学习”。常见的算法包括 k-Nearest Neighbor (KNN)，学习矢量量化（Learning Vector Quantization, LVQ），以及自组织映射算法（Self-Organizing Map, SOM）。深度学习的概念源于人工神经网络的研究。含多隐层的多层感知器就是一种深度学习结构。深度学习通过组合低层特征形成更加抽象的高层表示属性类别或特征，以发现数据的分布式特征表示。

3) . 决策树学习：决策树算法根据数据的属性采用树状结构建立决策模型，决策树模型常常用来解决分类和回归问题。常见的算法包括：分类及回归树（Classification And Regression Tree, CART），ID3（Iterative Dichotomiser 3），C4.5，Chi-squared Automatic Interaction Detection (CHAID)，Decision Stump，随机森林（Random Forest），多元自适应回归样条（MARS）以及梯度推进机（Gradient Boosting Machine, GBM）。

4) . 贝叶斯方法：贝叶斯方法算法是基于贝叶斯定理的一类算法，主要用来解决分类和回归问题。常见算法包括：朴素贝叶斯算法，平均单依赖估计（Averaged One-Dependence Estimators, AODE），以及 Bayesian Belief Network (BBN）。

5) . 基于核的算法：基于核的算法中最著名的莫过于支持向量机（SVM）了。基于核的算法把输入数据映射到一个高阶的向量空间，在这些高阶向量空间里，有些分类或者回归问题能够更容易的解决。常见的基于核的算法包括：支持向量机（Support Vector Machine, SVM），径向基函数（Radial Basis Function, RBF），以及线性判别分析（Linear Discriminate Analysis, LDA）等。

6) . 聚类算法：聚类，就像回归一样，有时候人们描述的是一类问题，有时候描述的是一类算法。聚类算法通常按照中心点或者分层的方式对输入数据进行归并。所以的聚类算法都试图找到数据的内在结构，以便按照最大的共同点将数据进行归类。常见的聚类算法包括 k-Means 算法以及期望最大化算法（Expectation Maximization, EM）。



7). 降低维度算法: 像聚类算法一样, 降低维度算法试图分析数据的内在结构, 不过降低维度算法是以非监督学习的方式试图利用较少的信息来归纳或者解释数据。这类算法可以用于高维数据的可视化或者用来简化数据以便监督式学习使用。常见的算法包括: 主成份分析

(Principle Component Analysis, PCA), 偏最小二乘回归 (Partial Least Square Regression, PLS), Sammon 映射, 多维尺度 (Multi-Dimensional Scaling, MDS), 投影追踪 (Projection Pursuit) 等。

8). 关联规则学习: 关联规则学习通过寻找最能够解释数据变量之间关系的规则, 来找出大量多元数据集中有用的关联规则。常见算法包括 Apriori 算法和 Eclat 算法等。

9). 集成算法: 集成算法用一些相对较弱的学习模型独立地就同样的样本进行训练, 然后把结果整合起来进行整体预测。集成算法的主要难点在于究竟集成哪些独立的较弱的学习模型以及如何把学习结果整合起来。这是一类非常强大的算法, 同时也非常流行。常见的算法包括: Boosting, Bootstrapped Aggregation (Bagging), AdaBoost, 堆叠泛化 (Stacked Generalization, Blending), 梯度推进机 (Gradient Boosting Machine, GBM), 随机森林 (Random Forest)。

10). 人工神经网络: 人工神经网络算法模拟生物神经网络, 是一类模式匹配算法。通常用于解决分类和回归问题。人工神经网络是机器学习的一个庞大的分支, 有几百种不同的算法。(其中深度学习就是其中的一类算法, 我们会单独讨论), 重要的人工神经网络算法包括: 感知器神经网络 (Perceptron Neural Network), 反向传递 (Back Propagation), Hopfield 网络, 自组织映射 (Self-Organizing Map, SOM)。学习矢量量化 (Learning Vector Quantization, LVQ)。

RF: 通过对训练数据样本以及属性进行有放回的抽样 (针对某一个属性随机选择样本) 这里有两种, 一种是每次都是有放回的采样, 有些样本是重复的, 组成和原始数据集样本个数一样的数据集; 另外一种是不放回的抽样, 抽取大约 60% 的训练信息。由此生成一颗 CART 树, 剩下的样本信息作为袋外数据, 用来当作验证集计算袋外误差测试模型; 把抽取出的样本信息再放回原数据集中, 再重新抽取一组训练信息, 再以此训练数据集生成一颗 CART 树。这样依次生成多颗 CART 树, 多颗树组成森林, 并且他们的生成都是通过随机采样的训练数据生成, 因此叫随机森林。RF 可以用于数据的回归, 也可以用于数据的分类。回归时是由多颗树的预测结果求均值; 分类是由多棵树的预测结果进行投票。正式由于它的随机性, RF 有极强的防止过拟合的特性。由于他是由 CART 组成, 因此它的训练数据不需要进行归一化, 因为每棵的建立过程都是通过选择一个能最好的对数据样本进行选择的属性来建立分叉, 因此有以上好处的同时也带来了一个缺点, 那就是忽略了属性与属性之间的关系。

K-means: 基本 K-Means 算法的思想很简单, 事先确定常数 K, 常数 K 意味着最终的聚类类别数, 首先随机选定初始点为质心, 并通过计算每一个样本与质心之间的相似度 (这里为欧式距离), 将样本点归到最相似的类中, 接着, 重新计算每个类的质心 (即为类中心), 重复这样的过程, 知道质心不再改变, 最终就确定了每个样本所属的类别以及每个类的质心。由于每次都要计算所有的样本与每一个质心之间的相似度, 故在大规模的数据集上, K-Means 算法的收敛速度比较慢。

初始化常数 K, 随机选取初始点为质心

重复计算一下过程, 直到质心不再改变

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类中

重新计算质心



输出最终的质心以及每个类

## 5、KMeans 讲讲，KMeans 有什么缺点，K 怎么确定

参考回答：

在 k-means 算法中，用质心来表示 cluster；且容易证明 k-means 算法收敛等同于所有质心不再发生变化。基本的 k-means 算法流程如下：

选取 k 个初始质心（作为初始 cluster）；

repeat：对每个样本点，计算得到距其最近的质心，将其类别标为该质心所对应的 cluster；重新计算 k 个 cluster 对应的质心；

until 质心不再发生变化

k-means 存在缺点：

1) k-means 是局部最优的，容易受到初始质心的影响；比如在下图中，因选择初始质心不恰当而造成次优的聚类结果。

2) 同时，k 值的选取也会直接影响聚类结果，最优聚类的 k 值应与样本数据本身的结构信息相吻合，而这种结构信息是很难去掌握，因此选取最优 k 值是非常困难的。

K 值得确定：

法 1：（轮廓系数）在实际应用中，由于 Kmean 一般作为数据预处理，或者用于辅助分聚类贴标签。所以 k 一般不会设置很大。可以通过枚举，令 k 从 2 到一个固定值如 10，在每个 k 值上重复运行数次 kmeans（避免局部最优解），并计算当前 k 的平均轮廓系数，最后选取轮廓系数最大的值对应的 k 作为最终的集群数目。

法 2：（Calinski-Harabasz 准则）

$$VRC_k = \frac{SSB}{SSW} * (N - k) / (k - 1)$$

其中 SSB 是类间方差， $SSB = \sum_{i=1}^k n_i ||m_i - m||^2$ ，m 为所有点的中心点， $m_i$  为某类的中心点；

SSW 是类内方差， $SSW = \sum_{i=1}^k \sum_{x \in c_i} ||x - m_i||^2$ ；

$(N - k) / (k - 1)$  是复杂度；

$VRC_k$  比率越大，数据分离度越大。

## 6、Kmeans



参考回答：

基本 K-Means 算法的思想很简单，事先确定常数 K，常数 K 意味着最终的聚类类别数，首先随机选定初始点为质心，并通过计算每一个样本与质心之间的相似度(这里为欧式距离)，将样本点归到最相似的类中，接着，重新计算每个类的质心(即为类中心)，重复这样的过程，知道质心不再改变，最终就确定了每个样本所属的类别以及每个类的质心。由于每次都要计算所有的样本与每一个质心之间的相似度，故在大规模的数据集上，K-Means 算法的收敛速度比较慢。

初始化常数 K，随机选取初始点为质心

重复计算一下过程，直到质心不再改变

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类中

重新计算质心

输出最终的质心以及每个类

## 7、DBSCAN 原理和算法伪代码，与 kmeans，OPTICS 区别

参考回答：

DBSCAN 聚类算法原理

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) 聚类算法，它是一种基于高密度连通区域的、基于密度的聚类算法，能够将具有足够高密度的区域划分为簇，并在具有噪声的数据中发现任意形状的簇。我们总结一下 DBSCAN 聚类算法原理的基本要点：

DBSCAN 算法需要选择一种距离度量，对于待聚类的数据集中，任意两个点之间的距离，反映了点之间的密度，说明了点与点是否能够聚到同一类中。由于 DBSCAN 算法对高维数据定义密度很困难，所以对于二维空间中的点，可以使用欧几里德距离来进行度量。

DBSCAN 算法需要用户输入 2 个参数：一个参数是半径 (Eps)，表示以给定点 P 为中心的圆形邻域的范围；另一个参数是以点 P 为中心的邻域内最少点的数量 (MinPts)。如果满足：以点 P 为中心、半径为 Eps 的邻域内的点的个数不少于 MinPts，则称点 P 为核心点。

DBSCAN 聚类使用到一个 k-距离的概念，k-距离是指：给定数据集  $P=\{p(i); i=0, 1, \dots, n\}$ ，对于任意点  $P(i)$ ，计算点  $P(i)$  到集合 D 的子集  $S=\{p(1), p(2), \dots, p(i-1), p(i+1), \dots, p(n)\}$  中所有点之间的距离，距离按照从小到大的顺序排序，假设排序后的距离集合为  $D=\{d(1), d(2), \dots, d(k-1), d(k), d(k+1), \dots, d(n)\}$ ，则  $d(k)$  就被称为 k-距离。也就是说，k-距离是点  $p(i)$  到所有点 (除了  $p(i)$  点) 之间距离第 k 近的距离。对待聚类集合中每个点  $p(i)$  都计算 k-距离，最后得到所有点的 k-距离集合  $E=\{e(1), e(2), \dots, e(n)\}$ 。

根据经验计算半径 Eps：根据得到的所有点的 k-距离集合 E，对集合 E 进行升序排序后得到 k-距离集合  $E'$ ，需要拟合一条排序后的  $E'$  集合中 k-距离的变化曲线图，然后绘出曲线，通过观察，将急剧发生变化的位置所对应的 k-距离的值，确定为半径 Eps 的值。



根据经验计算最少点的数量 MinPts: 确定 MinPts 的大小, 实际上也是确定 k-距离中 k 的值, DBSCAN 算法取  $k=4$ , 则  $\text{MinPts}=4$ 。

另外, 如果觉得经验值聚类的结果不满意, 可以适当调整 Eps 和 MinPts 的值, 经过多次迭代计算对比, 选择最合适的参数值。可以看出, 如果 MinPts 不变, Eps 取得值过大, 会导致大多数点都聚到同一个簇中, Eps 过小, 会导致已一个簇的分裂; 如果 Eps 不变, MinPts 的值取得过大, 会导致同一个簇中点被标记为噪声点, MinPts 过小, 会导致发现大量的核心点。

我们需要知道的是, DBSCAN 算法, 需要输入 2 个参数, 这两个参数的计算都来自经验知识。半径 Eps 的计算依赖于计算 k-距离, DBSCAN 取  $k=4$ , 也就是设置  $\text{MinPts}=4$ , 然后需要根据 k-距离曲线, 根据经验观察找到合适的半径 Eps 的值, 下面的算法实现过程中, 我们会详细说明。对于算法的实现, 首先我们概要地描述一下实现的过程:

1) 解析样本数据文件。2) 计算每个点与其他所有点之间的欧几里德距离。3) 计算每个点的 k-距离值, 并对所有点的 k-距离集合进行升序排序, 输出的排序后的 k-距离值。4) 将所有点的 k-距离值, 在 Excel 中用散点图显示 k-距离变化趋势。5) 根据散点图确定半径 Eps 的值。根据给定  $\text{MinPts}=4$ , 以及半径 Eps 的值, 计算所有核心点, 并建立核心点到到核心点距离小于半径 Eps 的点的映射。7) 根据得到的核心点集合, 以及半径 Eps 的值, 计算能够连通的核心点, 得到噪声点。8) 将能够连通的每一组核心点, 以及到核心点距离小于半径 Eps 的点, 都放到一起, 形成一个簇。9) 选择不同的半径 Eps, 使用 DBSCAN 算法聚类得到的一组簇及其噪声点, 使用散点图对比聚类效果。

算法伪代码:

算法描述:

算法: DBSCAN

输入: E——半径

MinPts——给定点在 E 邻域内成为核心对象的最小邻域点数。

D——集合。

输出: 目标类簇集合

方法: Repeat

1) 判断输入点是否为核心对象

2) 找出核心对象的 E 邻域中的所有直接密度可达点。

Until 所有输入点都判断完毕

Repeat

针对所有核心对象的 E 邻域内所有直接密度可达点找到最大密度相连对象集合, 中间涉及到一些密度可达对象的合并。Until 所有核心对象的 E 邻域都遍历完毕





DBSCAN 和 Kmeans 的区别：

- 1) K 均值和 DBSCAN 都是将每个对象指派到单个簇的划分聚类算法，但是 K 均值一般聚类所有对象，而 DBSCAN 丢弃被它识别为噪声的对象。
- 2) 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。
- 3) K 均值很难处理非球形的簇和不同大小的簇。DBSCAN 可以处理不同大小或形状的簇，并且不太受噪声和离群点的影响。当簇具有很不相同的密度时，两种算法的性能都很差。
- 4) K 均值只能用于具有明确定义的质心（比如均值或中位数）的数据。DBSCAN 要求密度定义（基于传统的欧几里得密度概念）对于数据是有意义的。
- 5) K 均值可以用于稀疏的高维数据，如文档数据。DBSCAN 通常在这类数据上的性能很差，因为对于高维数据，传统的欧几里得密度定义不能很好处理它们。
- 6) K 均值和 DBSCAN 的最初版本都是针对欧几里得数据设计的，但是它们都被扩展，以便处理其他类型的数据。
- 7) 基本 K 均值算法等价于一种统计聚类方法（混合模型），假定所有的簇都来自球形高斯分布，具有不同的均值，但具有相同的协方差矩阵。DBSCAN 不对数据的分布做任何假定。
- 8) K 均值 DBSCAN 和都寻找使用所有属性的簇，即它们都不寻找可能只涉及某个属性子集的簇。
- 9) 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。
- 10) K 均值算法的时间复杂度是  $O(m)$ ，而 DBSCAN 的时间复杂度是  $O(m^2)$ ，除非用于诸如低维欧几里得数据这样的特殊情况。
- 11) DBSCAN 多次运行产生相同的结果，而 K 均值通常使用随机初始化质心，不会产生相同的结果。
- 12) DBSCAN 自动地确定簇个数，对于 K 均值，簇个数需要作为参数指定。然而，DBSCAN 必须指定另外两个参数：Eps（邻域半径）和 MinPts（最少点数）。
- 13) K 均值聚类可以看作优化问题，即最小化每个点到最近质心的误差平方和，并且可以看作一种统计聚类（混合模型）的特例。DBSCAN 不基于任何形式化模型。

DBSCAN 与 OPTICS 的区别：

DBSCAN 算法，有两个初始参数 E（邻域半径）和 minPts（E 邻域最小点数）需要用户手动设置输入，并且聚类的类簇结果对这两个参数的取值非常敏感，不同的取值将产生不同的聚类结果，其实这也是大多数其他需要初始化参数聚类算法的弊端。

为了克服 DBSCAN 算法这一缺点，提出了 OPTICS 算法（Ordering Points to identify the clustering structure）。OPTICS 并不显示的产生结果类簇，而是为聚类分析生成一个增广的簇排序（比如，以可达距离为纵轴，样本点输出次序为横轴的坐标图），这个排序代表了各样本

点基于密度的聚类结构。它包含的信息等价于从一个广泛的参数设置所获得的基于密度的聚类，换句话说，从这个排序中可以得到基于任何参数  $E$  和  $\minPts$  的 DBSCAN 算法的聚类结果。

OPTICS 两个概念：

核心距离：对象  $p$  的核心距离是指是  $p$  成为核心对象的最小  $E'$ 。如果  $p$  不是核心对象，那么  $p$  的核心距离没有任何意义。

可达距离：对象  $q$  到对象  $p$  的可达距离是指  $p$  的核心距离和  $p$  与  $q$  之间欧几里得距离之间的较大值。如果  $p$  不是核心对象， $p$  和  $q$  之间的可达距离没有意义。

算法描述：OPTICS 算法额外存储了每个对象的核心距离和可达距离。基于 OPTICS 产生的排序信息来提取类簇。

## 4、推荐系统的常用算法

### 1、 问推荐算法，fm，lr，embedding

标签：FM

参考回答：

推荐算法：

基于人口学的推荐、基于内容的推荐、基于用户的协同过滤推荐、基于项目的协同过滤推荐、基于模型的协同过滤推荐、基于关联规则的推荐

FM:

$$y(\mathbf{x}) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

LR:

逻辑回归本质上是线性回归，只是在特征到结果的映射中加入了一层逻辑函数  $g(z)$ ，即先把特征线性求和，然后使用函数  $g(z)$  作为假设函数来预测。 $g(z)$  可以将连续值映射到 0 和 1。 $g(z)$  为 sigmoid function.

$$g(z) = \frac{1}{1 + e^{-z}}$$

则

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$





sigmoid function 的导数如下：

$$\begin{aligned}g'(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\&= \frac{1}{(1 + e^{-z})^2} (e^{-z}) \\&= \frac{1}{(1 + e^{-z})} \cdot \left(1 - \frac{1}{(1 + e^{-z})}\right) \\&= g(z)(1 - g(z)).\end{aligned}$$

逻辑回归用来分类 0/1 问题，也就是预测结果属于 0 或者 1 的二值分类问题。这里假设了二值满足伯努利分布，也就是

$$\begin{aligned}P(y = 1 \mid x; \theta) &= h_{\theta}(x) \\P(y = 0 \mid x; \theta) &= 1 - h_{\theta}(x)\end{aligned}$$

其也可以写成如下的形式：

$$p(y \mid x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$

对于训练数据集，特征数据  $x = \{x_1, x_2, \dots, x_m\}$  和对应的分类标签  $y = \{y_1, y_2, \dots, y_m\}$ ，假设  $m$  个样本是相互独立的，那么，极大似然函数为：

$$\begin{aligned}L(\theta) &= p(\vec{y} \mid X; \theta) \\&= \prod_{i=1}^m p(y^{(i)} \mid x^{(i)}; \theta) \\&= \prod_{i=1}^m (h_{\theta}(x^{(i)}))^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}}\end{aligned}$$

log 似然为：

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\&= \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))\end{aligned}$$

如何使其最大呢？与线性回归类似，我们使用梯度上升的方法（求最小使用梯度下降），那么  $\theta := \theta + \alpha \nabla_{\theta} \ell(\theta)$ 。

$$\begin{aligned}\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left( y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left( y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) g(\theta^T x)(1-g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\ &= (y(1-g(\theta^T x)) - (1-y)g(\theta^T x)) x_j \\ &= (y - h_\theta(x)) x_j\end{aligned}$$

如果只用一个训练样例  $(x, y)$ ，采用随机梯度上升规则，那么随机梯度上升更新规则为：

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

Embedding:

Embedding 在数学上表示一个 mapping:  $f: X \rightarrow Y$ ，也就是一个 function。其中该函数满足两个性质：1) injective (单射的)：就是我们所说的单射函数，每个  $Y$  只有唯一的  $X$  对应；2) structure-preserving (结构保存)：比如在  $X$  所属的空间上  $x_1 \leq x_2$ ，那么映射后在  $Y$  所属空间上同理  $y_1 \leq y_2$ 。

那么对于 word embedding, 就是找到一个映射(函数)将单词(word)映射到另外一个空间(其中这个映射具有 injective 和 structure-preserving 的特点), 生成在一个新的空间上的表达, 该表达就是 word representation。

## 2、协同过滤的 itemCF, userCF 区别适用场景

标签：协同过滤算法

参考回答：

Item CF 和 User CF 两个方法都能很好的给出推荐，并可以达到不错的效果。但是他们之间还是有不同之处的，而且适用性也有区别。下面进行一下对比

计算复杂度：

Item CF 和 User CF 是基于协同过滤推荐的两个最基本的算法，User CF 是很早以前就提出来了，Item CF 是从 Amazon 的论文和专利发表之后（2001 年左右）开始流行，大家都觉得 Item CF 从性能和复杂度上比 User CF 更优，其中的一个主要原因就是对于一个在线网站，用户的数量往往大大超过物品的数量，同时物品的数据相对稳定，因此计算物品的相似度不但计算量较小，同时也不必频繁更新。但我们往往忽略了这种情况只适应于提供商品的电子商务网站，对于新闻，博客或者微内容的推荐系统，情况往往是相反的，物品的数量是海量的，同时也是更新频繁的，所以单从复杂度的角度，这两个算法在不同的系统中各有优势，推荐引擎的设计者需要根据自己的应用的特点选择更加合适的算法。

适用场景：

在非社交网络的网站中，内容内在的联系是很重要的推荐原则，它比基于相似用户的推荐原则更加有效。比如在购书网站上，当你看一本书的时候，推荐引擎会给你推荐相关的书籍，这个



推荐的重要性远远超过了网站首页对该用户的综合推荐。可以看到，在这种情况下，Item CF 的推荐成为了引导用户浏览的重要手段。同时 Item CF 便于为推荐做出解释，在一个非社交网络的网站中，给某个用户推荐一本书，同时给出的解释是某某和你有相似兴趣的人也看了这本书，这很难让用户信服，因为用户可能根本不认识那个人；但如果解释说是因为这本书和你以前看的某本书相似，用户可能就觉得合理而采纳了此推荐。

相反的，在现今很流行的社交网络站点中，User CF 是一个更不错的选择，User CF 加上社会网络信息，可以增加用户对推荐解释的信服程度。

### 3、推荐系统的大概步骤，解决冷启动。。。

标签: 协同过滤算法

参考回答:

步骤: 1) 收集用户的所有信息。2) 使用大数据计算平台对收集的信息进行处理，得到用户偏好数据。3) 将偏好数据导入喜好类型计算算法中进行预算计算，得到预算结果。4) 将推荐的结果导入数据库(redis、hbase)。5) 发开一个推荐引擎，对外开放接口，输出推荐结果。

解决冷启动的方案:

#### 1) 提供非个性化的推荐

最简单的例子就是提供热门排行榜，可以给用户推荐热门排行榜，等到用户数据收集到一定的时候，再切换为个性化推荐。例如 Netflix 的研究也表明新用户冷启动阶段确实是更倾向于热门排行榜的，老用户会更加需要长尾推荐

#### 2) 利用用户注册信息

用户的注册信息主要分为 3 种: (1) 获取用户的注册信息; (2) 根据用户的注册信息对用户分类; (3) 给用户推荐他所属分类中用户喜欢的物品。

#### 3) 选择合适的物品启动用户的兴趣

用户在登录时对一些物品进行反馈，收集用户对这些物品的兴趣信息，然后给用户推荐那些和这些物品相似的物品。一般来说，能够用来启动用户兴趣的物品需要具有以下特点:

比较热门，如果要用用户对物品进行反馈，前提是用户得知道这是什么东西;

具有代表性和区分性，启动用户兴趣的物品不能是大众化或老少咸宜的，因为这样的物品对用户的兴趣没有区分性;

启动物品集合需要有多多样性，在冷启动时，我们不知道用户的兴趣，而用户兴趣的可能性非常多，为了匹配多样的兴趣，我们需要提供具有很高覆盖率的启动物品集合，这些物品能覆盖几乎所有主流的用户兴趣

#### 4) 利用物品的内容信息



用来解决物品的冷启动问题，即如何将新加入的物品推荐给对它感兴趣的用户。物品冷启动问题在新闻网站等时效性很强的网站中非常重要，因为这些网站时时刻刻都有新物品加入，而且每个物品必须能够再第一时间展现给用户，否则经过一段时间后，物品的价值就大大降低了。

#### 5) 采用专家标注

很多系统在建立的时候，既没有用户的行为数据，也没有充足的物品内容信息来计算物品相似度。这种情况下，很多系统都利用专家进行标注。

#### 6) 利用用户在其他地方已经沉淀的数据进行冷启动

以 QQ 音乐举例：QQ 音乐的猜你喜欢电台想要去猜测第一次使用 QQ 音乐的用户的口味偏好，一大优势是可以利用其它腾讯平台的数据，比如在 QQ 空间关注了谁，在腾讯微博关注了谁，更进一步，比如在腾讯视频刚刚看了一部动漫，那么如果 QQ 音乐推荐了这部动漫里的歌曲，用户会觉得很人性化。这就是利用用户在其它平台已有的数据。

再比如今日头条：它是在用户通过新浪微博等社交网站登录之后，获取用户的关注列表，并且爬取用户最近参与互动的 feed（转发/评论等），对其进行语义分析，从而获取用户的偏好。

所以这种方法的前提是，引导用户通过社交网络账号登录，这样一方面可以降低注册成本提高转化率；另一方面可以获取用户的社交网络信息，解决冷启动问题。

#### 7) 利用用户的手机等兴趣偏好进行冷启动

Android 手机开放的比较高，所以在安装自己的 app 时，就可以顺路了解下手机上还安装了什么其他的 app。比如一个用户安装了美丽说、蘑菇街、辣妈帮、大姨妈等应用，就可以判定这是女性了，更进一步还可以判定是备孕还是少女。目前读取用户安装的应用这部分功能除了 app 应用商店之外，一些新闻类、视频类的应用也在做，对于解决冷启动问题有很好的帮助。

### 4、传统的机器学习算法了解吗

参考回答：

常见的机器学习算法：

1) . 回归算法：回归算法是试图采用对误差的衡量来探索变量之间的关系的一类算法。回归算法是统计机器学习的利器。常见的回归算法包括：最小二乘法（Ordinary Least Square），逻辑回归（Logistic Regression），逐步式回归（Stepwise Regression），多元自适应回归样条（Multivariate Adaptive Regression Splines）以及本地散点平滑估计（Locally Estimated Scatterplot Smoothing）。

2) . 基于实例的算法：基于实例的算法常常用来对决策问题建立模型，这样的模型常常先选取一批样本数据，然后根据某些近似性把新数据与样本数据进行比较。通过这种方式来寻找最佳的匹配。因此，基于实例的算法常常也被称为“赢家通吃”学习或者“基于记忆的学习”。常见的算法包括 k-Nearest Neighbor (KNN)，学习矢量量化（Learning Vector Quantization, LVQ），以及自组织映射算法（Self-Organizing Map, SOM）。深度学习的概念源于人工神经网络的研究。含多隐层的多层感知器就是一种深度学习结构。深度学习通过组合低层特征形成更加抽象的高层表示属性类别或特征，以发现数据的分布式特征表示。



3). 决策树学习: 决策树算法根据数据的属性采用树状结构建立决策模型, 决策树模型常常用来解决分类和回归问题。常见的算法包括: 分类及回归树 (Classification And Regression Tree, CART), ID3 (Iterative Dichotomiser 3), C4.5, Chi-squared Automatic Interaction Detection (CHAID), Decision Stump, 随机森林 (Random Forest), 多元自适应回归样条 (MARS) 以及梯度推进机 (Gradient Boosting Machine, GBM)。

4). 贝叶斯方法: 贝叶斯方法算法是基于贝叶斯定理的一类算法, 主要用来解决分类和回归问题。常见算法包括: 朴素贝叶斯算法, 平均单依赖估计 (Averaged One-Dependence Estimators, AODE), 以及 Bayesian Belief Network (BBN)。

5). 基于核的算法: 基于核的算法中最著名的莫过于支持向量机 (SVM) 了。基于核的算法把输入数据映射到一个高阶的向量空间, 在这些高阶向量空间里, 有些分类或者回归问题能够更容易的解决。常见的基于核的算法包括: 支持向量机 (Support Vector Machine, SVM), 径向基函数 (Radial Basis Function, RBF), 以及线性判别分析 (Linear Discriminate Analysis, LDA) 等。

6). 聚类算法: 聚类, 就像回归一样, 有时候人们描述的是一类问题, 有时候描述的是一类算法。聚类算法通常按照中心点或者分层的方式对输入数据进行归并。所以的聚类算法都试图找到数据的内在结构, 以便按照最大的共同点将数据进行归类。常见的聚类算法包括 k-Means 算法以及期望最大化算法 (Expectation Maximization, EM)。

7). 降低维度算法: 像聚类算法一样, 降低维度算法试图分析数据的内在结构, 不过降低维度算法是以非监督学习的方式试图利用较少的信息来归纳或者解释数据。这类算法可以用于高维数据的可视化或者用来简化数据以便监督式学习使用。常见的算法包括: 主成份分析 (Principle Component Analysis, PCA), 偏最小二乘回归 (Partial Least Square Regression, PLS), Sammon 映射, 多维尺度 (Multi-Dimensional Scaling, MDS), 投影追踪 (Projection Pursuit) 等。

8). 关联规则学习: 关联规则学习通过寻找最能够解释数据变量之间关系的规则, 来找出大量多元数据集中有用的关联规则。常见算法包括 Apriori 算法和 Eclat 算法等。

9). 集成算法: 集成算法用一些相对较弱的学习模型独立地就同样的样本进行训练, 然后把结果整合起来进行整体预测。集成算法的主要难点在于究竟集成哪些独立的较弱的学习模型以及如何把学习结果整合起来。这是一类非常强大的算法, 同时也非常流行。常见的算法包括: Boosting, Bootstrapped Aggregation (Bagging), AdaBoost, 堆叠泛化 (Stacked Generalization, Blending), 梯度推进机 (Gradient Boosting Machine, GBM), 随机森林 (Random Forest)。

10). 人工神经网络: 人工神经网络算法模拟生物神经网络, 是一类模式匹配算法。通常用于解决分类和回归问题。人工神经网络是机器学习的一个庞大的分支, 有几百种不同的算法。(其中深度学习就是其中的一类算法, 我们会单独讨论), 重要的人工神经网络算法包括: 感知器神经网络 (Perceptron Neural Network), 反向传递 (Back Propagation), Hopfield 网络, 自组织映射 (Self-Organizing Map, SOM)。学习矢量量化 (Learning Vector Quantization, LVQ)。

RF: 通过对训练数据样本以及属性进行有放回的抽样 (针对某一个属性随机选择样本) 这里有两种, 一种是每次都是有放回的采样, 有些样本是重复的, 组成和原始数据集样本个数一样的数据集; 另外一种是不放回的抽样, 抽取 out 大约 60% 的训练信息。由此生成一颗 CART 树, 剩下的样本信息作为袋外数据, 用来当作验证集计算袋外误差测试模型; 把抽取出的样本信息再放回





到原数据集中，再重新抽取一组训练信息，再以此训练数据集生成一颗 CART 树。这样依次生成多颗 CART 树，多颗树组成森林，并且他们的生成都是通过随机采样的训练数据生成，因此叫随机森林。RF 可以用于数据的回归，也可以用于数据的分类。回归时是由多颗树的预测结果求均值；分类是由多棵树的预测结果进行投票。正式由于它的随机性，RF 有极强的防止过拟合的特性。由于他是由 CART 组成，因此它的训练数据不需要进行归一化，因为每棵的建立过程都是通过选择一个能最好的对数据样本进行选择的属性来建立分叉，因此有以上好处的同时也带来了一个缺点，那就是忽略了属性与属性之间的关系。

**K-means:** 基本 K-Means 算法的思想很简单，事先确定常数 K，常数 K 意味着最终的聚类类别数，首先随机选定初始点为质心，并通过计算每一个样本与质心之间的相似度(这里为欧式距离)，将样本点归到最相似的类中，接着，重新计算每个类的质心(即为类中心)，重复这样的过程，知道质心不再改变，最终就确定了每个样本所属的类别以及每个类的质心。由于每次都要计算所有的样本与每一个质心之间的相似度，故在大规模的数据集上，K-Means 算法的收敛速度比较慢。

初始化常数 K，随机选取初始点为质心

重复计算一下过程，直到质心不再改变

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类中

重新计算质心

输出最终的质心以及每个类

## 5、用 mapreduce 实现 10 亿级以上数据的 kmeans

参考回答：

算法 1. map(key, value)

输入：全局变量 centers，偏移量 key，样本 value

输出：<key', value>对，其中 key' 是最近中心的索引，value' 是样本信息的字符串

从 value 构造样本的 instance;

minDis=Double.MAX\_VALUE;

Index=-1;

For i=0 to centers.length do

dis=ComputeDist(instance, centers[i]);

If dis<minDis{

minDis=dis;

index=i;



```
}
```

```
End For
```

把 index 作为 key' ；

把不同维度的 values 构造成 value' ；

输出<key' , value' >对；

```
End
```

注意这里的 Step 2 和 Step 3 初始化了辅助变量 minDis 和 index；Step 4 通过计算找出了与样本最近的中心点，函数 ComputeDist(instance, centers[i]) 返回样本和中心点 centers[i] 的距离；Step 8 输出了用来进行下一个过程（combiner）的中间数据。

Combine 函数. 每个 map 任务完成之后，我们用 combiner 去合并同一个 map 任务的中间结果。因为中间结果是存储在结点的本地磁盘上，所以这个过程不会耗费网络传输的代价。在 combine 函数中，我们把属于相同簇的 values 求和。为了计算每个簇的对象的平均值，我们需要记录每个 map 的每个簇中样本的总数。Combine 函数的伪代码见算法 2。

算法 2. combine(key, V)

输入：key 为簇的索引，V 为属于该簇的样本列表

输出：<key' , value' >对，key' 为簇的索引，value' 是由属于同一类的所有样本总和以及样本数所组成的字符串。

初始化一个数组，用来记录同一类的所有样本的每个维度的总和，样本是 V 中的元素；

初始化一个计数器 num 为 0 来记录属于同一类的样本总数；

```
While (V.hasNext()) {
```

```
    从 V.next() 构造样本实例 instance;
```

```
    把 instance 的不同维度值相加到数组
```

```
    num++;
```

```
}
```

把 key 作为 key' ；

构造 value' ：不同维度的求和结果+num；

输出<key' , value' >对；

```
End
```





Reduce 函数。 Reduce 函数的输入数据由每个结点的 combine 函数获得。如 combine 函数所描述，输入数据包括部分样本（同一类）的求和以及对应样本数。在 reduce 函数中，我们可以把同一类的所有样本求和并且计算出对应的样本数。因此，我们可以得到用于下一轮迭代的新中心。Reduce 函数的伪代码见算法 3。

算法 3. Reduce (key, V)

输入：key 为簇的索引，V 为来自不同结点的部分总和的样本列表

输出：<key', value'>对，key' 为簇的索引，value' 是代表新的聚类中心的字符串

初始化一个数组，用来记录同一类的所有样本的每个维度的总和，样本是 V 中的元素；

初始化一个计数器 NUM 为 0 来记录属于同一类的样本总数；

```
While (V.hasNext()) {
```

```
    从 V.next() 构造样本实例 instance;
```

```
    把 instance 的不同维度值相加到数组
```

```
    NUM+=num;
```

```
}
```

数组的每个元素除以 NUM 来获得新的中心坐标；

把 key 作为 key' ；

构造 value' 为所有中心坐标的字符串；

输出<key', value'>对；

End

## 6、Kmeans

参考回答：

基本 K-Means 算法的思想很简单，事先确定常数 K，常数 K 意味着最终的聚类类别数，首先随机选定初始点为质心，并通过计算每一个样本与质心之间的相似度(这里为欧式距离)，将样本点归到最相似的类中，接着，重新计算每个类的质心(即为类中心)，重复这样的过程，知道质心不再改变，最终就确定了每个样本所属的类别以及每个类的质心。由于每次都要计算所有的样本与每一个质心之间的相似度，故在大规模的数据集上，K-Means 算法的收敛速度比较慢。

初始化常数 K，随机选取初始点为质心



重复计算一下过程，直到质心不再改变

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类中

重新计算质心

输出最终的质心以及每个类

## 7、A/B test 如何进行流量分流

标签：协同过滤算法

参考回答：

略

## 8、协同过滤中的算法怎么细分

考点：推荐算法

标签：协同过滤算法

参考回答：

协同过滤算法通过对用户历史行为数据挖掘发现用户的偏好，基于不同的偏好对用户进行群组划分并推荐相似的商品。协同过的算法分为两类分为基于用户的协同过滤算法和基于物品的协同过滤的算法。基于用户的协同过滤是基于用户对物品的偏好找到相邻邻居用户然后将邻居用户喜欢的推荐给当前的用户。基于物品的协同过滤是计算邻居时采用物品本身，基于用户对物品的偏好找到相似的物品，然后根据用户的历史偏好推荐相似的物品给他。

## 9、FM 公式

标签：FM

参考回答：

$$y(\mathbf{x}) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

## 10、FM 公式

标签：FM

参考回答：



$$y(\mathbf{x}) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

## 5、模型融合和提升的算法

### 1、 bagging 和 boosting 的区别

考点:模型集成的应用

详情: Bagging

参考回答:

Bagging 是从训练集中进行子抽样组成每个基模型所需要的子训练集, 然后对所有基模型预测的结果进行综合操作产生最终的预测结果。

Boosting 中基模型按次序进行训练, 而基模型的训练集按照某种策略每次都进行一定的转化, 最后以一定的方式将基分类器组合成一个强分类器。

Bagging 的训练集是在原始集中有放回的选取, 而 Boosting 每轮的训练集不变, 只是训练集中的每个样本在分类器中的权重都会发生变化, 此权值会根据上一轮的结果进行调整。

Bagging 的各个预测函数可以并行生成, Boosting 的各预测函数只能顺序生成。

Bagging 中整体模型的期望近似于基模型的期望, 所以整体模型的偏差相似于基模型的偏差, 因此 Bagging 中的基模型为强模型 (强模型拥有低偏差高方差)。

Boosting 中的基模型为弱模型, 若不是弱模型会导致整体模型的方差很大。

### 2、boosting 和 bagging 区别

参考回答:

Bagging 和 Boosting 的区别:

1) 样本选择上: Bagging: 训练集是在原始集中有放回选取的, 从原始集中选出的各轮训练集之间是独立的。Boosting: 每一轮的训练集不变, 只是训练集中每个样例在分类器中的权重发生变化。而权值是根据上一轮的分类结果进行调整。

2) 样例权重: Bagging: 使用均匀取样, 每个样例的权重相等。Boosting: 根据错误率不断调整样例的权值, 错误率越大则权重越大。



3) 预测函数: Bagging: 所有预测函数的权重相等。Boosting: 每个弱分类器都有相应的权重, 对于分类误差小的分类器会有更大的权重。

4) 并行计算: Bagging: 各个预测函数可以并行生成。Boosting: 各个预测函数只能顺序生成, 因为后一个模型参数需要前一轮模型的结果。

### 3、XGBOOST 和 GDBT 的区别

考点: 集成模型的基础

详情: GDBT

参考回答:

GDBT 在函数空间中利用梯度下降法进行优化而 XGB 在函数空间中使用了牛顿法进行优化。即 GDBT 在优化中使用了一阶导数信息, 而 XGB 对损失函数进行了二阶泰勒展开, 用到了一阶和二阶导数信息。XGB 在损失函数中加入了正则项(树叶子节点个数, 每个叶子节点上输出 score 的 L2 模平方和)。对于缺失的样本, XGB 可以自动学习出它的分裂方向。GDBT 的节点分裂方式使用的是 gini 系数, XGB 通过优化推导出分裂前后的增益来选择分裂节点。XGB 在处理每个特征列时可以做到并行。

### 4、GDBT 的原理, 以及常用的调参参数

考点: GDBT 知识点

详情: GDBT

参考回答:

先用一个初始值去学习一棵树, 然后在叶子处得到预测值以及预测后的残差, 之后的树则基于之前树的残差不断的拟合得到, 从而训练出一系列的树作为模型。

n\_estimators 基学习器的最大迭代次数, learning\_rate 学习率, max\_leaf\_nodes 最大叶子节点数, max\_depth 树的最大深度, min\_samples\_leaf 叶子节点上最少样本数。

### 5、stacking 和 blending 的区别?

考点: 模型集成

详情: Bagging

参考回答:

Stacking 和 blending 的区别在于数据的划分, blending 用不相交的数据训练不同的基模型, 并将其输出取加权平均。而 stacking 是将数据集划分为两个不相交的集合, 在第一个集合的数据



集中训练多个模型，在第二个数据集中测试这些模型，将预测结果作为输入，将正确的标签作为输出，再训练一个高层的模型。

## 6、AdaBoost 和 GBDT 的区别, AdaBoost 和 GBDT 的区别

考点: Boosting

详情: AdaBoost

参考回答:

AdaBoost 通过调整错分的数据点的权重来改进模型，而 GBDT 是从负梯度的方向去拟合改进模型。

AdaBoost 改变了训练数据的权值，即样本的概率分布，减少上一轮被正确分类的样本权值，提高被错误分类的样本权值，而随机森林在训练每棵树的时候，随机挑选部分训练集进行训练。在对新数据进行预测时，AdaBoost 中所有树加权投票进行预测，每棵树的权重和错误率有关，而随机森林对所有树的结果按照少数服从多数的原则进行预测。

## 7、gbdt 推导

参考回答:

GBDT 全称为 Gradient Boosting Decision Tree。顾名思义，它是一种基于决策树(decision tree)实现的分类回归算法。

Gradient Descent: method of steepest descent

梯度下降作为求解确定可微方程的常用方法而被人所熟知。它是一种迭代求解过程，具体就是使解沿着当前解所对应梯度的反方向迭代。这个方向也叫做最速下降方向。具体推导过程如下。假定当前已经迭代到第  $k$  轮结束，那么第  $k+1$  轮的结果怎么得到呢？我们对函数  $f$  做如下二阶泰勒展开：

$$f(x_{k+1}) = f(x_k + x_{k+1} - x_k) \approx f(x_k) + \nabla f(x_k)(x_{k+1} - x_k)$$

为了使得第  $k+1$  轮的函数值比第  $k$  轮的小，即如下不等式成立。

$$f(x_{k+1}) < f(x_k) \Rightarrow \nabla f(x_k)(x_{k+1} - x_k) < 0$$

则只需使：

$$x_{k+1} = x_k - \gamma \nabla f(x_k),$$

按照这样一直迭代下去，直到  $\nabla f(x_k) = 0$ ， $x_{k+1} = x_k$ ，函数收敛，迭代停止。由于在做泰勒展开时，要求  $x_{k+1} - x_k$  足够小。因此，需要  $\gamma$  比较小才行，一般设置为  $0 \sim 1$  的小数。



顺带提一下, Gradient Descent 是一种一阶优化方法, 为什么这么说呢?因为它在迭代过程中不需要二阶及以上的信息。如果我们在泰勒展开时, 不是一阶展开, 而是二阶展开。那对应的方法就是另一个被大家所熟知的可微方程求解方法: Newton Method, 关于牛顿法的详细内容, 我们会在后续文章介绍。

#### Boosting: Gradient Descent in functional space

Boosting 一般作为一种模型组合方式存在, 这也是它在 GBDT 中的作用。那 Boosting 与 gradient descent 有什么关系呢?上一节我们说到 gradient descent 是一种确定可微方程的求解方法。这里的可微有一个要求, 就是说上文中的损失函数  $f$  针对模型  $x$  直接可微。因此模型  $x$  可以根据梯度迭代直接求解。而这种损失函数针对模型直接可微是一个很强的假设, 不是所有的模型都满足, 比如说决策树模型。现在我们回到第一节, 将  $f(x)$  写的更具体一点:

$$f(x) = l(h(x, D), Y)$$

其中  $D$  为数据特征;  $Y$  为数据 label;  $h$  为模型函数, 解决由  $D \rightarrow Y$  的映射,  $x$  为模型函数参数, 即通常我们说的模型;  $l$  为目标函数或损失函数。

以逻辑回归为例,  $x$  为权重向量,  $h$  模型函数展开为:

$$h(x, d) = \frac{1}{1 + e^{-xd}}$$

目标函数  $l$  展开为:

$$l(h(x, D), Y) = \prod_{d \in D, y \in Y} (h(x, d)^y (1 - h(x, d))^{1-y}), y \in \{0, 1\}$$

我们发现函数  $l$  对  $h$  可微, 同时  $h$  对  $x$  可微, 因此  $l$  对  $x$  可微。因此, 我们可以通过 gradient descent 的方式对  $x$  进行直接求解, 而不用将  $h$  保存下来。然而, 如果  $l$  对  $h$  可微, 但  $h$  对  $x$  不可微呢?我们仍按照第一节的方法先对  $l$  进行泰勒展开, 只不过不是针对  $x$ , 而是对  $h$ 。为了简单起见, 我们省略  $D, Y$ 。

$$l(H(x_{t+1})) = l(H(x_t) + h(x_{t+1})) \approx l(H(x_t)) + \nabla l(H(x_t)) h(x_{t+1})$$

其中:

$$H(x_t) = \sum_{i=1}^t h(x_i)$$

按照第一节的逻辑, 我们不难得出如下迭代公式:

$$H(x_{t+1}) = H(x_t) - \gamma \nabla l(H(x_t)), \quad h(x_{t+1}) = \nabla l(H(x_t))$$

但别忘了, 我们的目的不是求  $h$ , 而是  $x$ 。由于  $h$  对  $x$  不可微, 所以  $x$  必须根据数据重新学习得到。而此时我们重新学习  $x$  的目标已经不是源目标  $Y$ , 而是原损失函数  $l$  在当前  $H$  处的梯度, 即:





$$h: D \xrightarrow{x} \nabla l(H(x, D))$$

这个重新学习  $x$  的过程正是每个 base weak learner 所做的事情。而这种通过 weak learner 拟合每一步迭代后的梯度, 进而实现 weak learner 组合的方式, 就是 Boosting。又由于我们在求导过程中, 损失函数  $l$  没法对模型  $x$  直接求导, 而只能对模型函数  $h$  求导。因此 Boosting 又有一个别名: “函数空间梯度下降”。

此外, 你可能会听过 boosting 的可加性 (additive)。这里顺便提一句, 可加性指的是  $h$  的可加, 而不是  $x$  的可加。比如  $x$  是决策树, 那两棵决策树本身怎么加在一起呢? 你顶多把他们并排放在一起。可加的只是样本根据决策树模型得到的预测值  $h(x, D)$  罢了。

Decision Tree: the based weak learner

Boosting 的本质就是使用每个 weak learner 来拟合截止到当前的梯度。则这里的  $D$ , 还是原来数据中的  $D$ , 而  $Y$  已经不是原来的  $Y$  了。而 GBDT 中的这个 weak learner 就是一棵分类回归树 (CART)。因此我们可以使用决策树直接拟合梯度:  $\nabla l(H(x_t))$ 。此时我们要求的  $x$  就变成了这样一棵有  $k$  个叶子节点的、使得如下目标函数最小化的决策树:

$$C = \sum_{i=1}^k \sum_{j \in L_i} (w_i - T_j)^2$$

其中  $T$  为目标  $\nabla l(H(x_t))$ ,  $w$  为每个叶子节点的权重,  $L$  为叶子节点集合。容易求得每个叶子节点的权重为归属到当前叶子节点的样本均值。即:

$$w_i = \frac{\sum_{j \in L_i} T_j}{\sum_{j \in L_i} 1}$$

每个样本的预测值即为其所归属的叶子节点的权重, 即  $h(x_{t+1})$

## 8、boosting 和 bagging 在不同情况下的选用

参考回答:

Bagging 与 Boosting 的区别:

1) 取样方式 (样本权重): Bagging 是均匀选取, 样本的权重相等, Boosting 根据错误率取样, 错误率越大则权重越大。2) 训练集的选择: Bagging 随机选择训练集, 训练集之间相互独立, Boosting 的各轮训练集的选择与前面各轮的学习结果有关。3) 预测函数: Bagging 各个预测函数没有权重, 可以并行生成, Boosting 有权重, 顺序生成。4) Bagging 是减少 variance, Boosting 是减少 bias。

Bagging 是 Bootstrap Aggregating 的简称, 意思就是再取样 (Bootstrap) 然后在每个样本上训练出来的模型取平均, 所以是降低模型的 variance。Bagging 比如 Random Forest 这种先天并行的算法都有这个效果。



Boosting 则是迭代算法，每一次迭代都根据上一次迭代的预测结果对样本进行加权，所以随着迭代不断进行，误差会越来越小，所以模型的 bias 会不断降低。这种算法无法并行。

## 9、gbdt 推导和适用场景

参考回答：

1) 明确损失函数是误差最小

$$F^* = \operatorname{argmin}_{F(x,y)} [L(y, F(x))]$$

$$F(x; \rho_m, a_m) = \sum_{m=0}^M \rho_m h(x; a_m)$$

2) 构建第一棵回归树

$$F_0(x) = f_0(x)$$

3) 学习多棵回归树

```
for m = 1 ... M :
    g_m(x) = - \frac{\partial E_y[L(y, F(x))|x]}{\partial F(x)} \Big|_{F(x)=F_{m-1}(x)} \quad \text{descent direction}
    \rho_m = \operatorname{argmin}_{\rho} E_y[L(y, F_{m-1}(x) + \rho g_m(x))|x] \quad \text{step size}
    f_m(x) = \rho_m g_m(x)
    F_m(x) = F_{m-1}(x) + \rho_m g_m(x)
end for
```

迭代：计算梯度/残差  $g_m$  (如果是均方误差为损失函数即为残差)

步长/缩放因子  $p$ , 用 a single Newton-Raphson step 去近似求解下降方向步长, 通常的实现中 Step3 被省略, 采用 shrinkage 的策略通过参数设置步长, 避免过拟合: 第  $m$  棵树  $f_m = p * g_m$ ; 模型  $F_m = F_{m-1} + p * g_m$

4)  $F(x)$  等于所有树结果累加

$$F^*(x) \approx F_M(x) = f_0(x) + \sum_{m=1}^M \rho_m g_m(x)$$

适用场景：GBDT 几乎可用于所有回归问题（线性/非线性），GBDT 的适用面非常广。亦可用于二分类问题（设定阈值，大于阈值为正例，反之为负例）。

## 10、说一下 gbdt 的全部算法过程



参考回答：

GBDT (Gradient Boosting Decision Tree) 又叫 MART (Multiple Additive Regression Tree)，是一种用于回归的机器学习算法，该算法由多棵回归决策树组成，所有树的结论累加起来做最终答案。当把目标函数做变换后，该算法亦可用于分类或排序。

1) 明确损失函数是误差最小

$$F^* = \operatorname{argmin} E_{x,y}[L(y, F(x))]$$

$$F(x; \rho_m, a_m) = \sum_{m=0}^M \rho_m h(x; a_m)$$

2) 构建第一棵回归树

$$F_0(x) = f_0(x)$$

3) 学习多棵回归树

```
for m = 1 ... M :
    g_m(x) = - \frac{\partial E_y[L(y, F(x))|x]}{\partial F(x)} \Big|_{F(x)=F_{m-1}(x)} \quad \text{descent direction}
    \rho_m = \operatorname{argmin} E_y[L(y, F_{m-1}(x) + \rho g_m(x)|x)] \quad \text{step size}
    f_m(x) = \rho_m g_m(x)
    F_m(x) = F_{m-1}(x) + \rho_m g_m(x)
end for
```

迭代：计算梯度/残差 g<sub>m</sub> (如果是均方误差为损失函数即为残差)

步长/缩放因子 p, 用 a single Newton-Raphson step 去近似求解下降方向步长, 通常的实现中 Step3 被省略, 采用 shrinkage 的策略通过参数设置步长, 避免过拟合

第 m 棵树 f<sub>m</sub>=p\*g<sub>m</sub>

模型 F<sub>m</sub>=F<sub>m-1</sub>+p\*g<sub>m</sub>

4) F(x) 等于所有树结果累加

$$F^*(x) \approx F_M(x) = f_0(x) + \sum_{m=1}^M \rho_m g_m(x)$$

11、rf 和 gbdt 基分类器区别，里面的决策树分别长啥样，怎么剪枝

参考回答：



GBDT 和 RF 都是集成方法中的经典模型，我们需要弄清楚下面几个问题：1) GBDT 是采用 boosting 方法，RF 采用的是 bagging 方法；2) bias 和 variance 是解释模型泛化性能的，其实还有噪声。

然后，理解 GBDT 和 RF 执行原理，其中 GBDT 中的核心是通过用分类器（如 CART、RF）拟合损失函数梯度，而损失函数的定义就决定了在子区域内各个步长，其中就是期望输出与分类器预测输出的差，即 bias；而 RF 的核心就是自采样（样本随机）和属性随机（所有样本中随机选择 K 个子样本选择最优属性来划分），样本数相同下的不同训练集产生的各个分类器，即数据的扰动导致模型学习性能的变化，即 variance。

#### Gradient boosting Decision Tree (GBDT)

GB 算法中最典型的基学习器是决策树，尤其是 CART，正如名字的含义，GBDT 是 GB 和 DT 的结合。要注意的是这里的决策树是回归树，GBDT 中的决策树是个弱模型，深度较小一般不超过 5，叶子节点的数量也不会超过 10，对于生成的每棵决策树乘上比较小的缩减系数（学习率  $< 0.1$ ），有些 GBDT 的实现加入了随机抽样（ $\text{subsample } 0.5 \leq f \leq 0.8$ ）提高模型的泛化能力。通过交叉验证的方法选择最优的参数。

#### Random Forest:

bagging（你懂得，原本叫 Bootstrap aggregating），bagging 的关键是重复的对经过 bootstrapped 采样来的观测集子集进行拟合。然后求平均。。。一个 bagged tree 充分利用近 2/3 的样本集。。。所以就有了 OOB 预估 (out of bag estimation)

#### GBDT 和随机森林的相同点:

- 1) 都是由多棵树组成；
- 2) 最终的结果都是由多棵树一起决定

#### GBDT 和随机森林的不同点:

- 1) 组成随机森林的树可以是分类树，也可以是回归树；而 GBDT 只由回归树组成；
- 2) 组成随机森林的树可以并行生成；而 GBDT 只能是串行生成；
- 3) 对于最终的输出结果而言，随机森林采用多数投票等；而 GBDT 则是将所有结果累加起来，或者加权累加起来；
- 4) 随机森林对异常值不敏感，GBDT 对异常值非常敏感；
- 5) 随机森林对训练集一视同仁，GBDT 是基于权值的弱分类器的集成；
- 6) 随机森林是通过减少模型方差提高性能，GBDT 是通过减少模型偏差提高性能。

#### RF 决策树:

学习随机森林模型前，一定要先了解决策树模型。树越深，模型越复杂。

决策树模型的优点如下。1) 容易理解和解释，树可以被可视化。2) 不需要太多的数据预处理工作，即不需要进行数据归一化，创造哑变量等操作。3) 隐含地创造了多个联合特征，并能够解决非线性问题。

GBDT 决策树：

迭代决策树 GBDT (Gradient Boosting Decision Tree) 也被称为是 MART (Multiple Additive Regression Tree) 或者是 GBRT (Gradient Boosting Regression Tree)，也是一种基于集成思想的决策树模型，但是它和 Random Forest 有着本质上的区别。不得不提的是，GBDT 是目前竞赛中最为常用的一种机器学习算法，因为它不仅可以适用于多种场景，更难能可贵的是，GBDT 有着出众的准确率。

树的剪枝：

(1) 前剪枝 (Pre-Pruning)

通过提前停止树的构造来对决策树进行剪枝，一旦停止该节点下树的继续构造，该节点就成了叶节点。一般树的前剪枝原则有：a. 节点达到完全纯度；b. 树的深度达到用户所要的深度；c. 节点中样本个数少于用户指定个数；d. 不纯度指标下降的最大幅度小于用户指定的幅度。

(2) 后剪枝 (Post-Pruning)

首先构造完整的决策树，允许决策树过度拟合训练数据，然后对那些置信度不够的结点的子树用叶结点来替代。CART 采用 Cost-Complexity Pruning (代价-复杂度剪枝法)，代价 (cost)：主要指样本错分率；复杂度 (complexity)：主要指树  $t$  的叶节点数，(Breiman...) 定义树  $t$  的代价复杂度 (cost-complexity)：信息熵  $H(X)$ ，信息增益  $= H(D) - H(Y|X)$ ，信息增益率  $= \text{gain}(x) / H(x)$ ，Gini 系数  $= 1 - \sum (p_k^2)$ ，基尼系数就是熵在  $x=1$  的地方一阶泰勒展开得到  $f(x)=1-x$ ，所以  $\text{gini} = \sum [x(1-x)] = 1 - \sum (x^2)$ 。

## 12、随机森林和 GBDT 的区别

参考回答：

1) 随机森林采用的 bagging 思想，而 GBDT 采用的 boosting 思想。这两种方法都是 Bootstrap 思想的应用，Bootstrap 是一种有放回的抽样方法思想。虽然都是有放回的抽样，但二者的区别在于：Bagging 采用有放回的均匀取样，而 Boosting 根据错误率来取样 (Boosting 初始化时对每一个训练样例赋相等的权重  $1/n$ ，然后用该算法对训练集训练  $t$  轮，每次训练后，对训练失败的样例赋以较大的权重)，因此 Boosting 的分类精度要优于 Bagging。Bagging 的训练集的选择是随机的，各训练集之间相互独立，弱分类器可并行，而 Boosting 的训练集的选择与前一轮的学习结果有关，是串行的。2) 组成随机森林的树可以是分类树，也可以是回归树；而 GBDT 只能由回归树组成。3) 组成随机森林的树可以并行生成；而 GBDT 只能是串行生成。4) 对于最终的输出结果而言，随机森林采用多数投票等；而 GBDT 则是将所有结果累加起来，或者加权累加起来。5) 随机森林对异常值不敏感；GBDT 对异常值非常敏感。6) 随机森林对训练集一视同



仁；GBDT 是基于权值的弱分类器的集成。7) 随机森林是通过减少模型方差提高性能；GBDT 是通过减少模型偏差提高性能。

### 13、xgboost 的特征重要性计算

考点:xgboost 基础

参考回答:

Xgboost 根据结构分数的增益情况计算出来选择哪个特征作为分割点, 而某个特征的重要性就是它在所有树中出现的次数之和。

### 14、xgboost 的正则项表达式

考点:xgboost 基础

参考回答:

$$\gamma T + \frac{1}{2} \lambda ||w||^2 \quad T \text{ 为叶子节点的个数, } w \text{ 为叶子节点的分数}$$

### 15、xgboost 原理，怎么防过拟合

参考回答:

XGBoost 是一个树集成模型，它使用的是 K（树的总数为 K）个树的每棵树对样本的预测值的和作为该样本在 XGBoost 系统中的预测，定义函数如下：

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^K f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F}, \quad (1)$$

对于所给的数据集有 n 个样本，m 个特征，定义为：

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\} \quad (|\mathcal{D}| = n, \mathbf{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R}).$$

其中  $x_i$  表示第 i 个样本， $y_i$  表示第 i 个样本的类别标签。CART 树的空间为 F，如下：

$$\mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\} (q: \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T)$$

其中 q 表示每棵树的结构映射每个样本到相应的叶节点的分数，即 q 表示树的模型，输入一个样本，根据模型将样本映射到叶节点输出预测的分数； $w_q(\mathbf{x})$  表示树 q 的所有叶节点的分数组成集合；T 是树 q 的叶节点数量。

所以，由（1）式可以看出，XGBoost 的预测值为每棵树的预测值之和，即每棵树相应的叶节点的得分之和（ $w_i$  的和， $w_i$  表示第 i 个叶节点的得分）。



我们的目标就是学习这样的  $K$  个树模型  $f(x)$ 。为了学习模型  $f(x)$ ，我们定义下面的目标函数：

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \quad (2)$$

where  $\Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2$

$\hat{y}_i$  表示模型的预测值， $y_i$  表示第  $i$  个样本的类别标签， $K$  表示树的数量， $f_k$  表示第  $k$  棵树模型， $T$  表示每棵树的叶子节点数量， $w$  表示每棵树的叶子节点的分数组成的集合， $\gamma$  和  $\lambda$  表示系数，在实际应用中需要调参。

其中，（2）式右边第一项为损失函数项，即训练误差，是一个可微的凸函数（比如用于回归的均方误差和用于分类的 Logistic 误差函数等），第二项为正则化项，即每棵树的复杂度之和，目的是控制模型的复杂度，防止过拟合。我们的目标是在  $L(\phi)$  取得最小化时得出对应的模型  $f(x)$ 。

由于 XGBoost 模型中的优化参数是模型  $f(x)$ ，不是一个具体的值，所以不能用传统的优化方法在欧式空间中进行优化，而是采用 additive training 的方式去学习模型。每一次保留原来的模型不变，加入一个新的函数  $f$  到模型中，如下：

$$\begin{aligned} \hat{y}_i^{(0)} &= 0 \\ \hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\ \hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\ &\dots \\ \hat{y}_i^{(t)} &= \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \end{aligned}$$

第  $t$  轮的模型预测                      保留前面  $t-1$  轮的模型预测                      加入一个新的函数

预测值在每一次迭代中加入一个新的函数  $f$  目的是使目标函数尽量最大地降低。

因为我们的目标是最小化  $L(\phi)$  时得到模型  $f(x)$ ，但是  $L(\phi)$  中并没有参数  $f(x)$ ，所以，我们将上图中的最后一式代入  $L(\phi)$  中可得到如下式子：

$$\mathcal{L}^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \quad (3)$$

对于平方误差（用于回归）来说（3）式转换成如下形式：

$$L^{(t)} = \sum_{i=1}^n (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \Omega(f_t) = \sum_{i=1}^n [2(\hat{y}_i^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2] + \Omega(f_t)$$

残差

对于不是平方误差的情况下，一般会采用泰勒展开式来定义一个近似的目标函数，以方便我们的进一步计算。

根据如下的泰勒展开式，移除高阶无穷小项，得：



$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$$

$$\text{令 } g_i = \frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}} \quad h_i = \frac{\partial^2 l(y_i, \hat{y}^{(t-1)})}{\partial (\hat{y}^{(t-1)})^2}$$

(3) 式等价于下面的式子：

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^n [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

由于我们的目标是求  $L(\phi)$  最小化时的模型  $f(x)$  (也是变量)，当移除常数项时模型的最小值变化，但是取最小值的变量不变 (比如： $y=x^2+C$ ，无论  $C$  去何值， $x$  都在 0 处取最小值)。所以，为了简化计算，我们移除常数项，得到如下的目标函数：

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^n [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t) \quad (4)$$

定义  $I_j = \{i | q(\mathbf{x}_i) = j\}$  为叶节点  $j$  的实例，重写 (4) 式，将关于树模型的迭代转换为关于树的叶子节点的迭代，得到如下过程：

$$\begin{aligned} \mathcal{L}^{(t)} &\approx \sum_{i=1}^n \left[ L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^n \left[ g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^n \left[ g_i w_{q(\mathbf{x}_i)} + \frac{1}{2} h_i w_{q(\mathbf{x}_i)}^2 \right] + \gamma \cdot T + \lambda \cdot \frac{1}{2} \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i \right) w_j^2 \right] + \gamma \cdot T + \lambda \cdot \frac{1}{2} \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma \cdot T \end{aligned}$$

此时我们的目标是求每棵树的叶节点  $j$  的分数  $w_j$ ，求出  $w_j$  后，将每棵树的  $w_j$  相加，即可得到最终的预测的分数。而要想得到最优的  $w_j$  的值，即最小化我们的目标函数，所以上式对  $w_j$  求偏导，并令偏导数为 0，算出此时的  $w_j^*$  为：

$$w_j^* = - \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

$$\text{定义: } G_j = \sum_{i \in I_j} g_i, \quad H_j = \sum_{i \in I_j} h_i \quad \text{得: } w_j^* = - \frac{G_j}{H_j + \lambda}$$

将  $w_j^*$  代入原式得：

$$\tilde{\mathcal{L}}^{(t)} = - \frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma \cdot T \quad (5)$$



方程 (5) 可以用作得分 (score) 函数来测量树结构  $q$  的质量。该得分类似于评估决策树的不纯度得分，除了它是针对更广泛的目标函数得出的。

在 xgboost 调中，一般有两种方式用于控制过拟合：1) 直接控制参数的复杂度：包括 `max_depth` `min_child_weight` `gamma`；2) `add_randomness` 来使得对训练对噪声鲁棒。包括 `subsample` `colsample_bytree`，或者也可以减小步长 `eta`，但是需要增加 `num_round`，来平衡步长因子的减小。

## 16、xgboost, rf, lr 优缺点场景。。。

参考回答：

Xgboost:

优缺点：1) 在寻找最佳分割点时，考虑传统的枚举每个特征的所有可能分割点的贪心法效率太低，xgboost 实现了一种近似的算法。大致的思想是根据百分位法列举几个可能成为分割点的候选者，然后从候选者中根据上面求分割点的公式计算找出最佳的分割点。2) xgboost 考虑了训练数据为稀疏值的情况，可以为缺失值或者指定的值指定分支的默认方向，这能大大提升算法的效率，paper 提到 50 倍。3) 特征列排序后以块的形式存储在内存中，在迭代中可以重复使用；虽然 boosting 算法迭代必须串行，但是在处理每个特征列时可以做到并行。4) 按照特征列方式存储能优化寻找最佳的分割点，但是当以行计算梯度数据时会导致内存的不连续访问，严重时会导致 cache miss，降低算法效率。paper 中提到，可先将数据收集到线程内部的 buffer，然后再计算，提高算法的效率。5) xgboost 还考虑了当数据量比较大，内存不够时怎么有效的使用磁盘，主要是结合多线程、数据压缩、分片的方法，尽可能的提高算法的效率。

适用场景：分类回归问题都可以。

Rf:

优点：

- 1) 表现性能好，与其他算法相比有着很大优势。
- 2) 随机森林能处理很高维度的数据（也就是很多特征的数据），并且不用做特征选择。
- 3) 在训练完之后，随机森林能给出哪些特征比较重要。
- 4) 训练速度快，容易做成并行化方法（训练时，树与树之间是相互独立的）。
- 5) 在训练过程中，能够检测到 feature 之间的影响。
- 6) 对于不平衡数据集来说，随机森林可以平衡误差。当存在分类不平衡的情况时，随机森林能提供平衡数据集误差的有效方法。
- 7) 如果有很大一部分的特征遗失，用 RF 算法仍然可以维持准确度。
- 8) 随机森林算法有很强的抗干扰能力（具体体现在 6, 7 点）。所以当数据存在大量的数据缺失，用 RF 也是不错的。



9) 随机森林抗过拟合能力比较强（虽然理论上说随机森林不会产生过拟合现象，但是在现实中噪声是不能忽略的，增加树虽然能够减小过拟合，但没有办法完全消除过拟合，无论怎么增加树都不行，再说树的数目也不可能无限增加的）。

10) 随机森林能够解决分类与回归两种类型的问题，并在这两方面都有相当好的估计表现。（虽然 RF 能做回归问题，但通常都用 RF 来解决分类问题）。

11) 在创建随机森林时候，对 `generalization error` (泛化误差) 使用的是无偏估计模型，泛化能力强。

缺点：

1) 随机森林在解决回归问题时，并没有像它在分类中表现的那么好，这是因为它并不能给出一个连续的输出。当进行回归时，随机森林不能够做出超越训练集数据范围的预测，这可能导致在某些特定噪声的数据进行建模时出现过拟合。（PS: 随机森林已经被证明在某些噪音较大的分类或者回归问题上回过拟合）。

2) 对于许多统计建模者来说，随机森林给人的感觉就像一个黑盒子，你无法控制模型内部的运行。只能在不同的参数和随机种子之间进行尝试。

3) 可能有很多相似的决策树，掩盖了真实的结果。

4) 对于小数据或者低维数据（特征较少的数据），可能不能产生很好的分类。（处理高维数据，处理特征遗失数据，处理不平衡数据是随机森林的长处）。

5) 执行数据虽然比 `boosting` 等快（随机森林属于 `bagging`），但比单只决策树慢多了。

适用场景：数据维度相对低（几十维），同时对准确性有较高要求时。因为不需要很多参数调整就可以达到不错的效果，基本上不知道用什么方法的时候都可以先试一下随机森林。

Lr:

优点：实现简单，广泛的应用于工业问题上；分类时计算量非常小，速度很快，存储资源低；便利的观测样本概率分数；对逻辑回归而言，多重共线性并不是问题，它可以结合 L2 正则化来解决该问题。

缺点：当特征空间很大时，逻辑回归的性能不是很好；容易欠拟合，一般准确度不太高

不能很好地处理大量多类特征或变量；只能处理两分类问题（在此基础上衍生出来的 `softmax` 可以用于多分类），且必须线性可分；对于非线性特征，需要进行转换。

适用场景：LR 同样是很多分类算法的基础组件，它的好处是输出值自然地落在 0 到 1 之间，并且有概率意义。因为它本质上是一个线性的分类器，所以处理不好特征之间相关的情况。虽然效果一般，却胜在模型清晰，背后的概率学经得住推敲。它拟合出来的参数就代表了每一个特征 (feature) 对结果的影响。也是一个理解数据的好工具。

## 17、xgboost 特征并行化怎么做的

考点: xgboost 并行化

标签: XGBoost

参考回答:

决策树的学习最耗时的一个步骤就是对特征值进行排序, 在进行节点分裂时需要计算每个特征的增益, 最终选增益大的特征做分裂, 各个特征的增益计算就可开启多线程进行。而且可以采用并行化的近似直方图算法进行节点分裂。

## 18、 xgboost 和 lightgbm 的区别和适用场景

标签: XGBoost

参考回答: (1) xgboost 采用的是 level-wise 的分裂策略, 而 lightGBM 采用了 leaf-wise 的策略, 区别是 xgboost 对每一层所有节点做无差别分裂, 可能有些节点的增益非常小, 对结果影响不大, 但是 xgboost 也进行了分裂, 带来了不必要的开销。 leaf-wise 的做法是在当前所有叶子节点中选择分裂收益最大的节点进行分裂, 如此递归进行, 很明显 leaf-wise 这种做法容易过拟合, 因为容易陷入比较高的深度中, 因此需要对最大深度做限制, 从而避免过拟合。

(2) lightgbm 使用了基于 histogram 的决策树算法, 这一点不同与 xgboost 中的 exact 算法, histogram 算法在内存和计算代价上都有不小优势。1) 内存上优势: 很明显, 直方图算法的内存消耗为  $(\#data * \#features * 1\text{Bytes})$  (因为对特征分桶后只需保存特征离散化之后的值), 而 xgboost 的 exact 算法内存消耗为:  $(2 * \#data * \#features * 4\text{Bytes})$ , 因为 xgboost 既要保存原始 feature 的值, 也要保存这个值的顺序索引, 这些值需要 32 位的浮点数来保存。2) 计算上的优势, 预排序算法在选择好分裂特征计算分裂收益时需要遍历所有样本的特征值, 时间为  $(\#data)$ , 而直方图算法只需要遍历桶就行了, 时间为  $(\#bin)$

(3) 直方图做差加速, 一个子节点的直方图可以通过父节点的直方图减去兄弟节点的直方图得到, 从而加速计算。

(4) lightgbm 支持直接输入 categorical 的 feature, 在对离散特征分裂时, 每个取值都当作一个桶, 分裂时的增益算的是 “是否属于某个 category” 的 gain。类似于 one-hot 编码。

(5) xgboost 在每一层都动态构建直方图, 因为 xgboost 的直方图算法不是针对某个特定的 feature, 而是所有 feature 共享一个直方图 (每个样本的权重是二阶导), 所以每一层都要重新构建直方图, 而 lightgbm 中对每个特征都有一个直方图, 所以构建一次直方图就够了。

其适用场景根据实际项目和两种算法的优点进行选择。

## 6、其他重要算法

### 1、问题: HMM 隐马尔可夫模型的参数估计方法是?

知识点: HMM、EM





详情：EM 算法

参考回答：：EM 算法

解析：期望最大化（Expectation-Maximum, EM）算法

## 2、问题：Bootstrap 方法是什么？

知识点：Bootstap、集成学习

详情：Bagging

答案：从一个数据集中有放回的抽取 N 次，每次抽 M 个。

解析：Bagging 算法基于 bootstrap。面试时结合 Bagging 算法讲述会更好。

## 3、问题：如何防止过拟合？

知识点：过拟合

详情：Bagging

答案：1. 早停法；2. L1 和 L2 正则化；3. 神经网络的 dropout；4. 决策树剪枝；5. SVM 的松弛变量；6. 集成学习

解析：能够达到模型权重减小，模型简单的效果

## 4、EM 算法推导，jensen 不等式确定的下界

标签：EM 算法

参考回答：

给定的训练样本是  $\{x^{(1)}, \dots, x^{(m)}\}$ ，样例间独立，我们想找到每个样例隐含的类别  $z$ ，能使得  $p(x, z)$  最大。 $p(x, z)$  的最大似然估计如下：

$$\begin{aligned}\ell(\theta) &= \sum_{i=1}^m \log p(x; \theta) \\ &= \sum_{i=1}^m \log \sum_z p(x, z; \theta).\end{aligned}$$

第一步是对极大似然取对数，第二步是对每个样例的每个可能类别  $z$  求联合分布概率和。但是直接求  $\theta$  一般比较困难，因为有隐藏变量  $z$  存在，但是一般确定了  $z$  后，求解就容易了。

EM 是一种解决存在隐含变量优化问题的有效方法。竟然不能直接最大化  $\ell(\theta)$ ，我们可以不断地建立  $\ell$  的下界（E 步），然后优化下界（M 步）。这句话比较抽象，看下面的。





对于每一个样例  $i$ ，让  $Q_i$  表示该样例隐含变量  $z$  的某种分布， $Q_i$  满足的条件是  $\sum_z Q_i(z) = 1, Q_i(z) \geq 0$ 。（如果  $z$  是连续性的，那么  $Q_i$  是概率密度函数，需要将求和符号换做积分符号）。比如要将班上学生聚类，假设隐藏变量  $z$  是身高，那么就是连续的高斯分布。如果按照隐藏变量是男女，那么就是伯努利分布了。

可以由前面阐述的内容得到下面的公式：

$$\sum_i \log p(x^{(i)}; \theta) = \sum_i \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta) \quad (1)$$

$$= \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \quad (2)$$

$$\geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \quad (3)$$

(1) 到 (2) 比较直接，就是分子分母同乘以一个相等的函数。(2) 到 (3) 利用了 Jensen 不等式，考虑到  $\log(x)$  是凹函数（二阶导数小于 0），而且

$$\sum_{z^{(i)}} Q_i(z^{(i)}) \left[ \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

就是  $[p(x^{(i)}, z^{(i)}; \theta)/Q_i(z^{(i)})]$  的期望（回想期望公式中的 Lazy Statistician 规则）

设  $Y$  是随机变量  $X$  的函数， $Y = g(X)$ （ $g$  是连续函数），那么

(1)  $X$  是离散型随机变量，它的分布律为  $P(X = x_k) = p_k, k=1, 2, \dots$ 。若  $\sum_{k=1}^{\infty} g(x_k)p_k$  绝对收敛，则有

$$E(Y) = E[g(X)] = \sum_{k=1}^{\infty} g(x_k)p_k$$

(2)  $X$  是连续型随机变量，它的概率密度为  $f(x)$ ，若  $\int_{-\infty}^{\infty} g(x)f(x)dx$  绝对收敛，则有

$$E(Y) = E[g(X)] = \int_{-\infty}^{\infty} g(x)f(x)dx$$

对应于上述问题， $Y$  是  $[p(x^{(i)}, z^{(i)}; \theta)/Q_i(z^{(i)})]$ ， $X$  是  $z^{(i)}$ ， $Q_i(z^{(i)})$  是  $p_k$ ， $g$  是  $z^{(i)}$  到  $[p(x^{(i)}, z^{(i)}; \theta)/Q_i(z^{(i)})]$  的映射。这样解释了式子 (2) 中的期望，再根据凹函数时的 Jensen 不等式：

$$f\left(E_{z^{(i)} \sim Q_i} \left[ \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] \right) \geq E_{z^{(i)} \sim Q_i} \left[ f\left( \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right) \right],$$

可以得到 (3)。

这个过程可以看作是对  $\ell(\theta)$  求了下界。对于  $Q_i$  的选择，有多种可能，那种更好的？假设  $\theta$  已经给定，那么  $\ell(\theta)$  的值就决定于  $Q_i(z^{(i)})$  和  $p(x^{(i)}, z^{(i)})$  了。我们可以通过调整这两个概率使下界不断上升，以逼近  $\ell(\theta)$  的真实值，那么什么时候算是调整好了呢？当不等式变成等式时，说明

我们调整后的概率能够等价于 $\ell(\theta)$ 了。按照这个思路，我们要找到等式成立的条件。根据 Jensen 不等式，要想让等式成立，需要让随机变量变成常数值，这里得到：

$$\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} = c$$

$c$  为常数，不依赖于 $z^{(i)}$ 。对此式子做进一步推导，我们知道 $\sum_z Q_i(z^{(i)}) = 1$ ，那么也就有 $\sum_z p(x^{(i)}, z^{(i)}; \theta) = c$ ，（多个等式分子分母相加不变，这个认为每个样例的两个概率比值都是  $c$ ），那么有下式：

$$\begin{aligned} Q_i(z^{(i)}) &= \frac{p(x^{(i)}, z^{(i)}; \theta)}{\sum_z p(x^{(i)}, z; \theta)} \\ &= \frac{p(x^{(i)}, z^{(i)}; \theta)}{p(x^{(i)}; \theta)} \\ &= p(z^{(i)} | x^{(i)}; \theta) \end{aligned}$$

至此，我们推出了在固定其他参数 $\theta$ 后， $Q_i(z^{(i)})$ 的计算公式就是后验概率，解决了 $Q_i(z^{(i)})$ 如何选择的问题。这一步就是 E 步，建立 $\ell(\theta)$ 的下界。接下来的 M 步，就是在给定 $Q_i(z^{(i)})$ 后，调整 $\theta$ ，去极大化 $\ell(\theta)$ 的下界（在固定 $Q_i(z^{(i)})$ 后，下界还可以调整的更大）。那么一般的 EM 算法的步骤如下：

循环重复直到收敛 {

（E 步）对于每一个  $i$ ，计算

$$Q_i(z^{(i)}) := p(z^{(i)} | x^{(i)}; \theta)$$

（M 步）计算

$$\theta := \arg \max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}.$$

那么究竟怎么确保 EM 收敛？假定 $\theta^{(t)}$ 和 $\theta^{(t+1)}$ 是 EM 第  $t$  次和  $t+1$  次迭代后的结果。如果我们证明了 $\ell(\theta^{(t)}) \leq \ell(\theta^{(t+1)})$ ，也就是说极大似然估计单调增加，那么最终我们会到达最大似然估计的最大值。下面来证明，选定 $\theta^{(t)}$ 后，我们得到 E 步

$$Q_i^{(t)}(z^{(i)}) := p(z^{(i)} | x^{(i)}; \theta^{(t)})$$

这一步保证了在给定 $\theta^{(t)}$ 时，Jensen 不等式中的等式成立，也就是

$$\ell(\theta^{(t)}) = \sum_i \sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t)})}{Q_i^{(t)}(z^{(i)})}.$$

然后进行 M 步，固定 $Q_i^{(t)}(z^{(i)})$ ，并将 $\theta^{(t)}$ 视作变量，对上面的 $\ell(\theta^{(t)})$ 求导后，得到 $\theta^{(t+1)}$ ，这样经过一些推导会有以下式子成立：

$$\ell(\theta^{(t+1)}) \geq \sum_i \sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t+1)})}{Q_i^{(t)}(z^{(i)})} \quad (4)$$

$$\geq \sum_i \sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t)})}{Q_i^{(t)}(z^{(i)})} \quad (5)$$

$$= \ell(\theta^{(t)}) \quad (6)$$

解释第（4）步，得到 $\theta^{(t+1)}$ 时，只是最大化 $\ell(\theta^{(t)})$ ，也就是 $\ell(\theta^{(t+1)})$ 的下界，而没有使等式成立，等式成立只是在固定 $\theta$ ，并按 E 步得到 $Q_i$ 时才能成立。

况且根据我们前面得到的下式，对于所有的 $Q_i$ 和 $\theta$ 都成立

$$\ell(\theta) \geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

第（5）步利用了 M 步的定义，M 步就是将 $\theta^{(t)}$ 调整到 $\theta^{(t+1)}$ ，使得下界最大化。因此（5）成立，（6）是之前的等式结果。这样就证明了 $\ell(\theta)$ 会单调增加。一种收敛方法是 $\ell(\theta)$ 不再变化，还有一种就是变化幅度很小。

再次解释一下（4）、（5）、（6）。首先（4）对所有的参数都满足，而其等式成立条件只是在固定 $\theta$ ，并调整好 Q 时成立，而第（4）步只是固定 Q，调整 $\theta$ ，不能保证等式一定成立。（4）到（5）就是 M 步的定义，（5）到（6）是前面 E 步所保证等式成立条件。也就是说 E 步会将下界拉到与 $\ell(\theta)$ 一个特定值（这里 $\theta^{(t)}$ ）一样的高度，而此时发现下界仍然可以上升，因此经过 M 步后，下界又被拉升，但达不到与 $\ell(\theta)$ 另外一个特定值一样的高度，之后 E 步又将下界拉到与这个特定值一样的高度，重复下去，直到最大值。

如果我们定义

$$J(Q, \theta) = \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})},$$

从前面的推导中我们知道 $\ell(\theta) \geq J(Q, \theta)$ ，EM 可以看作是 J 的坐标上升法，E 步固定 $\theta$ ，优化 Q，M 步固定 Q 优化 $\theta$ 。

Jensen 不等式表述如下：

如果 f 是凸函数，X 是随机变量，那么： $E(f(x)) \geq f(E(x))$ ，特别地，如果 f 是严格凸函数，当且仅当 X 是常量时，上式取等号。所以其下界是 $f(E(x))$ 。

## 三、机器学习

### 1、Scikit-learn

#### 1、Focal Loss 介绍一下



标签：Scikit-learn

参考回答：

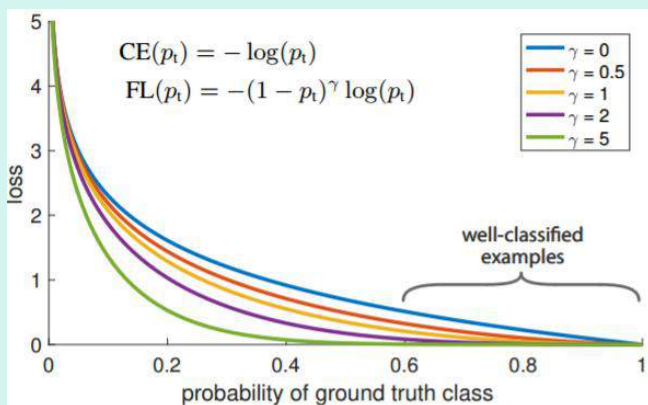
Focal loss 主要是为了解决 one-stage 目标检测中正负样本比例严重失衡的问题。该损失函数降低了大量简单负样本在训练中所占的权重，也可理解为一种困难样本挖掘。

损失函数形式：Focal loss 是在交叉熵损失函数基础上进行的修改，首先回顾二分类交叉熵损失：

$$L = -y \log y' - (1 - y) \log(1 - y') = \begin{cases} -\log y' & , \quad y = 1 \\ -\log(1 - y') & , \quad y = 0 \end{cases}$$

$y'$  是经过激活函数的输出，所以在 0-1 之间。可见普通的交叉熵对于正样本而言，输出概率越大损失越小。对于负样本而言，输出概率越小则损失越小。此时的损失函数在大量简单样本的迭代过程中比较缓慢且可能无法优化至最优。

$$L_{fl} = \begin{cases} -(1 - y')^\gamma \log y' & , \quad y = 1 \\ -y'^\gamma \log(1 - y') & , \quad y = 0 \end{cases}$$



首先在原有的基础上加了一个因子，其中  $\gamma > 0$  使得减少易分类样本的损失。使得更关注于困难的、错分的样本。

例如  $\gamma$  为 2，对于正类样本而言，预测结果为 0.95 肯定是简单样本，所以  $(1 - 0.95)$  的  $\gamma$  次方就会很小，这时损失函数值就变得很小。而预测概率为 0.3 的样本其损失相对很大。对于负类样本而言同样，预测 0.1 的结果应当远比预测 0.7 的样本损失值要小得多。对于预测概率为 0.5 时，损失只减少了 0.25 倍，所以更加关注于这种难以区分的样本。这样减少了简单样本的影响，大量预测概率很小的样本叠加起来后的效应才可能比较有效。

此外，加入平衡因子  $\alpha$ ，用来平衡正负样本本身的比例不均：

$$L_{fl} = \begin{cases} -\alpha(1 - y')^\gamma \log y' & , \quad y = 1 \\ -(1 - \alpha)y'^\gamma \log(1 - y') & , \quad y = 0 \end{cases}$$

只添加  $\alpha$  虽然可以平衡正负样本的重要性，但是无法解决简单与困难样本的问题。



$\lambda$  调节简单样本权重降低的速率，当  $\lambda$  为 0 时即为交叉熵损失函数，当  $\lambda$  增加时，调整因子的影响也在增加。实验发现  $\lambda$  为 2 是最优。

## 2、过拟合的解决方法

考点：机器学习基础

详情：Scikit-learn

参考回答：

正则化 (L1 正则化, L2 正则化), 扩增数据集, 特征的筛选, early stopping, dropout

## 3、方差偏差的分解公式

考点：机器学习应用

详情：scikit-learn

参考回答：

$$E(f;D) = E_D \left[ (f(x;D) - f'(x))^2 \right] + (f'(x) - y)^2 + E_D[(y_D - y)^2]$$

$$E_D \left[ (f(x;D) - f'(x))^2 \right] \text{ 为模型的方差}$$

$$(f'(x) - y)^2 \text{ 为模型的偏差}$$

$$E_D[(y_D - y)^2] \text{ 为模型的噪声}$$

$f(x;D)$  为训练集  $D$  上学得模型  $f$  在  $x$  上的输出

$f'(x)$  为模型的期望预测

## 4、问题：对应时间序列的数据集如何进行交叉验证？

知识点：交叉验证、时间序列

详情：Scikit-learn

参考回答：1. 预测后一半；2. 日向前链

解析：传统的交叉验证由于假定样本独立同分布，因此随机打乱分为训练集和验证集。但是对于时间序列来讲，需要考虑序列间的时间依赖。



#### 5、问题：正负样本不平衡的解决办法？评价指标的参考价值？

知识点：正负样例比不平衡

详情：Scikit-learn

参考回答：上下采样法。

好的指标：ROC 和 AUC、F 值、G-Mean；不好的指标：Precision、Recall

#### 6、迁移学习

标签：Scikit-learn

参考回答：

迁移学习就是把之前训练好的模型直接拿来用，可以充分利用之前数据信息，而且能够避免自己实验数据量较小等问题。简单来讲就是给模型做初始化，初始化的数据来自于训练好的模型。

#### 7、数据不平衡怎么办？

考点：数据处理

标签：Scikit-learn

参考回答：

使用正确的评估标准，当数据不平衡时可以采用精度，调用度，F1 得分，MCC，AUC 等评估指标。

重新采样数据集，如欠采样和过采样。欠采样通过减少冗余类的大小来平衡数据集。当数据量不足时采用过采样，尝试通过增加稀有样本的数量来平衡数据集，通过使用重复，自举，SMOTE 等方法生成新的样本。

以正确的方式使用 K-fold 交叉验证，组合不同的重采样数据集，对多数类进行聚类。

#### 8、AUC 的理解

考点：模型评估

标签：Scikit-learn

参考回答：

Auc 体现出容忍样本倾斜的能力，只反应模型对正负样本排序能力的强弱，而其直观含以上是任意取一个正样本和负样本，正样本的得分大于负样本的概率。



## 9、AUC 的计算公式

考点:模型评估

标签: Scikit-learn

参考回答:

$$AUC = \frac{\sum_{i \in \text{正类}} rank_i - \frac{M(1+M)}{2}}{M \times N}$$

M 为正样本数, N 为负样本数。Rank 的值代表能够产生前大后小这样的组合数, 但是其中包含了(正, 正)的情况, 所以要减去正例的个数所以可得上述公式。

## 10、生成模型和判别模型的区别

考点:模型区别

标签: Scikit-learn

参考回答:

生成模型是先从数据中学习联合概率分布, 然后利用贝叶斯公式求得特征和标签对应的条件概率分布。判别模型直接学习条件概率分布, 直观的输入什么特征就预测可能的类别。

## 11、过拟合的解决方法

考点:机器学习基础

标签: Scikit-learn

参考回答:

正则化(L1 正则化, L2 正则化), 扩增数据集, 特征的筛选, early stopping, dropout

## 12、特征选择怎么做

标签: Scikit-learn

参考回答:

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。L1 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。

### 13、怎么防止过拟合

标签：Scikit-learn

参考回答：

1)。在训练和建立模型的时候，从相对简单的模型开始，不要一开始就把特征做的非常多，模型参数跳的非常复杂。

2)。增加样本，要覆盖全部的数据类型。数据经过清洗之后再进行模型训练，防止噪声数据干扰模型。

3)。正则化。在模型算法中添加惩罚函数来防止过拟合。常见的有 L1, L2 正则化。

4)。集成学习方法 bagging (如随机森林) 能有效防止过拟合

5)。减少特征个数 (不太推荐) 注意：降维不能解决过拟合。降维只是减小了特征的维度，并没有减小特征所有的信息。

### 14、L1 和 L2 正则

标签：Scikit-learn

参考回答：

L 范数 (L1 norm) 是指向量中各个元素绝对值之和，也有个美称叫“稀疏规则算子” (Lasso regularization)。比如 向量  $A=[1, -1, 3]$ ，那么 A 的 L1 范数为  $|1|+|-1|+|3|$ 。简单总结一下就是：L1 范数：为 x 向量各个元素绝对值之和。L2 范数：为 x 向量各个元素平方和的  $1/2$  次方，L2 范数又称 Euclidean 范数或者 Frobenius 范数  $L_p$  范数：为 x 向量各个元素绝对值  $p$  次方和的  $1/p$  次方。在支持向量机学习过程中，L1 范数实际是一种对于成本函数求解最优的过程，因此，L1 范数正则化通过向成本函数中添加 L1 范数，使得学习得到的结果满足稀疏化，从而方



便人类提取特征，即 L1 范数可以使权值稀疏，方便特征提取。L2 范数可以防止过拟合，提升模型的泛化能力。L1 和 L2 的差别，为什么一个让绝对值最小，一个让平方最小，会有那么大的差别呢？看导数一个是 1 一个是  $w$  便知，在靠近零附近，L1 以匀速下降到零，而 L2 则完全停下来了。这说明 L1 是将不重要的特征（或者说，重要性不在一个数量级上）尽快剔除，L2 则是把特征贡献尽量压缩最小但不至于为零。两者一起作用，就是把重要性在一个数量级（重要性最高的）的那些特征一起平等共事（简言之，不养闲人也不要超人）。

## 15、ID3 树用什么指标选择特征

标签：Scikit-learn

参考回答：

基于信息增益最大的作为最优特征，以此为决策树的根节点

## 16、特征工程的问题

标签：Scikit-learn

参考回答：

特征工程包括数据与特征处理、特征选择和降维三部分。数据与特征处理包括：

1. 数据选择、清洗、采样

数据格式化；

数据清洗，填充缺失值、去掉脏数据，将不可信的样本丢掉，缺省值极多的字段考虑不用；

采样：针对正负样本不平衡的情况，当正样本远大于负样本时，且量都很大时，使用下采样，量不大时，可采集更多的数据或 oversampling 或修改损失函数；采样过程中可利用分层抽样保持不同类别数据的比例。

2. 不同类型数据的特征处理

数值型：幅度调整/归一化、log 等变化、统计值（例如 max、min、mean、std）、离散化、分桶等

类别型：one-hot 编码等

时间型：提取出连续值的持续时间和间隔时间；提取出离散值的“年”、“月”、“日”、“一年中哪个星期/季度”、“一周中的星期几”、“工作日/周末”等信息

文本型：使用 If-idf 特征



统计型：加减平均、分位线、次序、比例

意义：

对数据进行预处理，可提高数据质量，提高挖掘质量。对数据进行清洗可填充缺失值、光滑噪声数据，识别和删除离群点数据，保证数据的一致性；

使用正确的采样方法可解决因数据不平衡带来的预测偏差；

对不同的数据类型进行不同的特征处理有助于提高特征的可用性，例如对数值型数据进行归一化可将数据转化到统一量纲下；对类别型数据，可用 one-hot 编码方法将类别数据数字化，数字化特征之后可更用来计算距离、相似性等；可从时间型数据当中提取中更多的时间特征，例如年、月和日等，这些特征对于业务场景以及模型的预测往往有很大的帮助。统计型特征处理有助于从业务场景中挖掘更丰富的信息。

特征选择包括：

1. Filter：使用方差、Pearson 相关系数、互信息等方法过滤特征，评估单个特征和结果值之间的相关程度，留下 Top 相关的特征部分。

2. Wrapper：可利用“递归特征删除算法”，把特征选择看做一个特征子集搜索问题，筛选各种特征子集，用模型评估效果。

3. Embedded：可利用正则化方式选择特征，使用带惩罚项的基模型，除了选择出特征外，同时也进行了降维。

意义：

-剔除对结果预测不大的特征，减小冗余，选择有意义的特征输入模型，提高计算性能。

降维：

方法：主成分分析法（PCA）和线性判别分析（LDA）

意义：通过 PCA 或 LDA 方法，将较高纬度样本空间映射到较低维度的样本空间，从而达到降维的目的，减少模型的训练时间，提高模型的计算性能。

17、给了个链接线上写代码，要求写读文本、文本预处理、特征提取和建模的基本过程，不过写到特征就没写了

标签：Scikit-learn

参考回答：

收集数据

众所周知，数据挖掘模型中非常重要的部分是训练模型，训练集与测试集便是整个数据挖掘过程中花费时间最多的过程。数据集通过有如下的一些途径获得：

经典数据集：Python NLTK 便提供了非常多经典的数据集。很多数据集都是手工标注而成，所以使用的时候不得不感叹工程的浩大。例如 NLP 中使用的 Penn TreeBank，有兴趣的同学可以看看他们的论文《Building a Large Annotated Corpus of English: The Penn TreeBank》，那简直就是一部辛酸史啊！

从网页上抓取：直接动手写一个爬虫爬取特定的网页不难，通过正则表达式就能够将有效的内容提取出来；当然，发扬拿来主义精神的话，我们可以使用 Python 中一些优秀的库，比如 scrapy, beautifulsoup 等等。

从日志、已有文件中分析：如果是海量数据的话可以使用 hadoop 这样的系统。结合传统 SQL 中的一些特殊功能，例如 Partition，有时会有不错的效果，不过最多压缩空间、缩减特征再用 SQL 处理。

其他网络数据集：Stanford Large Network Dataset Collectionm, 100+ Interesting Data Sets for Statistics

#### 预处理

如果是网页内容，首先需要去掉 Html Tag，lxml 和 html5lib 是比较有名的 Python 库，beautifulsoup 也对他们做了一层封装。不过别忘了，Python 本身也自带了 sgmlib 这样的基本可扩展的解析器。如果有特别的处理，其实正则表达式也是不错的选择。

处理编码，由于我主要是处理英文的数据，这一步基本也跳过了。

将文档分割成句子（可选）。很多时候我们采用的是词袋模型（bag of words），所以是否分割成句子也无所谓。比较简单的方法就是 Python NLTK 中的 sent\_tokenize() 函数，用的是 punkt 算法，论文在这里。

将句子分割成词。首先用正则表达式可以自己完成：如果要利用已有工具，Python NLTK 中的 word\_tokenize()，这个方式就是前文提到的 Penn TreeBank 语料库所使用的分词方法。听起来是不是很高大上，我是不会告诉你其实它也是正则表达式实现的，想知道具体实现，戳这里。分词其实主要干了这么几个事：1) 将' 分开. don't -> do n't, they'll -> they 'll; 2) 将大部分标点当作单独的一个词；3) 将最后一位是逗号或者引号的词分开；4) 单独出现在一行的句号分开。中文分词区别比较大，可以采用斯坦福或者 ICTCLAS（中科院背景）的方案。

拼写错误纠正。推荐 pyenchant，非常喜欢，因为简洁到四句语句就能完成。Windows 8 中操作系统也直接提供了拼写检查的 COM 端口，不过就得多花时间研究啦。

POS Tagging（根据实际应用）。还是 Nltk，首页就有介绍；斯坦福也提供了这类工具。这一块属于 NLP 的范畴，还是 Parsing 等应用，要了解 NLP 原理推荐 Coursera 上一门不错的课程 Natural Language Processing

去掉标点。正则表达式即可，有的时间非常短的单词也可以一起去掉，len<3 的常见的选择

去掉非英文字符的词（根据实际应用决定）。



转换成小写。

去掉停用词。就是在各种句子中都经常出现的一些词，I、and 什么的。NLTK 有一个 Stopwords。Matthew L. Jockers 提供了一份比机器学习和自然语言处理中常用的停用表更长的停用表。中文停用词戳这里。什么？你问我停用词怎么找到的，我想大概是 IDF 这样的算法吧。

词型转换。简单来讲，我们希望 do、did、done 都能统一的返回 do。第一种方法叫 stem，Porter 是比较常见的一种基于规则的算法，网页有 snowball 工具，也是它的论文。Porter 的结果差强人意，单词末尾有 e、y 的，基本上 stem 之后都不见了，例如 replace→replac；末尾有重复单词的，基本只剩一个了，例如 ill→il。NLTK 中也有 Stem 库，算法应该是类似的。第二种方法叫 lemmatization，就是基于词典做词型转换，NLTK 的 Stem 库中便有 WordNetLemmatizer 可以使用。

去掉长度过小的词（可选）。如果之前做了，这里要再做一次，因为 stem 会改变词型。

重新去停用词。理由同上。

特征提取：

- 1、TF-IDF：
- 2、词频方法 (Word Frequency)：
- 3、文档频次方法 (Document Frequency)：
- 4、互信息 (Mutual Information)：
- 5、期望交叉熵 (Expected Cross Entropy)：
- 6、二次信息熵 (QEMI)：
- 7、信息增益方法 (Information Gain)：
- 8、 $\chi^2$  统计量方法：
- 9、文本证据权 (The Weight of Evidence for Text)：
- 10、优势率 (Odds Ratio)：
- 11、遗传算法 (Genetic Algorithm, GA)：
- 12、主成分分析法 (Principal Component Analysis, PCA)：
- 13、模拟退火算法 (Simulating Anneal, SA)：
- 14、N-Gram 算法

文本建模：





LDA、pLSA、LSA

## 18、softmax 公式

标签: Scikit-learn

参考回答: 考虑一个多分类问题, 即预测变量  $y$  可以取  $k$  个离散值中的任何一个. 比如一个邮件分类系统将邮件分为私人邮件, 工作邮件和垃圾邮件. 由于  $y$  仍然是一个离散值, 只是相对于二分类的逻辑回归多了一些类别. 下面将根据多项式分布建模.

考虑将样本共有  $k$  类, 每一类的概率分别为  $\phi_1, \dots, \phi_k$ , 由于  $\sum_{i=1}^k \phi_i = 1$ , 所以通常我们只需要  $k-1$  个参数  $\phi_1, \dots, \phi_{k-1}$  即可

$$\phi_i = p(y = i; \phi), \quad p(y = k; \phi) = 1 - \sum_{i=1}^{k-1} \phi_i$$

为了推导, 引入表达式:

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(3) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, T(k-1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, T(k) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

上面  $T(y)$  是  $k-1$  维列向量, 其中  $y = 1, 2, \dots, k$ .

$T(y)_i$  表示向量  $T(y)$  的第  $i$  个元素.

还要引入表达式  $1\{\cdot\}$ , 如果大括号里面为真, 则真个表达式就为 1, 否则为 0. 例如:  $1\{2=3\} = 0$  和  $1\{3=3\} = 1$ .

则上面的  $k$  个向量就可以表示为  $(T(y))_i = 1\{y = i\}$ .

以为  $y$  只能属于某一个类别, 于是  $T(y)$  中只能有一个元素为 1 其他元素都为 0, 可以求出  $k-1$  个元素的期望:  $E[(T(y))_i] = P(y = i) = \phi_i$ .

定义:  $\eta_i = \log \frac{\phi_i}{\phi_k}$ .

其中  $i = 1, 2, \dots, k$ . 则有:

$$\begin{aligned} e^{\eta_i} &= \frac{\phi_i}{\phi_k} \\ \phi_k e^{\eta_i} &= \phi_i \\ \phi_k \sum_{i=1}^k e^{\eta_i} &= \sum_{i=1}^k \phi_i = 1 \end{aligned}$$

也就容易得出:  $\phi_k = 1 / \sum_{i=1}^k e^{\eta_i}$ , 由该式和上面使得等式:  $\phi_k e^{\eta_i} = \phi_i$  一起可以

得到:  $\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}}$  这个函数就是 softmax 函数。



然后假设  $\eta_i$ 's 和  $x$ 's 具有线性关系，即  $\eta_i = \theta_i^T x$  (for  $i = 1, \dots, k-1$ )

于是从概率的角度出发：

$$\begin{aligned} p(y=i|x;\theta) &= \phi_i \\ &= \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}} \\ &= \frac{e^{\theta_i^T x}}{\sum_{j=1}^k e^{\theta_j^T x}} \end{aligned}$$

其中  $y \in \{1, \dots, k\}$  这个模型就是 softmax 回归 (softmax regression)，它是逻辑回归的泛化。

这样我们的输出：

$$\begin{aligned} h_{\theta}(x) &= E[T(y)|x; \theta] \\ &= E \left[ \begin{bmatrix} 1\{y=1\} \\ 1\{y=2\} \\ \vdots \\ 1\{y=k-1\} \end{bmatrix} \middle| x; \theta \right] \\ &= \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{k-1} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\exp(\theta_1^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \frac{\exp(\theta_2^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \vdots \\ \frac{\exp(\theta_{k-1}^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \end{bmatrix}. \end{aligned}$$

就是输出了  $x$  属于  $(1, 2, \dots, k-1)$  中每一类的概率，当然属于第  $k$  类的概率就是：

$$1 - \sum_{i=1}^{k-1} \phi_i.$$

下面开始拟合参数

同样使用最大化参数  $\theta$  的对数似然函数：

$$\begin{aligned} \ell(\theta) &= \sum_{i=1}^m \log p(y^{(i)}|x^{(i)}; \theta) \\ &= \sum_{i=1}^m \log \prod_{l=1}^k \left( \frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}} \right)^{1\{y^{(i)}=l\}} \end{aligned}$$

这里使用梯度下降和牛顿法均可。

## 19、softmax 公式

标签: Scikit-learn

参考回答: 考虑一个多分类问题, 即预测变量  $y$  可以取  $k$  个离散值中的任何一个. 比如一个邮件分类系统将邮件分为私人邮件, 工作邮件和垃圾邮件. 由于  $y$  仍然是一个离散值, 只是相对于二分类的逻辑回归多了一些类别. 下面将根据多项式分布建模.

考虑将样本共有  $k$  类, 每一类的概率分别为  $\phi_1, \dots, \phi_k$ , 由于  $\sum_{i=1}^k \phi_i = 1$ , 所以通常我们只需要  $k-1$  个参数  $\phi_1, \dots, \phi_{k-1}$  即可

$$\phi_i = p(y = i; \phi), \quad p(y = k; \phi) = 1 - \sum_{i=1}^{k-1} \phi_i$$

为了推导, 引入表达式:

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(3) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, T(k-1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, T(k) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

上面  $T(y)$  是  $k-1$  维列向量, 其中  $y = 1, 2, \dots, k$ .

$T(y)_i$  表示向量  $T(y)$  的第  $i$  个元素.

还要引入表达式  $1\{\cdot\}$ , 如果大括号里面为真, 则真个表达式就为 1, 否则为 0. 例如:  $1\{2=3\} = 0$  和  $1\{3=3\} = 1$ .

则上面的  $k$  个向量就可以表示为  $(T(y))_i = 1\{y = i\}$ .

以为  $y$  只能属于某一个类别, 于是  $T(y)$  中只能有一个元素为 1 其他元素都为 0, 可以求出  $k-1$  个元素的期望:  $E[(T(y))_i] = P(y = i) = \phi_i$ .

定义:  $\eta_i = \log \frac{\phi_i}{\phi_k}$ .

其中  $i = 1, 2, \dots, k$ . 则有:

$$\begin{aligned} e^{\eta_i} &= \frac{\phi_i}{\phi_k} \\ \phi_k e^{\eta_i} &= \phi_i \\ \phi_k \sum_{i=1}^k e^{\eta_i} &= \sum_{i=1}^k \phi_i = 1 \end{aligned}$$

也就容易得出:  $\phi_k = 1 / \sum_{i=1}^k e^{\eta_i}$ , 由该式和上面使得等式:  $\phi_k e^{\eta_i} = \phi_i$  一起可以得到:  $\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}}$  这个函数就是 softmax 函数.

然后假设  $\eta_i$ 's 和  $x$ 's 具有线性关系, 即  $\eta_i = \theta_i^T x$  (for  $i = 1, \dots, k-1$ )

于是从概率的角度出发:

$$\begin{aligned} p(y = i|x; \theta) &= \phi_i \\ &= \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}} \\ &= \frac{e^{\theta_i^T x}}{\sum_{j=1}^k e^{\theta_j^T x}} \end{aligned}$$

其中  $y \in \{1, \dots, k\}$  这个模型就是 softmax 回归 (softmax regression)，它是逻辑回归的泛化。

这样我们的输出：

$$\begin{aligned} h_{\theta}(x) &= E[T(y)|x; \theta] \\ &= E \left[ \begin{bmatrix} 1\{y=1\} \\ 1\{y=2\} \\ \vdots \\ 1\{y=k-1\} \end{bmatrix} \middle| x; \theta \right] \\ &= \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{k-1} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\exp(\theta_1^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \frac{\exp(\theta_2^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \vdots \\ \frac{\exp(\theta_{k-1}^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \end{bmatrix} \end{aligned}$$

就是输出了  $x$  属于  $(1, 2, \dots, k-1)$  中每一类的概率，当然属于第  $k$  类的概率就是：  
 $1 - \sum_{i=1}^{k-1} \phi_i$

下面开始拟合参数

同样使用最大化参数  $\theta$  的对数似然函数：

$$\begin{aligned} \ell(\theta) &= \sum_{i=1}^m \log p(y^{(i)}|x^{(i)}; \theta) \\ &= \sum_{i=1}^m \log \prod_{l=1}^k \left( \frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}} \right)^{1\{y^{(i)}=l\}} \end{aligned}$$

这里使用梯度下降和牛顿法均可。

## 2、Libsvm

### 1、检测 20 类物体，多少张训练集，怎么训练



标签: Libsvm

参考回答:

多分类问题，保证各类别的样例比，提取特征，用 libsvm 等多分类。

## 2、lightgbm 优势

标签: lightgbm

参考回答:

1) 更快的训练速度和更高的效率: LightGBM 使用基于直方图的算法。2) 更低的内存占用: 使用离散的箱子(bins)保存并替换连续值导致更少的内存占用。3) 更高的准确率(相比于其他任何提升算法): 它通过 leaf-wise 分裂方法产生比 level-wise 分裂方法更复杂的树, 这就是实现更高准确率的主要因素。然而, 它有时候或导致过拟合, 但是我们可以通过设置|max-depth|参数来防止过拟合的发生。4) 大数据处理能力: 相比于 XGBoost, 由于它在训练时间上的缩减, 它同样能够具有处理大数据的能力。5) 支持并行学习。

## 3、Keras/tensorflow

### 1、MXNet 和 Tensorflow 的区别

考点: 深度学习框架

详情: keras/tensorflow

参考回答:

MXNet 有两个主要的进程 server 和 worker, worker 之间不能进行通信, 只能通过 server 互相影响。Tensorflow 有 worker, server, client 三种进程, worker 是可以相互通信的, 可以根据 op 的依赖关系主动收发数据。MXNet 常用来做数据并行, 每个 GPU 设备上包含了计算图中所有的 op, 而 Tensorflow 可以由用户指定 op 的放置, 一般情况下一个 GPU 设备负责某个和几个 op 的训练任务。

### 2、Tensorflow 的工作原理

考点: 深度学习框架

详情: keras/tensorflow

参考回答:



Tensorflow 是用数据流图来进行数值计算的, 而数据流图是描述有向图的数值计算过程。在有向图中, 节点表示为数学运算, 边表示传输多维数据, 节点也可以被分配到计算设备上从而并行的执行操作。

### 3.Tensorflow 中 interactivesession 和 session 的区别

考点:tensorflow 基础

参考回答:

Tf. Interactivesession() 默认自己就是用户要操作的会话, 而 tf.Session() 没有这个默认, 所以 eval() 启动计算时需要指明使用的是哪个会话。

4、手写了 tensorflow 的图像分类代码, 还有问之前线下笔试最后编程题的思路, 算法复杂度, 然后项目也问。

参考回答:

tensorflow 的图像分类代码

```
# -*- coding: utf-8 -*-
```

```
from skimage import io, transform
```

```
import glob
```

```
import os
```

```
import tensorflow as tf
```

```
import numpy as np
```

```
import time
```

```
path='e:/flower/'
```

```
#将所有的图片 resize 成 100*100
```

```
w=100
```

```
h=100
```

```
c=3
```

```
#读取图片
```





```
def read_img(path):

    cate=[path+x for x in os.listdir(path) if os.path.isdir(path+x)]

    imgs=[]

    labels=[]

    for idx, folder in enumerate(cate):

        for im in glob.glob(folder+'/*.jpg'):

            print('reading the images:%s'%(im))

            img=io.imread(im)

            img=transform.resize(img, (w, h))

            imgs.append(img)

            labels.append(idx)

    return np.asarray(imgs, np.float32), np.asarray(labels, np.int32)

data, label=read_img(path)

#打乱顺序

num_example=data.shape[0]

arr=np.arange(num_example)

np.random.shuffle(arr)

data=data[arr]

label=label[arr]


#将所有数据分为训练集和验证集

ratio=0.8

s=np.int(num_example*ratio)

x_train=data[:s]

y_train=label[:s]
```



```
x_val=data[s:]

y_val=label[s:]

#-----构建网络-----

x=tf.placeholder(tf.float32, shape=[None, w, h, c], name='x')

y=tf.placeholder(tf.int32, shape=[None, ], name='y_')

#第一个卷积层 (100-->50)

conv1=tf.layers.conv2d(inputs=x, filters=32, kernel_size=[5, 5], padding="same",

activation=tf.nn.relu, kernel_initializer=tf.truncated_normal_initializer(stddev=0.01))

pool1=tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)

#第二个卷积层 (50->25)

conv2=tf.layers.conv2d(inputs=pool1, filters=64, kernel_size=[5, 5], padding="same",

activation=tf.nn.relu, kernel_initializer=tf.truncated_normal_initializer(stddev=0.01))

pool2=tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)

#第三个卷积层 (25->12)

conv3=tf.layers.conv2d(inputs=pool2, filters=128, kernel_size=[3, 3], padding="same",

activation=tf.nn.relu, kernel_initializer=tf.truncated_normal_initializer(stddev=0.01))

pool3=tf.layers.max_pooling2d(inputs=conv3, pool_size=[2, 2], strides=2)

#第四个卷积层 (12->6)

conv4=tf.layers.conv2d(inputs=pool3, filters=128, kernel_size=[3, 3], padding="same",
```



```
activation=tf.nn.relu,kernel_initializer=tf.truncated_normal_initializer(stddev=0.01))

pool4=tf.layers.max_pooling2d(inputs=conv4, pool_size=[2, 2], strides=2)

rel = tf.reshape(pool4, [-1, 6 * 6 * 128])

#全连接层

dense1 = tf.layers.dense(inputs=rel, units=1024, activation=tf.nn.relu,

kernel_initializer=tf.truncated_normal_initializer(stddev=0.01),

kernel_regularizer=tf.contrib.layers.l2_regularizer(0.003))

dense2= tf.layers.dense(inputs=dense1, units=512, activation=tf.nn.relu,

kernel_initializer=tf.truncated_normal_initializer(stddev=0.01),

kernel_regularizer=tf.contrib.layers.l2_regularizer(0.003))

logits= tf.layers.dense(inputs=dense2, units=5, activation=None,

kernel_initializer=tf.truncated_normal_initializer(stddev=0.01),

kernel_regularizer=tf.contrib.layers.l2_regularizer(0.003))

#-----网络结束-----

loss=tf.losses.sparse_softmax_cross_entropy(labels=y_, logits=logits)

train_op=tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)

correct_prediction = tf.equal(tf.cast(tf.argmax(logits, 1), tf.int32), y_)

acc= tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

#定义一个函数，按批次取数据

def minibatches(inputs=None, targets=None, batch_size=None, shuffle=False):
```



```
assert len(inputs) == len(targets)

if shuffle:

    indices = np.arange(len(inputs))

    np.random.shuffle(indices)

for start_idx in range(0, len(inputs) - batch_size + 1, batch_size):

    if shuffle:

        excerpt = indices[start_idx:start_idx + batch_size]

    else:

        excerpt = slice(start_idx, start_idx + batch_size)

    yield inputs[excerpt], targets[excerpt]

#训练和测试数据，可将 n_epoch 设置更大一些

n_epoch=10

batch_size=64

sess=tf.InteractiveSession()

sess.run(tf.global_variables_initializer())

for epoch in range(n_epoch):

    start_time = time.time()

    #training

    train_loss, train_acc, n_batch = 0, 0, 0

    for x_train_a, y_train_a in minibatches(x_train, y_train, batch_size,
shuffle=True):

        _, err, ac=sess.run([train_op, loss, acc], feed_dict={x: x_train_a, y:
y_train_a})

        train_loss += err; train_acc += ac; n_batch += 1

    print("    train loss: %f" % (train_loss/ n_batch))
```



```
print("    train acc: %f" % (train_acc/ n_batch))

#validation

val_loss, val_acc, n_batch = 0, 0, 0

for x_val_a, y_val_a in minibatches(x_val, y_val, batch_size, shuffle=False):

    err, ac = sess.run([loss, acc], feed_dict={x: x_val_a, y_: y_val_a})

    val_loss += err; val_acc += ac; n_batch += 1

print("    validation loss: %f" % (val_loss/ n_batch))

print("    validation acc: %f" % (val_acc/ n_batch))

sess.close()
```

## 四、深度学习

### 1、Batch Normalization 的作用

考点:神经网络基础

详情:深度学习

参考回答:

神经网络在训练的时候随着网络层数的加深,激活函数的输入值的整体分布逐渐往激活函数的取值区间上下限靠近,从而导致在反向传播时低层的神经网络的梯度消失。而Batch Normalization的作用是通过规范化的手段,将越来越偏的分布拉回到标准化的分布,使得激活函数的输入值落在激活函数对输入比较敏感的区域,从而使梯度变大,加快学习收敛速度,避免梯度消失的问题。

### 2、梯度消失

考点:深度学习

参考回答:在神经网络中,当前面隐藏层的学习速率低于后面隐藏层的学习速率,即随着隐藏层数目的增加,分类准确率反而下降了。这种现象叫做消失的梯度问题。

### 3、循环神经网络,为什么好?



考点：深度学习

参考回答：循环神经网络模型（RNN）是一种节点定向连接成环的人工神经网络，是一种反馈神经网络，RNN 利用内部的记忆来处理任意时序的输入序列，并且在其处理单元之间既有内部的反馈连接又有前馈连接，这使得 RNN 可以更加容易处理不分段的文本等。

#### 4、什么是 Group Convolution

考点：卷积的应用

详情：深度学习

参考回答：

若卷积神经网络的上一层有  $N$  个卷积核，则对应的通道数也为  $N$ 。设群数目为  $M$ ，在进行卷积操作的时候，将通道分成  $M$  份，每个 group 对应  $N/M$  个通道，然后每个 group 卷积完成后输出叠在一起，作为当前层的输出通道。

#### 5、什么是 RNN

考点：循环神经网络

详情：深度学习

参考回答：

一个序列当前的输出与前面的输出也有关，在 RNN 网络结构中，隐藏层的输入不仅包括输入层的输出还包含上一时刻隐藏层的输出，网络会对之前的信息进行记忆并应用于当前的输入计算中。

#### 6、训练过程中，若一个模型不收敛，那么是否说明这个模型无效？导致模型不收敛的原因有哪些？

考点：神经网络基础

详情：深度学习

参考回答：

并不能说明这个模型无效，导致模型不收敛的原因可能有数据分类的标注不准确，样本的信息量太大导致模型不足以 fit 整个样本空间。学习率设置的太大容易产生震荡，太小会导致不收敛。可能复杂的分类任务用了简单的模型。数据没有进行归一化的操作。





## 7、图像处理中锐化和平滑的操作

考点:图像处理基础

详情:深度学习

参考回答:

锐化就是通过增强高频分量来减少图像中的模糊,在增强图像边缘的同时也增加了图像的噪声。

平滑与锐化相反,过滤掉高频分量,减少图像的噪声是图片变得模糊。

## 8、VGG 使用 3\*3 卷积核的优势是什么?

考点:VGG 基础

详情:深度学习

参考回答:

2 个 3\*3 的卷积核串联和 5\*5 的卷积核有相同的感知野,前者拥有更少的参数。多个 3\*3 的卷积核比一个较大尺寸的卷积核有更多层的非线性函数,增加了非线性表达,使判决函数更具有判决性。

## 9、Relu 比 Sigmoid 的效果好在哪里?

考点:激活函数对比

详情:深度学习

参考回答:

Sigmoid 的导数只有在 0 的附近时有较好的激活性,而在正负饱和区域的梯度趋向于 0,从而产生梯度弥散的现象,而 relu 在大于 0 的部分梯度为常数,所以不会有梯度弥散现象。Relu 的导数计算的更快。Relu 在负半区的导数为 0,所以神经元激活值为负时,梯度为 0,此神经元不参与训练,具有稀疏性。

## 10、问题:神经网络中权重共享的是?

知识点:权重共享

详情:深度学习

参考回答:卷积神经网络、循环神经网络

解析:通过网络结构直接解释

11、问题：神经网络激活函数？

知识点：激活函数

详情：深度学习

参考回答：sigmoid、tanh、relu

解析：需要掌握函数图像，特点，互相比，优缺点以及改进方法

12、问题：在深度学习中，通常会 finetuning 已有的成熟模型，再基于新数据，修改最后几层神经网络权值，为什么？

知识点：finetuning，迁移学习

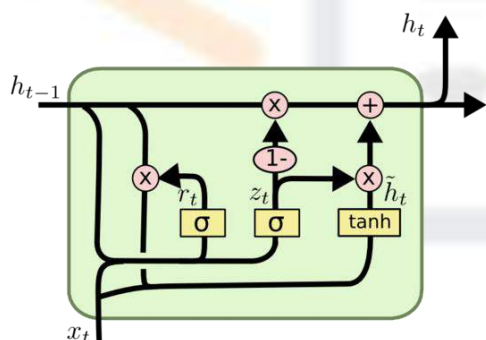
详情：深度学习

参考回答：实践中的数据集质量参差不齐，可以使用训练好的网络来进行提取特征。把训练好的网络当做特征提取器。

13、问题：画 GRU 结构图

知识点：GRU

详情：深度学习



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

参考回答：GRU 有两个门：更新们，输出们

解析：如果不会画 GRU，可以画 LSTM 或者 RNN。再或者可以讲解 GRU 与其他两个网络的联系和区别。不要直接就说不会。

14、Attention 机制的作用

考点：注意力机制基础

详情：深度学习

参考回答：

减少处理高维输入数据的计算负担, 结构化的选取输入的子集, 从而降低数据的维度。让系统更加容易的找到输入的数据中与当前输出信息相关的有用信息, 从而提高输出的质量。帮助类似于 decoder 这样的模型框架更好的学到多种内容模态之间的相互关系。

### 15、Lstm 和 Gru 的原理

考点: 循环神经网络

详情: 深度学习

参考回答:

Lstm 由输入门, 遗忘门, 输出门和一个 cell 组成。第一步是决定从 cell 状态中丢弃什么信息, 然后在决定有多少新的信息进入到 cell 状态中, 最终基于目前的 cell 状态决定输出什么样的信息。

Gru 由重置门和更新门组成, 其输入为前一时刻隐藏层的输出和当前的输入, 输出为下一时刻隐藏层的信息。重置门用来计算候选隐藏层的输出, 其作用是控制保留多少前一时刻的隐藏层。更新门的作用是控制加入多少候选隐藏层的输出信息, 从而得到当前隐藏层的输出。

### 16、什么是 dropout

考点: 防过拟合方法

详情: 深度学习

参考回答:

在神经网络的训练过程中, 对于神经单元按一定的概率将其随机从网络中丢弃, 从而达到对于每个 mini-batch 都是在训练不同网络的效果, 防止过拟合。

### 17、LSTM 每个门的计算公式

考点: 循环网络基础

详情: 深度学习

参考回答:

遗忘门:  $f = \sigma(W_f[h_{t-1}, x_t] + b_f)$

输入门:  $i = \sigma(W_i[h_{t-1}, x_t] + b_i)$

输出门:  $o = \sigma(W_o[h_{t-1}, x_t] + b_o)$



### 18、HOG 算法原理

考点:目标检测算法

详情:深度学习

### 19、DropConnect 的原理

考点:防过拟合方法

详情:深度学习

参考回答:

防止过拟合方法的一种,与 dropout 不同的是,它不是按概率将隐藏层的节点输出清 0,而是对每个节点与之相连的输入权值以一定的概率清 0。

### 20、深度学习了解多少，有看过底层代码吗？caffe,tf?

建议找一个项目中常用的方法，看一下底层代码逻辑

详情:深度学习

参考回答:

略

(腾讯)深度学习过拟合，过拟合的解决方案

1. 早停法；2. L1 和 L2 正则化；3. 神经网络的 dropout；4. 决策树剪枝；5. SVM 的松弛变量；6. 集成学习

### 21、深度学习过拟合，过拟合的解决方案

参考回答:

1. 早停法；2. L1 和 L2 正则化；3. 神经网络的 dropout；4. 决策树剪枝；5. SVM 的松弛变量；6. 集成学习

### 22、除了 GMM-HMM，你了解深度学习在语音识别中的应用吗？

参考回答:

讲了我用的过 DNN-HMM，以及与 GMM-HMM 的联系与区别；然后 RNN+CTC，这里我只是了解，大概讲了一下 CTC 损失的原理；然后提了一下 CNN+LSTM。

### 23、用过哪些移动端深度学习框架？

参考回答：

开源的有：小米的 MACE，骁龙的 SNPE，腾讯的 FeatherCNN 和 ncnn，百度的 mobile-deep-learning (MDL)；caffe、tensorflow lite 都有移动端，只是可能没有上面的框架效率高。据传还有支付宝的 xNN，商汤的 PPL，不过都是自用，未开源。

### 24、Caffe：整体架构说一下，新加一个层需要哪些步骤，卷积是怎么实现的，多卡机制，数据并行还是模型并行？

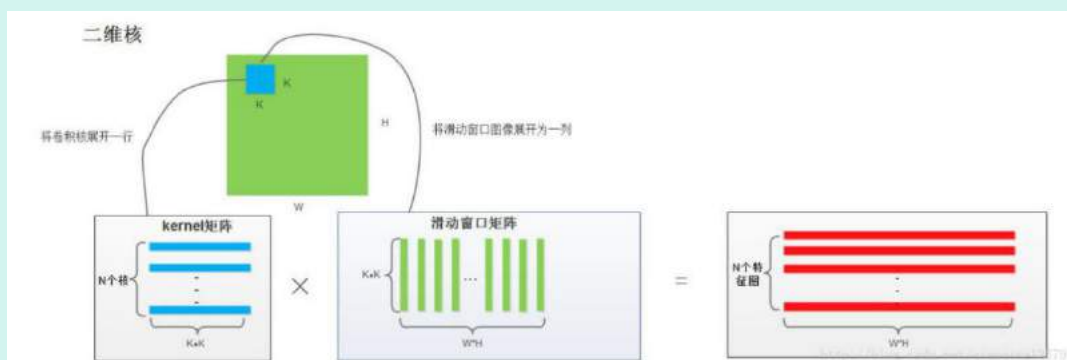
参考回答：

Caffe 是深度学习的一个框架，Caffe 框架主要包括五个组件：Blob、Solver、Net、Layer、Proto；框架结构如下图所示。这五大组件可以分为两个部分：第一部分，Blob、Layer 和 Net，这三个组件使得 Caffe 构成基于自己的模块化的模型，caffe 是逐层地定义一个 net，而 net 是从数据输入层到损失层自下而上定义整个模型，Blob 在 caffe 中是处理和传递实际数据的数据封装包；第二部分：Solver 和 Proto，这两个模型分别用于协调模型的优化以及用于网络模型的结构定义、存储和读取的方式 (Layer-By-Layer) 定义 Net，而贯穿所有 Nets 的结构就是 caffe 框架或模型；对于 Layer 而言，输入的是 Blob 数据封装包格式的实际数据，当采用该框架进行训练时，也就是 Solver 调优模型，则需要 Proto 这种网络模型的结构定义、存储和读取。

总体来说，caffe 是通过 Layer

Caffe 中卷积运算的原理

俗话说，一图胜千言，首先先给出原理示意图，为了方便理解，这里以二维核为例

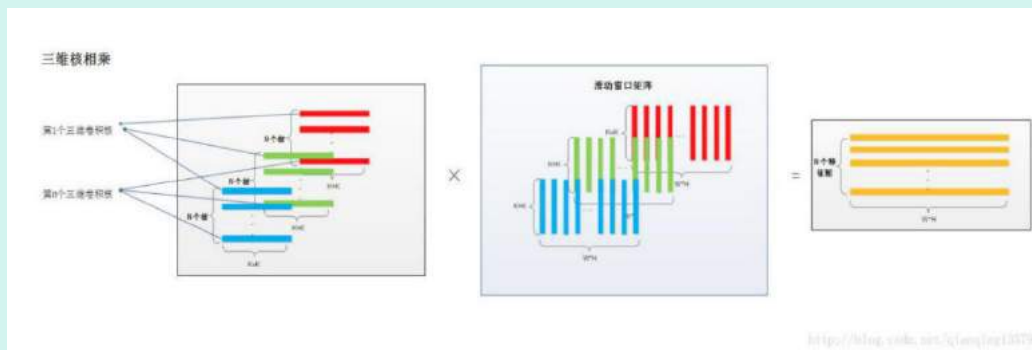
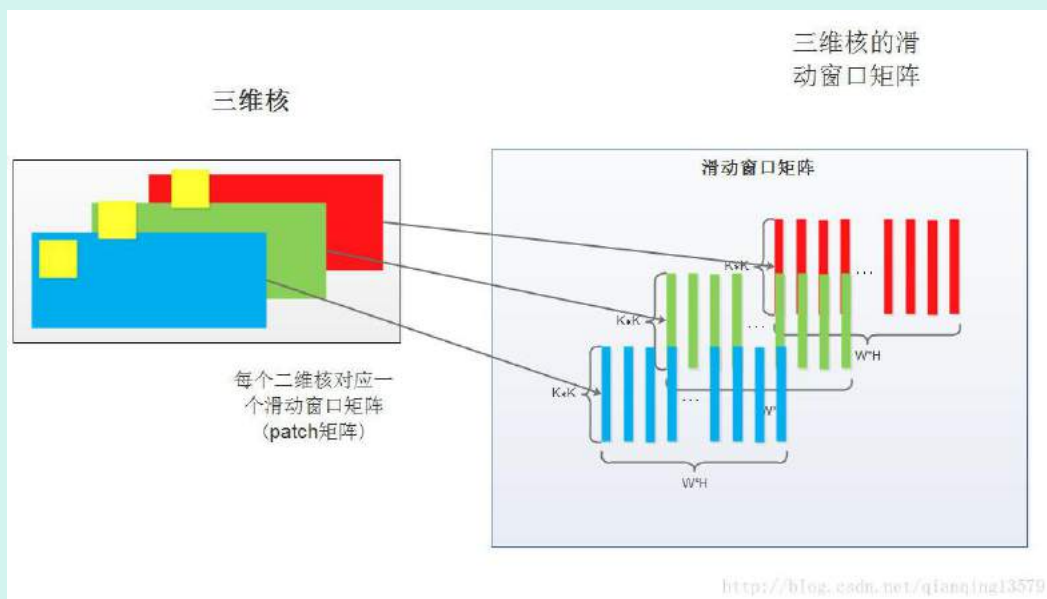


滑动窗口在图像中每滑动一个地方，将图像中该滑动窗口图像展开为一列，所有列组成图中的滑动窗口矩阵，这里假设 pad=1, stride=1, K=3, 则滑动窗口矩阵每行大小为 W\*H, 一共 K\*K 行。

每个核展开为一行，N 个核形成的核矩阵大小为 N\*K\*K。

最后将核矩阵和滑动窗口矩阵相乘，每一行就是一个特征图， $N$  个卷积核形成  $N$  个特征图。

扩展到三维核



三维核就是多了一个通道的概念，原理与二维核一样。

caffe 支持多 GPU 并行了，原理比较简单，就是每个 GPU 分别算一个 batch， $n$  个 GPU，实际的 batchsize 就是  $n \times \text{batch}$ ，比如原来用一个 GPU，batchsize 设置成 256，现在用 4 个 GPU，把 batchsize 设置成 64，和原来的一个 GPU 的运算是等价的。

实际使用的时候基本不用设置，和原来一样编译好就可以用了。命令就是在 `-gpu` 后面对多个 GPU 号用逗号隔开，比如 `-gpu 1, 2, 3, 4` 就是同时使用 1-4 共 4 个 GPU，GPU 编号可以不连续，或者直接用 `-gpu all`，就是使用所有的 GPU。

Caffe 是数据并行的。

## 25、HOG 算子是怎么求梯度的

参考回答：





略

## 26、BN 层的作用，为什么要在后面加伽马和贝塔，不加可以吗

标签：深度学习

参考回答：

BN 层的作用是把一个 batch 内的所有数据，从不规范的分布拉到正态分布。这样做的好处是使得数据能够分布在激活函数的敏感区域，敏感区域即为梯度较大的区域，因此在反向传播的时候能够较快反馈误差传播。

## 27、梯度消失，梯度爆炸的问题，

标签：深度学习

参考回答：

激活函数的原因，由于梯度求导的过程中梯度非常小，无法有效反向传播误差，造成梯度消失的问题。

## 28、Adam

标签：深度学习

参考回答：

Adam 算法和传统的随机梯度下降不同。随机梯度下降保持单一的学习率（即 alpha）更新所有的权重，学习率在训练过程中并不会改变。而 Adam 通过计算梯度的一阶矩估计和二阶矩估计而为不同的参数设计独立的自适应性学习率。

## 29、attention 机制

标签：深度学习

参考回答：

Attention 简单理解就是权重分配，。以 seq2seq 中的 attention 公式作为讲解。就是对输入的每个词分配一个权重，权重的计算方式为与解码端的隐含层时刻作比较，得到的权重的意义就是权重越大，该词越重要。最终加权求和。

$$u_t^i = v^T \tanh(W_1 h^i + W_2 H^t)$$



$$\alpha_t^i = \text{softmax}(u_t^i)$$

$$c^t = \sum_i \alpha_t^i h^i$$

### 30、RNN 梯度消失问题,为什么 LSTM 和 GRU 可以解决此问题

考点:循环网络模型

标签:深度学习

参考回答:

RNN 由于网络较深,后面层的输出误差很难影响到前面层的计算,RNN 的某一单元主要受它附近单元的影响。而 LSTM 因为可以通过阀门记忆一些长期的信息,相应的也就保留了更多的梯度。而 GRU 也可通过重置和更新两个阀门保留长期的记忆,也相对解决了梯度消失的问题。

### 31、GAN 网络的思想

考点:GAN

标签:深度学习

参考回答:

GAN 用一个生成模型和一个判别模型,判别模型用于判断给定的图片是不是真实的图片,生成模型自己生成一张图片和想要的图片很像,开始时两个模型都没有训练,然后两个模型一起进行对抗训练,生成模型产生图片去欺骗判别模型,判别模型去判别真假,最终两个模型在训练过程中,能力越来越强最终达到稳态。

### 32、1\*1 的卷积作用

考点:CNN 基础

标签:深度学习

参考回答:

实现跨通道的交互和信息整合,实现卷积核通道数的降维和升维,可以实现多个 feature map 的线性组合,而且可是实现与全连接层的等价效果。

### 33、怎么提升网络的泛化能力



考点:网络性能提升

标签:深度学习

参考回答:

从数据上提升性能:收集更多的数据,对数据做缩放和变换,特征组合和重新定义问题。

从算法调优上提升性能:用可靠的模型诊断工具对模型进行诊断,权重的初始化,用小的随机数初始化权重。对学习率进行调节,尝试选择合适的激活函数,调整网络的拓扑结构,调节 batch 和 epoch 的大小,添加正则化的方法,尝试使用其它的优化方法,使用 early stopping。

### 34、什么是 seq2seq model

考点:编码解码模型

标签:深度学习

参考回答:

Seq2seq 属于 encoder-decoder 结构的一种,利用两个 RNN,一个作为 encoder 一个作为 decoder。Encoder 负责将输入序列压缩成指定长度的向量,这个向量可以看作这段序列的语义,而 decoder 负责根据语义向量生成指定的序列。

### 35、激活函数的作用

考点:激活函数

参考回答:

激活函数是用来加入非线性因素的,提高神经网络对模型的表达能力,解决线性模型所不能解决的问题。

### 36、为什么用 relu 就不用 sigmoid 了

考点:激活函数

标签:深度学习

参考回答:

Sigmoid 的导数只有在 0 的附近时有比较好的激活性,在正负饱和区域的梯度都接近 0,会导致梯度弥散。而 relu 函数在大于 0 的部分梯度为常数,不会产生梯度弥散现象。Relu 函数在负半区导数为 0,也就是说这个神经元不会经历训练,就是所谓稀疏性。而且 relu 函数的导数计算的更快。



## 37、讲一下基于 WFST 的静态解码网络的语音识别流程？

参考回答：

从语音特征开始讲起，我讲了 MFCC 和 LPC 的原理以及提取过程，这一部分讲的很细，然后讲了 viterbi 解码过程，最后概述了一下 HCLG.fst 构建流程

## 38、目标检测了解吗，Faster RCNN 跟 RCNN 有什么区别

标签：深度学习

参考回答：

目标检测，也叫目标提取，是一种基于目标几何和统计特征的图像分割，它将目标的分割和识别合二为一，其准确性和实时性是整个系统的一项重要能力。尤其是在复杂场景中，需要对多个目标进行实时处理时，目标自动提取和识别就显得特别重要。

随着计算机技术的发展和计算机视觉原理的广泛应用，利用计算机图像处理技术对目标进行实时跟踪研究越来越热门，对目标进行动态实时跟踪定位在智能化交通系统、智能监控系统、军事目标检测及医学导航手术中手术器械定位等方面具有广泛的应用价值。

	使用方法	缺点	改进
R-CNN	1、SS 提取 RP； 2、CNN 提取特征； 3、SVM 分类； 4、BB 盒回归。	1、训练步骤繁琐（微调网络+训练 SVM+训练 bbox）； 2、训练、测试均速度慢； 3、训练占空间	1、从 DPM HSC 的 34.3%直接提升到了 66%（mAP）； 2、引入 RP+CNN
Faster RCNN	1、RPN 提取 RP； 2、CNN 提取特征； 3、softmax 分类； 4、多任务损	1、还是无法达到实时检测目标； 2、获取 region proposal，再对每个 proposal 分类计算量还是比较大。	1、提高了检测精度和速度； 2、真正实现端到端的目标检测框架； 3、生成建议框仅需约 10ms。



	失函数边框回归。		

### 39、SPP, YOLO 了解吗?

标签：深度学习

参考回答：

SPP-Net 简介：

SPP-Net 主要改进有下面两个：

1) . 共享卷积计算、2) . 空间金字塔池化

在 SPP-Net 中同样由这几个部分组成：

ss 算法、CNN 网络、SVM 分类器、bounding box

ss 算法的区域建议框同样在原图上生成，但是却在 Conv5 上提取，当然由于尺寸的变化，在 Conv5 层上提取时要经过尺度变换，这是它 R-CNN 最大的不同，也是 SPP-Net 能够大幅缩短时长的原因。因为它充分利用了卷积计算，也就是每张图片只卷积一次，但是这种改进带来了一个新的问题，由于 ss 算法生成的推荐框尺度是不一致的，所以在 cov5 上提取到的特征尺度也是不一致的，这样是没有办法做全尺寸卷积的（Alexnet）。

所以 SPP-Net 需要一种算法，这种算法能够把不一致的输入产生统一的输出，这就 SPP，即空间金字塔池化，由它替换 R-CNN 中的 pooling 层，除此之外，它和 R-CNN 就一样了。

YOLO 详解：

YOLO 的名字 You only look once 正是自身特点的高度概括。YOLO 的核心思想在于将目标检测作为回归问题解决，YOLO 首先将图片划分成  $S \times S$  个区域，注意这个区域的概念不同于上文提及将图片划分成  $N$  个区域扔进 detector 这里的区域不同。上文提及的区域是真的将图片进行剪裁，或者说把图片的某个局部的像素扔进 detector，而这里的划分区域，只是逻辑上的划分。

### 40、梯度消失梯度爆炸怎么解决

标签：深度学习

参考回答：

1)、使用 ReLU、LReLU、ELU、maxout 等激活函数

sigmoid 函数的梯度随着  $x$  的增大或减小而消失，而 ReLU 不会。

2)、使用批规范化



通过规范化操作将输出信号  $x$  规范化到均值为 0，方差为 1 保证网络的稳定性。从上述分析分可以看到，反向传播式子中有  $w$  的存在，所以  $w$  的大小影响了梯度的消失和爆炸，Batch Normalization 就是通过对每一层的输出规范为均值和方差一致的方法，消除了  $w$  带来的放大缩小的影响，进而解决梯度消失和爆炸的问题。

#### 41、RNN 容易梯度消失，怎么解决？

标签：深度学习

参考回答：

##### 1)、梯度裁剪 (Clipping Gradient)

既然在 BP 过程中会产生梯度消失（就是偏导无限接近 0，导致长时记忆无法更新），那么最简单粗暴的方法，设定阈值，当梯度小于阈值时，更新的梯度为阈值。

优点：简单粗暴

缺点：很难找到满意的阈值

##### 2)、LSTM (Long Short-Term Memory)

一定程度上模仿了长时记忆，相比于梯度裁剪，最大的优点就是，自动学习在什么时候可以将 error 反向传播，自动控制哪些是需要作为记忆存储在 LSTM cell 中。一般长时记忆模型包括写入，读取，和忘记三个过程对应到 LSTM 中就变成了 input\_gate, output\_gate,

forget\_gate, 三个门，范围在 0 到 1 之间，相当于对输入输出进行加权的学习，利用大量数据来自动学习加权的参数（即学习了哪些错误可以用 BP 更新参数）。具体的公式表达：

$$\begin{aligned} 1. \quad i_t &= \text{Sigm}(\theta_{ix}x_t + \theta_{ih}h_{t-1} + b_i) \\ 2. \quad o_t &= \text{Sigm}(\theta_{ox}x_t + \theta_{oh}h_{t-1} + b_o) \\ 3. \quad f_t &= \text{Sigm}(\theta_{fx}x_t + \theta_{fh}h_{t-1} + b_f) \\ 4. \quad g_t &= \text{Tanh}(\theta_{gx}x_t + \theta_{gh}h_{t-1} + b_g) \\ 5. \quad c_t &= i_t \odot g_t + f_t \odot c_{t-1} \\ 6. \quad h_t &= o_t \odot \text{Tanh}(c_t) \end{aligned}$$

[http://blog.csdn.net/qq\\_29340857](http://blog.csdn.net/qq_29340857)

优点：模型自动学习更新参数

#### 42、LSTM 跟 RNN 有啥区别

标签：深度学习

参考回答：

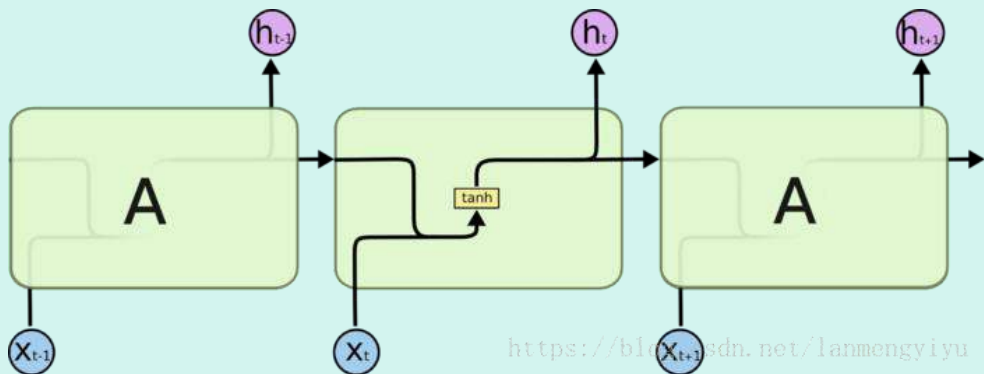


## LSTM 与 RNN 的比较

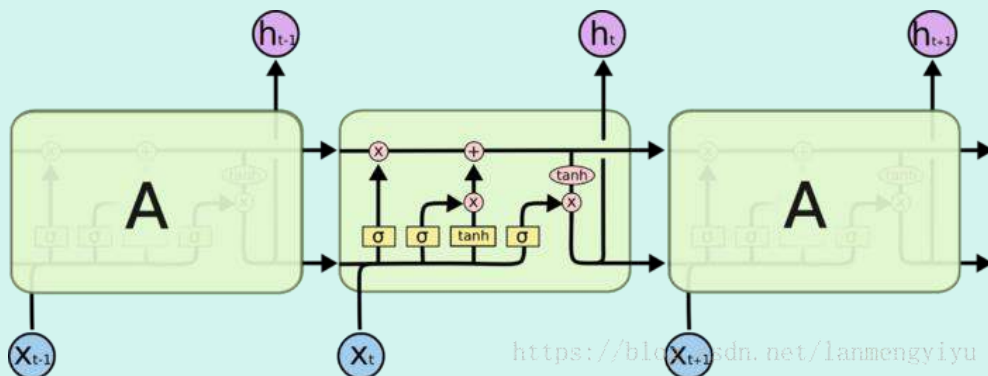
RNN 在处理 long term memory 的时候存在缺陷，因此 LSTM 应运而生。LSTM 是一种变种的 RNN，它的精髓在于引入了细胞状态这样一个概念，不同于 RNN 只考虑最近的状态，LSTM 的细胞状态会决定哪些状态应该被留下来，哪些状态应该被遗忘。

下面来看一些 RNN 和 LSTM 内部结构的不同：

RNN



LSTM



由上面两幅图可以观察到，LSTM 结构更为复杂，在 RNN 中，将过去的输出和当前的输入 concatenate 到一起，通过 tanh 来控制两者的输出，它只考虑最近时刻的状态。在 RNN 中有两个输入和一个输出。

而 LSTM 为了能记住长期的状态，在 RNN 的基础上增加了一路输入和一路输出，增加的这一路就是细胞状态，也就是途中最上面的一条通路。事实上整个 LSTM 分成了三个部分：

- 1) 哪些细胞状态应该被遗忘
- 2) 哪些新的状态应该被加入
- 3) 根据当前的状态和现在的输入，输出应该是什么

下面来分别讨论：

- 1) 哪些细胞状态应该被遗忘

这部分功能是通过 sigmoid 函数实现的，也就是最左边的通路。根据输入和上一时刻的输出来决定当前细胞状态是否有需要被遗忘的内容。举个例子，如果之前细胞状态中有主语，而输入中又有了主语，那么原来存在的主语就应该被遗忘。concatenate 的输入和上一时刻的输出经过 sigmoid 函数后，越接近于 0 被遗忘的越多，越接近于 1 被遗忘的越少。

#### 2) 哪些新的状态应该被加入

继续上面的例子，新进来的主语自然就是应该被加入到细胞状态的内容，同理也是靠 sigmoid 函数来决定应该记住哪些内容。但是值得一提的是，需要被记住的内容并不是直接

concatenate 的输入和上一时刻的输出，还要经过 tanh，这点应该也是和 RNN 保持一致。并且需要注意，此处的 sigmoid 和前一步的 sigmoid 层的 w 和 b 不同，是分别训练的层。

细胞状态在忘记了该忘记的，记住了该记住的之后，就可以作为下一时刻的细胞状态输入了。

#### 3) 根据当前的状态和现在的输入，输出应该是什么

这是最右侧的通路，也是通过 sigmoid 函数做门，对第二步求得的状态做 tanh 后的结果过滤，从而得到最终的预测结果。

事实上，LSTM 就是在 RNN 的基础上，增加了对过去状态的过滤，从而可以选择哪些状态对当前更有影响，而不是简单的选择最近的状态。

在这之后，研究者们实现了各种 LSTM 的变种网络。不变的是，通常都会用 sigmoid 函数做门，筛选状态或者输入。并且输出都是要经过 tanh 函数。具体为什么要用这两个函数，由于刚接触还不能给出一定的解释，日后理解了再补充。

### 43、卷积层和池化层有什么区别

标签：深度学习

参考回答：

	卷积层	池化层
功		
能	提取特征	压缩特征图,提取主要特征
操	可惜是二维的,对于三维数据比如 RGB 图像 (3 通道),卷积核的深度必须同输入的通道数,	池化只是在二维数据上操作的,因此不改变输入的通

作	输出的通道数等于卷积核的个数。	道数。对于多通道的输入，这一点和卷积区别很大。
	卷积操作会改变输入特征图的通道数。	
特	权值共享：减少了参数的数量，并利用了图像目标的位置无关性。	
性	稀疏连接：输出的每个值只依赖于输入的部分值。	

#### 44、防止过拟合有哪些方法

标签：深度学习

参考回答：

1) Dropout ; 2) 加 L1/L2 正则化; 3) BatchNormalization ; 4) 网络 bagging

#### 45、dropout 咋回事讲讲

标签：深度学习

参考回答：

Dropout 的目标是在指数级数量的神经网络上近似这个过程。Dropout 训练与 Bagging 训练不太一样。在 Bagging 的情况下，所有模型是独立的。

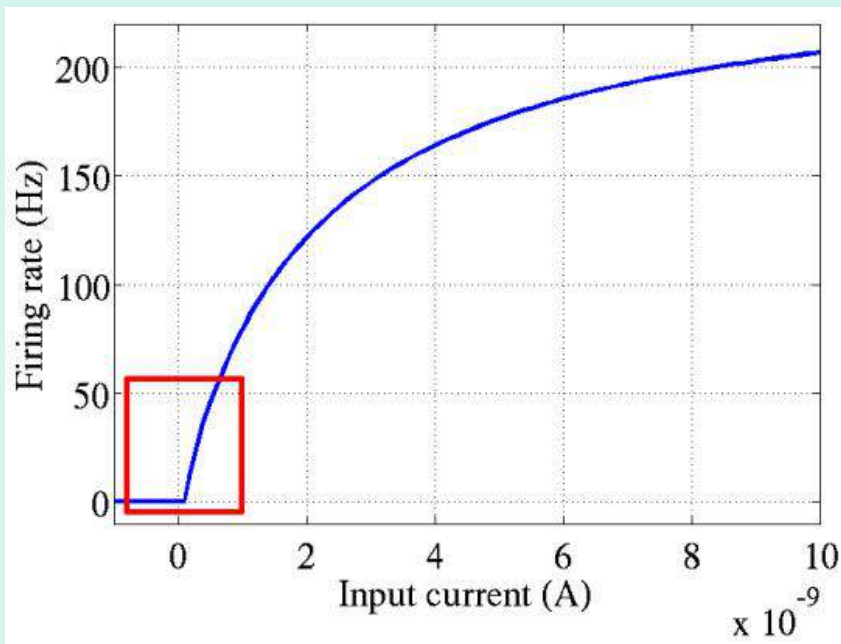
在 Dropout 的情况下，模型是共享参数的，其中每个模型继承的父神经网络参数的不同子集。参数共享使得在有限可用的内存下代表指数数量的模型变得可能。在 Bagging 的情况下，每一个模型在其相应训练集上训练到收敛。

在 Dropout 的情况下，通常大部分模型都没有显式地被训练，通常该模型很大，以致到宇宙毁灭都不能采样所有可能的子网络。取而代之的是，可能的子网络的一小部分训练单个步骤，参数共享导致剩余的子网络能有好的参数设定。

#### 46、relu

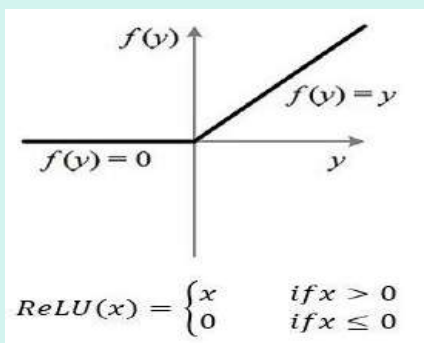
标签：深度学习

参考回答：在深度神经网络中，通常使用一种叫修正线性单元 (Rectified linear unit, ReLU) 作为神经元的激活函数。ReLU 起源于神经科学的研究：2001 年，Dayan、Abott 从生物学角度模拟出了脑神经元接受信号更精确的激活模型，如下图：



其中横轴是时间(ms)，纵轴是神经元的放电速率(Firing Rate)。同年，Attwell 等神经科学家通过研究大脑的能量消耗过程，推测神经元的工作方式具有稀疏性和分布性；2003 年 Lennie 等神经科学家估测大脑同时被激活的神经元只有 1~4%，这进一步表明了神经元的工作稀疏性。而对于 ReLU 函数而言，类似表现是如何体现的？其相比于其他线性函数(如 purlin)和非线性函数(如 sigmoid、双曲正切)又有何优势？下面请各位看官容我慢慢道来。

首先，我们来看一下 ReLU 激活函数的形式，如下图：



从上图不难看出，ReLU 函数其实是分段线性函数，把所有的负值都变为 0，而正值不变，这种操作被成为单侧抑制。可别小看这个简单的操作，正因为有了这单侧抑制，才使得神经网络中的神经元也具有了稀疏激活性。尤其体现在深度神经网络模型(如 CNN)中，当模型增加 N 层之后，理论上 ReLU 神经元的激活率将降低 2 的 N 次方倍。这里或许有童鞋会问：ReLU 的函数图像为什么一定要长这样？反过来，或者朝下延伸行不行？其实还不一定要长这样。只要能起到单侧抑制的作用，无论是镜面翻转还是 180 度翻转，最终神经元的输出也只是相当于加上了一个常数项系数，并不影响模型的训练结果。之所以这样定，或许是为了契合生物学角度，便于我们理解吧。

那么问题来了：这种稀疏性有何作用？换句话说，我们为什么需要让神经元稀疏？不妨举栗子来说明。当看名侦探柯南的时候，我们可以根据故事情节进行思考和推理，这时用到的是我们的大脑左半球；而当看蒙面唱将时，我们可以跟着歌手一起哼唱，这时用到的则是我们的右半球。左半球侧重理性思维，而右半球侧重感性思维。也就是说，当我们在进行运算或者欣赏时，都会

有一部分神经元处于激活或是抑制状态，可以说是各司其职。再比如，生病了去医院看病，检查报告里面上百项指标，但跟病情相关的通常只有那么几个。与之类似，当训练一个深度分类模型的时候，和目标相关的特征往往也就那么几个，因此通过 ReLU 实现稀疏后的模型能够更好地挖掘相关特征，拟合训练数据。

此外，相比于其它激活函数来说，ReLU 有以下优势：对于线性函数而言，ReLU 的表达能力更强，尤其体现在深度网络中；而对于非线性函数而言，ReLU 由于非负区间的梯度为常数，因此不存在梯度消失问题 (Vanishing Gradient Problem)，使得模型的收敛速度维持在一个稳定状态。这里稍微描述一下什么是梯度消失问题：当梯度小于 1 时，预测值与真实值之间的误差每传播一层会衰减一次，如果在深层模型中使用 sigmoid 作为激活函数，这种现象尤为明显，将导致模型收敛停滞不前。

#### 47、神经网络为啥用交叉熵。

标签：深度学习

参考回答：通过神经网络解决多分类问题时，最常用的一种方式就是在最后一层设置  $n$  个输出节点，无论在浅层神经网络还是在 CNN 中都是如此，比如，在 AlexNet 中最后的输出层有 1000 个节点，而即便是 ResNet 取消了全连接层，也会在最后有一个 1000 个节点的输出层。

一般情况下，最后一个输出层的节点个数与分类任务的目标数相等。假设最后的节点数为  $N$ ，那么对于每一个样例，神经网络可以得到一个  $N$  维的数组作为输出结果，数组中每一个维度会对应一个类别。在最理想的情况下，如果一个样本属于  $k$ ，那么这个类别所对应的的输出节点的输出值应该为 1，而其他节点的输出都为 0，即  $[0, 0, 1, 0, \dots, 0, 0]$ ，这个数组也就是样本的 Label，是神经网络最期望的输出结果，交叉熵就是用来判定实际的输出与期望的输出的接近程度。

#### 48、注意力公式

标签：深度学习

参考回答：

Soft attention、global attention、动态 attention

Hard attention

静态 attention “半软半硬” 的 attention (local attention)

强制前向 attention

$$\text{Attention}(\text{Query}, \text{Source}) = \sum_{i=1}^{L_x} a_i \cdot \text{Value}_i$$

#### 49、论文 flow 情况

标签：深度学习

参考回答：



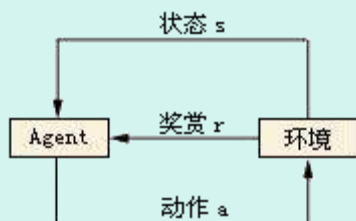
谈谈自己投稿的论文，论文投稿级别，论文内容，用到的方法，对比方法等

## 50、Flappy.Bird 开发者, 怎么利用 DNQ 方法强化学习你的游戏 AI

标签：深度学习

参考回答：

强化学习是机器学习里面的一个分支。它强调如何基于环境而行动，以取得最大化的预期收益。其灵感来源于心理学中的行为主义理论，既有机体如何在环境给予的奖励或者惩罚的刺激下，逐步形成对刺激的预期，产生能够最大利益的习惯性行为。结构简图如下：



因为强化学习考虑到了自主个体、环境、奖励等因素，所以很多人包括强化学习的研究者 Richard Sutton 都认为它是人工智能中最高层的模型，其它深度学习、机器学习模型都是它的子系统。在围棋界先后打败世界冠军的李世石和柯洁的 alphaGo 就使用了强化学习模型，也正是这两次比赛，把人工智能这个概念传递给了大众。使用的是卷积神经网络结构。

## 51、LeNet-5 结构

标签：深度学习

参考回答：

输入层：32\* 32\* 32 的图片，也就是相当于 1024\*1024 个神经元

C1 层：选取 66 个特征卷积核，大小为 5\* 5\* 5(不包含偏置)，得到 66 个特征图，每个特征图的大小为 32-5+1=28\* 32-5+1=28，也就是神经元的个数由 1024\*1024 减小到了 28\* 28=784\* 28=784。

输入层与 C1 层之间的参数： $6 * (5 * 5 + 1) * 6 * (5 * 5 + 1)$ ，对于卷积层 C1，每个像素都与前一层的 5\* 5\* 5 个像素和 11 个 bias 有连接，有  $6 * (5 * 5 + 1) * (28 * 28) * 6 * (5 * 5 + 1) * (28 * 28)$  个连接

S2 层：池化，是一个下采样层（为什么是下采样？利用图像局部相关性的原理，对图像进行子抽样，可以减少数据处理量同时保留有用信息），有 66 个 14\* 14\* 14 的特征图，特征图中的每个单元与 C1 中相对应特征图的 2\* 2\* 2 邻域相连接。S2 层每个单元对应 C1 中 44 个求和，乘以一个可训练参数，再加上一个可训练偏置。



C1 与 S2 之间的参数:每一个  $2 \times 22 \times 2$  求和, 然后乘以一个参数, 加上一个偏置, 共计  $2 \times 6 = 12 \times 6 = 12$  个参数。S2S2 中的每个像素都与 C1C1 中的  $2 \times 22 \times 2$  个像素和 11 个偏置相连接, 所以有  $6 \times 5 \times 14 \times 14 = 5880$  个连接

C3 层:选取卷积核大小为  $5 \times 55 \times 5$ , 得到新的图片大小为  $10 \times 1010 \times 10$  我们知道 S2 包含:6 张  $14 \times 146$  张  $14 \times 14$  大小的图片, 我们希望这一层得到的结果是: 16 张  $10 \times 1016$  张  $10 \times 10$  的图片。这 1616 张图片的每一张, 是通过 S2S2 的 66 张图片进行加权组合得到的, 具体是怎么组合的呢?

S2 与 C3 之间的组合

前 66 个 feature map 与 S2S2 层相连的 33 个 feature map 相连接, 后面 66 个 feature map 与 S2 层相连的 4 个 S2 层相连的 4 个 feature map 相连接, 后面 33 个 feature map 与 S2S2 层部分不相连的 44 个 feature map 相连接, 最后一个与 S2S2 层的所有 feature map 相连。卷积核大小依然为  $5 \times 55 \times 5$ , 总共有  $6 \times (3 \times 5 \times 5 + 1) + 6 \times (3 \times 5 \times 5 + 1) + 6 \times (4 \times 5 \times 5 + 1) + 6 \times (4 \times 5 \times 5 + 1) + 3 \times (4 \times 5 \times 5 + 1) + 3 \times (4 \times 5 \times 5 + 1) + 1 \times (6 \times 5 \times 5 + 1) = 15161 \times (6 \times 5 \times 5 + 1) = 1516$  个参数。而图像大小为  $10 \times 1010 \times 10$ , 所以共有 151600151600 个连接。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X	X	X	X	X	X	X	X	X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I  
EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED  
BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

S4 层

池化, 窗口大小为  $2 \times 22 \times 2$ , 有 1616 个特征图, 总共有 3232 个参数

C3 与 S4 之间的参数

$16 \times (25 \times 4 + 25) = 2000$   $16 \times (25 \times 4 + 25) = 2000$  个连接

C5 层

总共 120120 个 feature map, 每个 feature map 与 S4S4 层所有的 feature map 相连接, 卷积核大小是  $5 \times 55 \times 5$ , 而 S4S4 层的 feature map 的大小也是  $5 \times 55 \times 5$ , 所以 C5C5 的 feature map 就变成了 1 个点, 共计有  $120 (25 \times 16 + 1) = 48120$   $120 (25 \times 16 + 1) = 48120$  个参数。

F6 层

全连接



F6F6 相当于 MLP 中的隐含层，有 8484 个节点，所以有  $84 * (120+1) = 1016484 * (120+1) = 10164$  个参数。F6F6 层采用了正切函数。

输出层

采用了 RBF 函数，即径向欧式距离函数

## 52、推导 LSTM 正向传播和单向传播过程

标签：深度学习

参考回答：

前向推导过程：

*Forget Gates*

$$a_{\phi}^t = \sum_{i=1}^I w_{i\phi} x_i^t + \sum_{h=1}^H w_{h\phi} b_h^{t-1} + \sum_{c=1}^C w_{c\phi} s_c^{t-1} \quad (4.4)$$

$$b_{\phi}^t = f(a_{\phi}^t) \quad (4.5)$$

*Cells*

$$a_c^t = \sum_{i=1}^I w_{ic} x_i^t + \sum_{h=1}^H w_{hc} b_h^{t-1} \quad (4.6)$$

$$s_c^t = b_{\phi}^t s_c^{t-1} + b_i^t g(a_c^t) \quad (4.7)$$

*Output Gates*

$$a_{\omega}^t = \sum_{i=1}^I w_{i\omega} x_i^t + \sum_{h=1}^H w_{h\omega} b_h^{t-1} + \sum_{c=1}^C w_{c\omega} s_c^t \quad (4.8)$$

$$b_{\omega}^t = f(a_{\omega}^t) \quad (4.9)$$

*Cell Outputs*

$$b_c^t = b_{\omega}^t h(s_c^t) \quad (4.10)$$

反向推导过程：



$$\epsilon_c^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial b_c^t} \quad \epsilon_s^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial s_c^t}$$

Cell Outputs

$$\epsilon_c^t = \sum_{k=1}^K w_{ck} \delta_k^t + \sum_{g=1}^G w_{cg} \delta_g^{t+1} \quad (4.11)$$

Output Gates

$$\delta_\omega^t = f'(a_\omega^t) \sum_{c=1}^C h(s_c^t) \epsilon_c^t \quad (4.12)$$

States

$$\epsilon_s^t = b_\omega^t h'(s_c^t) \epsilon_c^t + b_\phi^{t+1} \epsilon_s^{t+1} + w_{ci} \delta_i^{t+1} + w_{c\phi} \delta_\phi^{t+1} + w_{c\omega} \delta_\omega^t \quad (4.13)$$

Cells

$$\delta_c^t = b_i^t g'(a_c^t) \epsilon_s^t \quad (4.14)$$

Forget Gates

$$\delta_\phi^t = f'(a_\phi^t) \sum_{c=1}^C s_c^{t-1} \epsilon_s^t \quad (4.15)$$

Input Gates

$$\delta_i^t = f'(a_i^t) \sum_{c=1}^C g(a_c^t) \epsilon_s^t \quad (4.16)$$

### 53、LSTM 原理，与 GRU 区别

标签：深度学习

参考回答：

LSTM 算法全称为 Long short-term memory，是一种特定形式的 RNN（Recurrent neural network，循环神经网络），而 RNN 是一系列能够处理序列数据的神经网络的总称。

RNN 在处理长期依赖（时间序列上距离较远的节点）时会遇到巨大的困难，因为计算距离较远的节点之间的联系时会涉及雅可比矩阵的多次相乘，这会带来梯度消失（经常发生）或者梯度膨胀（较少发生）的问题，这样的现象被许多学者观察到并独立研究。为了解决该问题，研究人员提出 LSTM。

LSTM 是门限 RNN，其单一节点的结构如下图 1 所示。LSTM 的巧妙之处在于通过增加输入门限，遗忘门限和输出门限，使得自循环的权重是变化的，这样一来在模型参数固定的情况下，不同时刻的积分尺度可以动态改变，从而避免了梯度消失或者梯度膨胀的问题。

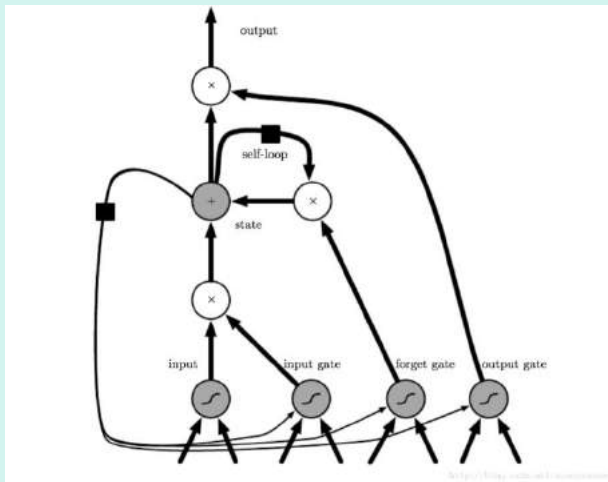


图 1 LSTM 的 CELL 示意图

根据 LSTM 网络的结构，每个 LSTM 单元的计算公式如下图 2 所示，其中  $F_t$  表示遗忘门限， $I_t$  表示输入门限， $C_t$  表示前一时刻 cell 状态、 $\tilde{C}_t$  表示 cell 状态（这里就是循环发生的地方）， $O_t$  表示输出门限， $H_t$  表示当前单元的输出， $H_{t-1}$  表示前一时刻单元的输出。

$$\begin{aligned}f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\h_t &= o_t * \tanh(C_t)\end{aligned}$$

图 2 LSTM 计算公式

与 GRU 区别：1) GRU 和 LSTM 的性能在很多任务上不分伯仲。2) GRU 参数更少因此更容易收敛，但是数据集很大的情况下，LSTM 表达性能更好。3) 从结构上来说，GRU 只有两个门（update 和 reset），LSTM 有三个门（forget, input, output），GRU 直接将 hidden state 传给下一个单元，而 LSTM 则用 memory cell 把 hidden state 包装起来。

#### 54、DNN 的梯度更新方式

参考回答：

##### 1) 批量梯度下降法 BGD

批量梯度下降法（Batch Gradient Descent，简称 BGD）是梯度下降法最原始的形式，它的具体思路是在更新每一参数时都使用所有的样本来进行更新，其数学形式如下：



(1) 对上述的能量函数求偏导：

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m} \sum_{i=1}^m (y^i - h_{\theta}(x^i)) x_j^i$$

(2) 由于是最小化风险函数，所以按照每个参数的梯度负方向来更新每个：

$$\theta_j' = \theta_j + \frac{1}{m} \sum_{i=1}^m (y^i - h_{\theta}(x^i)) x_j^i$$

### 2) 随机梯度下降法 SGD

由于批量梯度下降法在更新每一个参数时，都需要所有的训练样本，所以训练过程会随着样本数量的加大而变得异常的缓慢。随机梯度下降法（Stochastic Gradient Descent，简称 SGD）正是为了解决批量梯度下降法这一弊端而提出的。

将上面的能量函数写为如下形式：

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (y^i - h_{\theta}(x^i))^2 = \frac{1}{m} \sum_{i=1}^m \text{cost}(\theta, (x^i, y^i))$$

$$\text{cost}(\theta, (x^i, y^i)) = \frac{1}{2} (y^i - h_{\theta}(x^i))^2$$

利用每个样本的损失函数对求偏导得到对应的梯度，来更新：

$$\theta_j' = \theta_j + (y^i - h_{\theta}(x^i)) x_j^i$$

### 3) 小批量梯度下降法 MBGD

有上述的两种梯度下降法可以看出，其各自均有优缺点，那么能不能在两种方法的性能之间取得一个折衷呢？即，算法的训练过程比较快，而且也要保证最终参数训练的准确率，而这正是小批量梯度下降法（Mini-batch Gradient Descent，简称 MBGD）的初衷。

## 55、CNN 为什么比 DNN 在图像识别上更好

标签：深度学习

参考回答：

DNN 的输入是向量形式，并未考虑到平面的结构信息，在图像和 NLP 领域这一结构信息尤为重要，例如识别图像中的数字，同一数字与所在位置无关（换句话说任一位置的权重都应相同），CNN 的输入可以是 tensor，例如二维矩阵，通过 filter 获得局部特征，较好的保留了平面结构信息。

56、现场用 collabedit 写代码，一个怪异的归并算法。。。之前没遇到过，直接把归并写出来，但是说复杂度太高，优化了三遍还不行，最后说出用小顶堆解决了。。。



标签：数据结构与算法

参考回答：

归并算法：

```
#include <iostream>

using namespace std;

//将有二个有序数列 a[first...mid]和 a[mid...last]合并。

void __merge(int a[], int first, int mid, int last, int temp[])
{
    int i = first, j = mid + 1;

    int m = mid, n = last;

    int k = 0;

    while (i <= m && j <= n)
    {
        if (a[i] <= a[j])
            temp[k++] = a[i++];

        else
            temp[k++] = a[j++];
    }

    while (i <= m)
        temp[k++] = a[i++];

    while (j <= n)
        temp[k++] = a[j++];

    for (i = 0; i < k; i++)
        a[first + i] = temp[i];
}
```





```
void __merge_sort(int a[], int first, int last, int temp[])

{

    if (first < last)

    {

        int mid = (first + last) / 2;

        __merge_sort(a, first, mid, temp);

        __merge_sort(a, mid + 1, last, temp);

        __merge(a, first, mid, last, temp);

    }

}

bool MergeSort(int a[], int n)

{

    int *p = new int[n];

    if (p == NULL)

    {

        return false;

    }

    else

    {

        __merge_sort(a, 0, n - 1, p);

        delete[] p;

        return true;

    }

}
```



```
int main()

{

    const int LEN = 10;

    int a[LEN] = { 23, 40, 45, 19, 12, 16, 90, 39, 87, 71 };

    cout<<"Before the merge sort, the Array is:"<<endl;

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

    {

        cout<<a[i]<<" ";

    }

    cout<<endl;

    cout<<endl;

    MergeSort(a, LEN);

    cout<<"After the merge sort, the Array is:"<<endl;

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

    {

        cout<<a[i]<<" ";

    }

    cout<<endl;

    system("pause");

    return 0;

}
```

小顶堆:

```
import java.util.Arrays;
```

```
public class SmallHeap {
```



```
public static int[] topN(int[] arr, int n) {

    int[] list = new int[n];

    System.arraycopy(arr, 0, list, 0, n);

    for (int i = 0; i < n; i++) {

        int t = i;

        while (t != 0 && list[parent(t)] > list[t]) {

            swap(list, t, t = parent(t));

        }

    }

    for (int i = n, len = arr.length; i < len; i++) {

        if (arr[i] >= list[0]) {

            // 置换栈顶

            list[0] = arr[i];

            // 调整栈顶

            int t = 0;

            while((left(t)<n&&list[t]>list[left(t)])||(right(t)<n&&list[t]>list[right(t)))) {

                if (right(t) < n && list[right(t)] < list[left(t)]) {

                    swap(list, t, t = right(t));

                } else {

                    swap(list, t, t = left(t));

                }

            }

        }

    }

    return list;

}
```



```
    }

    private static void swap(int[] list, int i, int j) {

        int tmp = list[i];

        list[i] = list[j];

        list[j] = tmp;

    }

    private static int parent(int i) {

        return (i - 1) / 2;

    }

    private static int left(int i) {

        return 2 * i + 1;

    }

    private static int right(int i) {

        return 2 * i + 2;

    }

    public static void main(String[] args) {

        int[] arr = new int[] {56, 30, 71, 18, 29, 93, 44, 75, 20, 65, 68, 34};

        System.out.println("原始数组: ");

        System.out.println(Arrays.toString(arr));

        System.out.println("调整后数组: ");

        System.out.println(Arrays.toString(SmallHeap.topN(arr, 5)));

    }

}
```

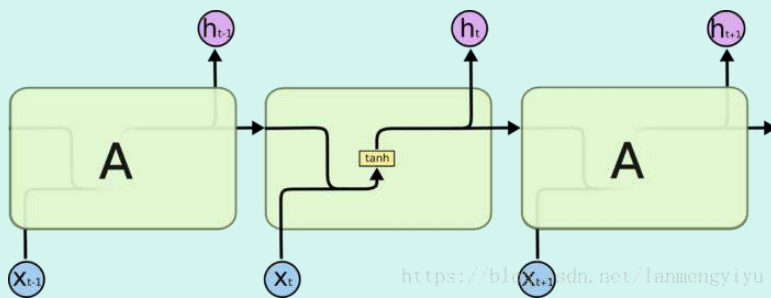
## 57、LSTM 和 Naive RNN 的区别

标签：深度学习

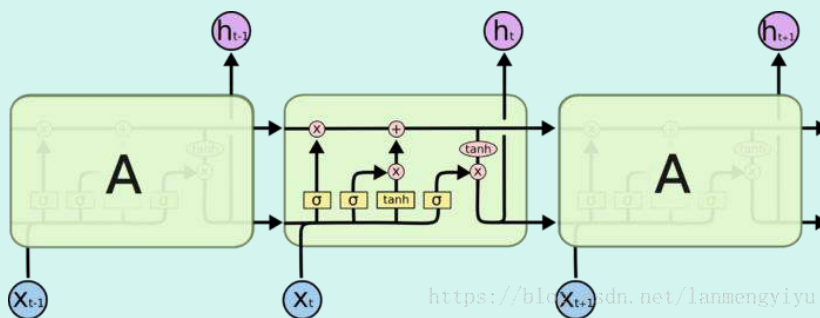


参考回答：

RNN 和 LSTM 内部结构的不同：



RNN



LSTM

由上面两幅图可以观察到，LSTM 结构更为复杂，在 RNN 中，将过去的输出和当前的输入 concatenate 到一起，通过 tanh 来控制两者的输出，它只考虑最近时刻的状态。在 RNN 中有两个输入和一个输出。

而 LSTM 为了能记住长期的状态，在 RNN 的基础上增加了一路输入和一路输出，增加的这一路就是细胞状态，也就是途中最上面的一条通路。事实上整个 LSTM 分成了三个部分：

- 1) 哪些细胞状态应该被遗忘
- 2) 哪些新的状态应该被加入
- 3) 根据当前的状态和现在的输入，输出应该是什么

下面来分别讨论：

- 1) 哪些细胞状态应该被遗忘

这部分功能是通过 sigmoid 函数实现的，也就是最左边的通路。根据输入和上一时刻的输出来决定当前细胞状态是否有需要被遗忘的内容。举个例子，如果之前细胞状态中有主语，而输入中又有了主语，那么原来存在的主语就应该被遗忘。concatenate 的输入和上一时刻的输出经过 sigmoid 函数后，越接近于 0 被遗忘的越多，越接近于 1 被遗忘的越少。

- 2) 哪些新的状态应该被加入



继续上面的例子，新进来的主语自然就是应该被加入到细胞状态的内容，同理也是靠 sigmoid 函数来决定应该记住哪些内容。但是值得一提的是，需要被记住的内容并不是直接 concatenate 的输入和上一时刻的输出，还要经过 tanh，这点应该也是和 RNN 保持一致。并且需要注意，此处的 sigmoid 和前一步的 sigmoid 层的 w 和 b 不同，是分别训练的层。细胞状态在忘记了该忘记的，记住了该记住的之后，就可以作为下一时刻的细胞状态输入了。

3) 根据当前的状态和现在的输入，输出应该是什么

这是最右侧的通路，也是通过 sigmoid 函数做门，对第二步求得的状态做 tanh 后的结果过滤，从而得到最终的预测结果。事实上，LSTM 就是在 RNN 的基础上，增加了对过去状态的过滤，从而可以选择哪些状态对当前更有影响，而不是简单的选择最近的状态。

## 58、神经网络为啥用交叉熵。

标签：深度学习

参考回答：

通过神经网络解决多分类问题时，最常用的一种方式就是在最后一层设置 n 个输出节点，无论在浅层神经网络还是在 CNN 中都是如此，比如，在 AlexNet 中最后的输出层有 1000 个节点，而即便是 ResNet 取消了全连接层，也会在最后有一个 1000 个节点的输出层。

一般情况下，最后一个输出层的节点个数与分类任务的目标数相等。假设最后的节点数为 N，那么对于每一个样例，神经网络可以得到一个 N 维的数组作为输出结果，数组中每一个维度会对应一个类别。在最理想的情况下，如果一个样本属于 k，那么这个类别所对应的输出节点的输出值应该为 1，而其他节点的输出都为 0，即  $[0, 0, 1, 0, \dots, 0, 0]$ ，这个数组也就是样本的 Label，是神经网络最期望的输出结果，交叉熵就是用来判定实际的输出与期望的输出的接近程度。

## 59、注意力公式

参考回答：

Soft attention、global attention、动态 attention

Hard attention

“半软半硬”的 attention (local attention)

静态 attention

强制前向 attention

$$\text{Attention}(\text{Query}, \text{Source}) = \sum_{i=1}^{L_x} a_i \cdot \text{Value}_i$$

## 60、Inception Score 评价指标介绍

参考回答：





定义：

$$IS(G) = \exp(E_{x \sim p_g} D_{KL}(p(y|x) || p(y)))$$

推导出上式的意义：

$$\begin{aligned} \ln(IS(G)) &= E_{x \sim p_g} D_{KL}(p(y|x) || p(y)) \\ &= \sum_x p(x) D_{KL}(p(y|x) || p(y)) \\ &= \sum_x p(x) \sum_i p(y=i|x) \ln\left(\frac{p(y=i|x)}{p(y=i)}\right) \\ &= \sum_x \sum_i p(x, y=i) \ln \frac{p(x, y=i)}{p(x)p(y=i)} \\ &= I(y; x) \\ &= H(y) - H(y|x) \end{aligned}$$

故要使得生成图像的 inception score 高，就需要

1. 最大化  $H(y)$ ；也就是对于输入的样本，通过 inception\_v3 模型后的类别要均衡，衡量模式坍塌。
2. 最小化  $H(y|x)$ ；说明对于输入的样本，通过 inception\_v3 模型后预测某类别的置信度要高，衡量图片生成的质量。

## 61、使用的 CNN 模型权重之间有关联吗？

参考回答：

权重之间有关联。CNN 是权重共享，减少了参数的数量。

简单来说就是用一个卷积核来和一个图像来进行卷积，记住是同一个卷积核，不改变卷积核的值。这样可以减少权值参数。共享就是一个图片对卷积核是共同享有的。对于一个  $100 \times 100$  像素的图像，如果我们用一个神经元来对图像进行操作，这个神经元大小就是  $100 \times 100 = 10000$ ，单如果我们使用  $10 \times 10$  的卷积核，我们虽然需要计算多次，但我们需要的参数只有  $10 \times 10 = 100$  个，加上一个偏向  $b$ ，一共只需要 101 个参数。我们取得图像大小还是  $100 \times 100$ 。如果我们取得图像比较大，它的参数将会更加多。我们通过  $10 \times 10$  的卷积核对图像进行特征提取，这样我们就得到一个 Feature Map。

一个卷积核只能提取一个特征，所以我们需要多几个卷积核，假设我们有 6 个卷积核，我们就会得到 6 个 Feature Map，将这 6 个 Feature Map 组成一起就是一个神经元。这 6 个 Feature Map 我们需要  $101 \times 6 = 606$  个参数。这个值和 10000 比还是比较小的。如果像之前的神经网络，两两相连，需要  $28 \times 28 = 784$  输入层，加上第一个隐藏层 30 个神经元，则需要  $784 \times 30$  再加上 30 个  $b$ ，总共 23,550 个参数！多了 40 倍的参数。



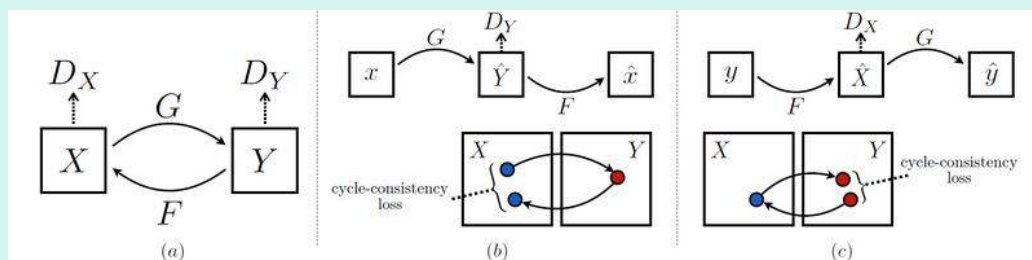
5、百度实习：1) 模型压缩方法；2) CPM 模型压缩用了哪些方法；3) 压缩效果（体积、指标、部署）；4) Kaggle 比赛，比赛背景，怎么进行数据清洗，类别平衡，相近类别重分类，最终成绩是多少，觉得跟前几名差距在哪，有没有尝试过集成的方法；5) 人脸项目，大概流程，GPU 加速的地方，两个网络的训练过程，级联网络的 inference 过程，能同时检测多个人脸吗？多尺度缩放怎么处理，resize 自己写？只是检测吗，有没有识别？或者其他

## 62、CycleGAN 原理介绍一下

标签：深度学习

参考回答：

CycleGAN 其实就是一个  $A \rightarrow B$  单向 GAN 加上一个  $B \rightarrow A$  单向 GAN。两个 GAN 共享两个生成器，然后各自带一个判别器，所以加起来总共有两个判别器和两个生成器。一个单向 GAN 有两个 loss，而 CycleGAN 加起来总共有四个 loss。CycleGAN 论文的原版原理图和公式如下，其实理解了单向 GAN 那么 CycleGAN 已经很好理解。



**Figure 3:** (a) Our model contains two mapping functions  $G: X \rightarrow Y$  and  $F: Y \rightarrow X$ , and associated adversarial discriminators  $D_Y$  and  $D_X$ .  $D_Y$  encourages  $G$  to translate  $X$  into outputs indistinguishable from domain  $Y$ , and vice versa for  $D_X$ ,  $F$ , and  $X$ . To further regularize the mappings, we introduce two “cycle consistency losses” that capture the intuition that if we translate from one domain to the other and back again we should arrive where we started: (b) forward cycle-consistency loss:  $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ , and (c) backward cycle-consistency loss:  $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$ .

$X \rightarrow Y$  的判别器损失为，字母换了一下，和上面的单向 GAN 是一样的：

$$L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)} [\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)} [\log (1 - D_Y(G(x)))]$$

同理  $Y \rightarrow X$  的判别器损失为

$$L_{GAN}(F, D_X, Y, X) = \mathbb{E}_{x \sim p_{data}(x)} [\log D_X(x)] + \mathbb{E}_{y \sim p_{data}(y)} [\log (1 - D_X(F(y)))]$$

而两个生成器的 loss 加起来表示为：

$$L_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)} [\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)} [\|G(F(y)) - y\|_1]$$

最终网络的所有损失加起来为：

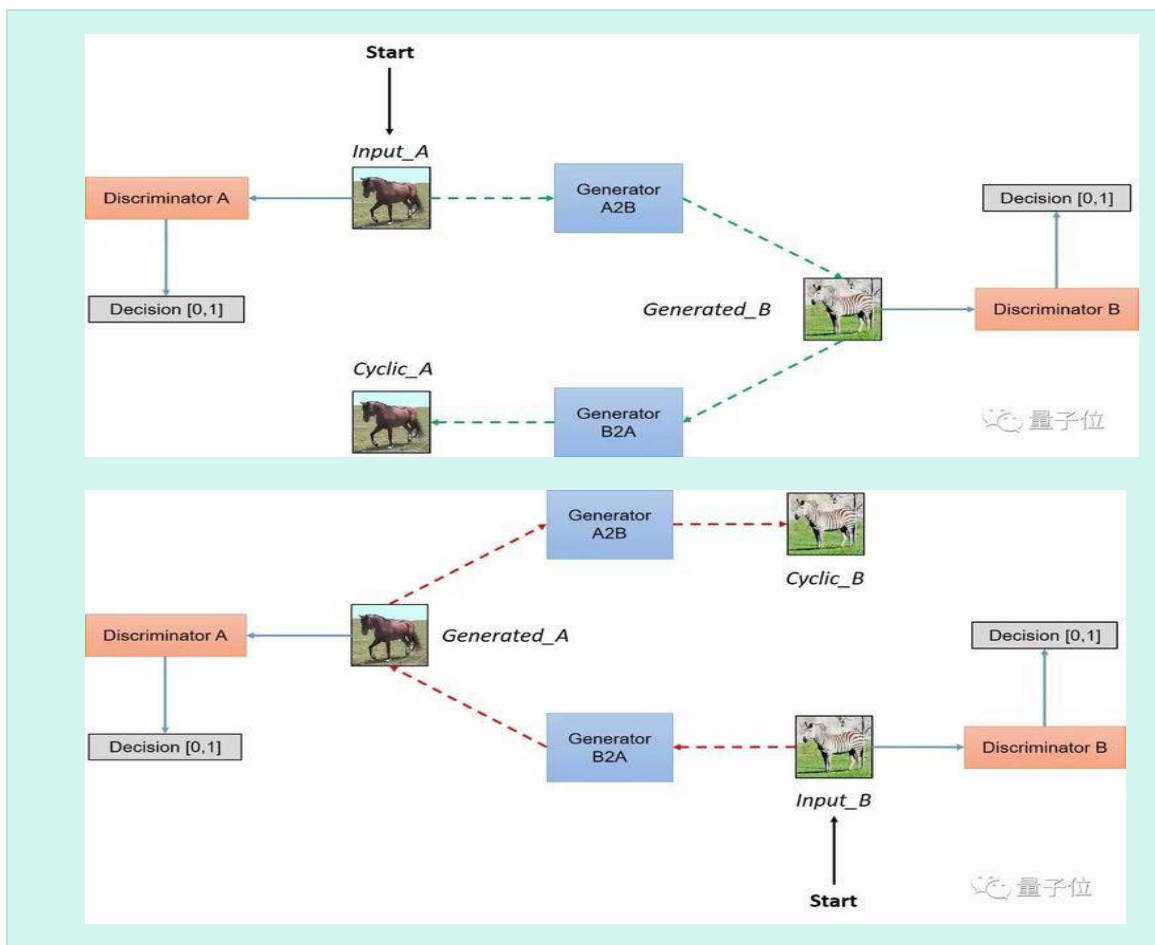
$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + L_{cyc}(G, F)$$

• Note

论文里面提到判别器如果是对数损失训练不是很稳定，所以改成的均方误差损失，如下

$$L_{LSGAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)} [(D_Y(y) - 1)^2] + \mathbb{E}_{x \sim p_{data}(x)} [(1 - D_Y(G(x)))^2]$$

下面放一张网友们自制的 CycleGAN 示意图，比论文原版的更加直观，出处见水印。



### 63、训练 GAN 的时候有没有遇到什么问题

标签：深度学习

参考回答：

遇到 GAN 训练不稳定问题。通过 Wasserstein GAN 来解决这个问题。WGAN 前作分析了 Ian Goodfellow 提出的原始 GAN 两种形式各自的问题，第一种形式等价在最优判别器下等价于最小化生成分布与真实分布之间的 JS 散度，由于随机生成分布很难与真实分布有不可忽略的重叠以及 JS 散度的突变特性，使得生成器面临梯度消失的问题；第二种形式在最优判别器下等价于既要最小化生成分布与真实分布直接的 KL 散度，又要最大化其 JS 散度，相互矛盾，导致梯度不稳定，而且 KL 散度的不对称性使得生成器宁可丧失多样性也不愿丧失准确性，导致 collapse mode 现象。

WGAN 前作针对分布重叠问题提出了一个过渡解决方案，通过对生成样本和真实样本加噪声使得两个分布产生重叠，理论上可以解决训练不稳定的问题，可以放心训练判别器到接近最优，但是未能提供一个指示训练进程的可靠指标，也未做实验验证。

WGAN 本作引入了 Wasserstein 距离，由于它相对 KL 散度与 JS 散度具有优越的平滑特性，理论上可以解决梯度消失问题。接着通过数学变换将 Wasserstein 距离写成可求解的形式，利用一个参数数值范围受限的判别器神经网络来最大化这个形式，就可以近似 Wasserstein 距离。在此近似最优判别器下优化生成器使得 Wasserstein 距离缩小，就能有效拉近生成分布与真实分布。

WGAN 既解决了训练不稳定的问题，也提供了一个可靠的训练进程指标，而且该指标确实与生成样本的质量高度相关。

#### 64、CPM 模型压缩怎么做的？有压过 OpenPose 吗？

标签：深度学习

参考回答：

预测和图像特征计算模块可以被深度网络架构来取代，其中图像和组织特征的表达可以从数据中直接学习。卷积架构让全局可导，因此可以 CPM 所有阶段联合训练。CPM 可以描述为在 PM 隐含空间模型框架下的卷积架构。

##### 1) 用局部图线索来进行关键定位

第一阶段只用局部图线索来预测部件信任度。figure 2c 展示用本地图信息的部件检测的深度网络。先序哦是局部的因为第一阶段感知野只是输出像素附近的一小块。我们用 5 层卷积网络组成的结构（尾部是量个  $1 \times 1$  卷积层的全卷积架构）。实践中，为了得到一定精度，我们把图片标准化为  $368 \times 368$ ，感受野是  $160 \times 160$ 。网络可以看成让深度网络在图像上滑动，并将  $160 \times 160$  中局部图像线索回归至代表了各个部件在各个位置的 score 的  $P+1$  大小输出向量。

##### 2) 基于空间环境信息的级联预测

对于性状稳定的头和肩膀，检测效果很好，然而人体骨架的连接处准确率就很低，因为形状差异很大。部件周围的信任映射，虽然有噪声，但是很有价值。figure 3 中，当检测右手肘时，右肩膀的信任映射达到高峰，可以成为一个很强的线索。后续阶段的预测器（gt）可以用图位置  $z$  附近含有噪声的信任映射里的空间组织信息（fai），并且利用“部件的几何设定都是恒定的”这一事实来提高改善预测。

第二个阶段，分类器  $g_2$  接收特征  $x_2$  和前一阶段 fai 的输入。前一阶段不同部件的位置  $z$  附近的区域产生信任映射，特征方程是把信任映射出的特点编码。CPM 不用显式方程来计算环境特征，而是定义含有前一阶段信任度的 fai 作为预测机的感受野。

这个网络的设计为了在第二阶段输出层得到一个足够大的感知野，可以学习复杂和长距离的部件关系。通过应用迁移阶段的输出层特征（而不是用图模型的显式方程），后续卷积层自由结合最有预测力的特征，来形成环境信息。第一阶段的信任映射来自用小感知野来检验局部图像的网络。第二阶段，我们设计了一个极大扩充的等价感知野。大感知野可以用两种方法实现：牺牲准确度的池化，增加参数为代价的加大卷积核大小，或者冒着可能让反传消失风险增加网络层数。我们选择增加卷积层，在  $8 \times 8$  降维热力图上达到大感知野，让我们尽可能减少参数数量。8 步网络更容易获得大感知野，它和 4 步网络表现一样好（在高精确度区域也是）。我们也在 PM 之后图像特征上映射上重复了类似架构，让空间组织依赖图像而且允许错误关联。

我们发现，感受野变大，准确性也变大。通过一系列实验，figure 4 的准确度随着感受野的变化曲线，改变感受野只通过改变结构而不是增加参数。准确度随着感受野变大而变大，在 250 像素饱和，这也大概是归一化物体的大小。这说明，网络确实让远距离物体关系编码，并且这是有益的。我们最好的数据集中，我们把图像归一化为  $368 \times 368$ ，基于第一级信任映射的第二级感知野输出是  $31 \times 31$ ，这和原始图片的  $400 \times 400$  像素等价，其半径可以覆盖任何部件。当阶段增多，有效感知野就会变大。我们有 6 个阶段。





### 3) 用 CPM 学习

这个深度架构可以有許多层。训练这个网可能让梯度消失，就是反向传播在中间层会减弱。pm 级联预测框架有一个自然的解决这个问题方法。我们不断激励这个网络，通过在每个阶段  $t$  的输出定义一个损失函数，让预测的和实际信任映射的距离最小化。部件  $p$  理想的信任映射是  $bp$ ，通过把  $p$  部件的最可能点设定在 `ground truth` 位置。

压缩过 OpenPose，效果还可以。

## 65、用过哪些 Optimizer，效果如何

标签：深度学习

参考回答：

1) SGD; 2) Momentum; 3) Nesterov; 4) Adagrad; 5) Adadelta; 6) RMSprop; 7) Adam; 8) Adamax; 9) Nadam。(1) 对于稀疏数据，尽量使用学习率可自适应的算法，不用手动调节，而且最好采用默认参数。(2) SGD 通常训练时间最长，但是在好的初始化和学习率调度方案下，结果往往更可靠。但 SGD 容易困在鞍点，这个缺点也不能忽略。(3) 如果在意收敛的速度，并且需要训练比较深比较复杂的网络时，推荐使用学习率自适应的优化方法。(4) Adagrad, Adadelta 和 RMSprop 是比较相近的算法，表现都差不多。(5) 在能使用带动量的 RMSprop 或者 Adam 的地方，使用 Nadam 往往能取得更好的效果。

## 66、图像基础：传统图像处理方法知道哪些，图像对比度增强说一下

标签：深度学习

参考回答：

数字图像处理常用方法：

1) 图像变换：由于图像阵列很大，直接在空间域中进行处理，涉及计算量很大。因此，往往采用各种图像变换的方法，如傅立叶变换、沃尔什变换、离散余弦变换等间接处理技术，将空间域的处理转换为变换域处理，不仅可减少计算量，而且可获得更有效的处理（如傅立叶变换可在频域中进行数字滤波处理）。目前新兴研究的小波变换在时域和频域中都具有良好的局部化特性，它在图像处理中也有着广泛而有效的应用。

2) 图像编码压缩：图像编码压缩技术可减少描述图像的数据量（即比特数），以便节省图像传输、处理时间和减少所占用的存储器容量。压缩可以在不失真的前提下获得，也可以在允许的失真条件下进行。编码是压缩技术中最重要的方法，它在图像处理技术中是发展最早且比较成熟的技术。

3) 图像增强和复原：图像增强和复原的目的是为了提高图像的质量，如去除噪声，提高图像的清晰度等。图像增强不考虑图像降质的原因，突出图像中所感兴趣的部分。如强化图像高频分量，可使图像中物体轮廓清晰，细节明显；如强化低频分量可减少图像中噪声影响。图像复原要求对图像降质的原因有一定的了解，一般讲应根据降质过程建立“降质模型”，再采用某种滤波方法，恢复或重建原来的图像。



4) 图像分割：图像分割是数字图像处理中的关键技术之一。图像分割是将图像中有意义的特征部分提取出来，其有意义的特征有图像中的边缘、区域等，这是进一步进行图像识别、分析和理解的基础。虽然目前已研究出不少边缘提取、区域分割的方法，但还没有一种普遍适用于各种图像的有效方法。因此，对图像分割的研究还在不断深入之中，是目前图像处理中研究的热点之一。

5) 图像描述：图像描述是图像识别和理解的必要前提。作为最简单的二值图像可采用其几何特性描述物体的特性，一般图像的描述方法采用二维形状描述，它有边界描述和区域描述两类方法。对于特殊的纹理图像可采用二维纹理特征描述。随着图像处理研究的深入发展，已经开始进行三维物体描述的研究，提出了体积描述、表面描述、广义圆柱体描述等方法。

6) 图像分类（识别）：图像分类（识别）属于模式识别的范畴，其主要内容是图像经过某些预处理（增强、复原、压缩）后，进行图像分割和特征提取，从而进行判决分类。图像分类常采用经典的模式识别方法，有统计模式分类和句法（结构）模式分类，近年来新发展起来的模糊模式识别和人工神经网络模式分类在图像识别中也越来越受到重视。

全局对比度增强

#### 1. 直方图均衡化 Histogram Equalization

算法：

- 1) 根据图像灰度计算灰度概率密度函数 PDF
- 2) 计算累积概率分布函数 CDF
- 3) 将 CDF 归一化到原图灰度取值范围，如 $[0, 255]$ 。
- 4) 之后 CDF 四舍五入取整，得到灰度转换函数  $s_k = T(r_k)$
- 5) 将 CDF 作为转换函数，将灰度为  $r_k$  的点转换为  $s_k$  灰度

#### 2. 直方图匹配 Histogram Matching

算法：

- 1) 根据图像计算概率密度分布  $pr(r)$ ；
- 2) 根据  $pr(r)$  计算累计分布函数  $s_k = T(r_k)$ ；
- 3) 根据给定的目标分布  $p_z(z)$  计算累计分布函数  $G(z_q)$ ；
- 4) 对于每一个  $k$ ，找到一个  $q$ ，使得  $G(z_q)$  约等于  $s_k$ ；
- 5) 将原图中灰度为  $k$  的点变为灰度  $q$ ；

局部对比度增强

1. 邻域直方图均衡：将全局直方图均衡的思想应用于邻域直方图处理中。



2. 邻域直方图匹配：将全局直方图匹配的思想应用于邻域直方图处理中。

3. 邻域统计方法

算法

1) 初始化：增强常数  $E$ ，灰度下阈值  $k_0$ ，标准差下阈值  $k_1$ ，标准差上阈值  $k_2$ ，窗口半宽  $s$ ；

2) 计算图像灰度均值  $MG$  和灰度标准差  $\sigma G$ ；

3) 对于每一个像素，计算邻域（大小为  $2s+1$  的方块）内灰度均值  $ML$  和标准差  $\sigma L$ ；

4) 如果  $ML \leq k_0 * MG$  且  $ML \geq k_0 * MG$  并且  $k_1 * \sigma G \leq \sigma L \leq k_2 * \sigma G$ ，将像素灰度乘以  $E$ 。

## 67、介绍一下图像的高频、低频部分，知道哪些图像补全的方法

参考回答：

图像的频率：灰度值变化剧烈程度的指标，是灰度在平面空间上的梯度。

(1) 什么是低频？

低频就是颜色缓慢地变化，也就是灰度缓慢地变化，就代表着那是连续渐变的一块区域，这部分就是低频。对于一幅图像来说，除去高频的就是低频了，也就是边缘以内的内容为低频，而边缘内的内容就是图像的大部分信息，即图像的大致概貌和轮廓，是图像的近似信息。

(2) 什么是高频？

反过来，高频就是频率变化快。图像中什么时候灰度变化快？就是相邻区域之间灰度相差很大，这就是变化得快。图像中，一个影像与背景的边缘部位，通常会有明显的差别，也就是说变化那条边线那里，灰度变化很快，也即是变化频率高的部位。因此，图像边缘的灰度值变化快，就对应着频率高，即高频显示图像边缘。图像的细节处也是属于灰度值急剧变化的区域，正是因为灰度值的急剧变化，才会出现细节。

另外噪声（即噪点）也是这样，在一个像素所在的位置，之所以是噪点，就是因为它与正常的点颜色不一样了，也就是说该像素点灰度值明显不一样了，，也就是灰度有快速地变化了，所以是高频部分，因此有噪声在高频这么一说。

图像补全的方法：

Region Filling and Object Removal by Exemplar-Based Image Inpainting

算法的流程大致如下：

1) 对待补全区域边界的像素依次计算补全的优先度(priority)，这个优先度主要考虑2个因素。一个是周围像素可信度高的位置要优先补，另一个是位于图像梯度变化剧烈的位置要优先补。综合二者得到所有优先度之后，挑选优先度最高的像素来补





2) 对于上一步找到的待补全像素，考虑它周围的一个小 patch(比如  $3 \times 3$ )。在图像已知部分搜索所有的 patch，找到最相似的 patch

3) 用找到的 best match 来补全未知部分，并更新相关数值

但是我们也不难发现这个方法存在的问题：如果图像已知部分找不到相似的 patch，那算法将无法进行；这个方法只适用于补全背景以低频信息和重复性纹理为主的图像；搜索相似的 patch 计算复杂度非常高，算法运行效率低。

#### Scene Completion Using Millions of Photographs

算法的大致流程如下：

1) 从 Flickr 上下载两百万图片构建数据库，以 "landscape" "city" "park" 等关键词搜索户外场景的图片。

2) 对于一张待补全图像，从数据库中挑选 200 个场景最相似的图片，这里使用 gist scene descriptor 和图像下采样到  $4 \times 4$  作为匹配的特征向量。

3) 将补全区域边界外 80 个 pixel 的区域作为 context。对于每一张匹配的图片，搜索所有的平移空间和 3 个尺度的 scale 空间，根据 context 部分的匹配误差，选择最佳的补全位置；之后利用 graph-cut 算法求解最佳的融合边界。

4) 利用标准的泊松融合处理融合边界。

5) 将前几步的匹配 cost 和 graph-cut 的 cost 加起来，返回 cost 最小的 20 的结果供用户挑选。

#### Context Encoders: Feature Learning by Inpainting

文章提出的网络结构如下，包括 3 个部分：Encoder, Channel-wise fully-connected layer, Decoder。Encoder 的结构直接借鉴了 AlexNet 前 5 层的卷积层结构，具体结构如下。输入的 crop 尺寸是  $227 \times 227$ ，卷积之后得到的 feature map 结构是 256 层  $64 \times 64$ 。所有的 weight 都随机初始化。

Channel-wise fully-connected layer 是对普通 fc 层的一种改进。之所以加入 fc 层是为了使 feature map 每一层的信息可以在内部交流。但传统的 fc 层参数太多，因此作者提出可以在 fc 中去掉 feature map 层间的信息交流，从而减少参数规模。在 fc 之后会接一个 stride 为 1 的卷积层，来实现层间的信息交流。

Decoder 的目的是将压缩的 feature map 一步步放大，恢复到原始图片的尺寸。文章提出采用 5 个 up-convolutional 层，每层后接一个 RELU。上采样的结构如下。

### 68、模型压缩的大方向。CPM 模型怎么压缩的，做了哪些工作？

标签：深度学习

参考回答：

预测和图像特征计算模块可以被深度网络架构来取代，其中图像和组织特征的表达可以从数据中直接学习。卷积架构让全局可导，因此可以 CPM 所有阶段联合训练。CPM 可以描述为在 PM 隐含空间模型框架下的卷积架构。

### 1) 用局部图线索来进行关键定位

第一阶段只用局部图线索来预测部件信任度。figure 2c 展示用本地图信息的部件检测的深度网络。先序哦是局部的因为第一阶段感知野只是输出像素附近的一小块。我们用 5 层卷积网络组成的结构（尾部是量个  $1 \times 1$  卷积层的全卷积架构）。实践中，为了得到一定精度，我们把图片标准化为  $368 \times 368$ ，感受野是  $160 \times 160$ 。网络可以看成让深度网络在图像上滑动，并将  $160 \times 160$  中局部图像线索回归至代表了各个部件在各个位置的 score 的  $P+1$  大小输出向量。

### 2) 基于空间环境信息的级联预测

对于性状稳定的头和肩膀，检测效果很好，然而人体骨架的连接处准确率就很低，因为形状差异很大。部件周围的信任映射，虽然有噪声，但是很有价值。figure 3 中，当检测右手肘时，右肩膀的信任映射达到高峰，可以成为一个很强的线索。后续阶段的预测器（gt）可以用图位置  $z$  附近含有噪声的信任映射里的空间组织信息（fai），并且利用“部件的几何设定都是恒定的”这一事实来提高改善预测。

第二个阶段，分类器  $g_2$  接收特征  $x_2$  和前一阶段 fai 的输入。前一阶段不同部件的位置  $z$  附近的区域产生信任映射，特征方程是把信任映射出的特点编码。CPM 不用显式方程来计算环境特征，而是定义含有前一阶段信任度的 fai 作为预测机的感受野。

这个网络的设计为了在第二阶段输出层得到一个足够大的感知野，可以学习复杂和长距离的部件关系。通过应用迁移阶段的输出层特征（而不是用图模型的显式方程），后续卷积层自由结合最有预测力的特征，来形成环境信息。第一阶段的信任映射来自用小感知野来检验局部图像的网络。第二阶段，我们设计了一个极大扩充的等价感知野。大感知野可以用两种方法实现：牺牲准确度的池化，增加参数为代价的加大卷积核大小，或者冒着可能让反传消失风险增加网络层数。我们选择增加卷积层，在  $8 \times$  降维热力图上达到大感知野，让我们尽可能减少参数数量。8 步网络更容易获得大感知野，它和 4 步网络表现一样好（在高精确度区域也是）。我们也在 PM 之后图像特征上映射上重复了类似架构，让空间组织依赖图像而且允许错误关联。

我们发现，感受野变大，准确性也变大。通过一系列实验，figure 4 的准确度随着感受野的变化曲线，改变感受野只通过改变结构而不是增加参数。准确度随着感受野变大而变大，在 250 像素饱和，这也大概是归一化物体的大小。这说明，网络确实让远距离物体关系编码，并且这是有益的。我们最好的数据集中，我们把图像归一化为  $368 \times 368$ ，基于第一级信任映射的第二级感知野输出是  $31 \times 31$ ，这和原始图片的  $400 \times 400$  像素等价，其半径可以覆盖任何部件。当阶段增多，有效感知野就会变大。我们有 6 个阶段。

### 3) 用 CPM 学习

这个深度架构可以有許多层。训练这个网可能让梯度消失，就是反向传播在中间层会减弱。pm 级联预测框架有一个自然的解决这个问题方法。我们不断激励这个网络，通过在每个阶段  $t$  的输出定义一个损失函数，让预测的和实际信任映射的距离最小化。部件  $p$  理想的信任映射是  $bp$ ，通过把  $p$  部件的最可能点设定在 ground truth 位置。

### 69、Depthwise 卷积实际速度与理论速度差距较大，解释原因。

标签：深度学习

参考回答：

首先，caffe 原先的 gpu 实现 group convolution 很糟糕，用 for 循环每次算一个卷积，速度极慢。第二，cudnn7.0 及之后直接支持 group convolution，但本人实测，速度比 github 上几个直接写 cuda kernel 计算的 dw convolution 速度慢。例如对于  $n=128$ ,  $c=512$ ,  $h=32$ ,  $w=32$ ,  $group=512$  的卷积跑 100 次，cudnn 7.0 里的 group convolution 需要 4 秒多，而 DepthwiseConvolution 大概只需要 1 秒。

分析了一下 dw convolution 与普通 convolution 的理论计算复杂度，举例如下：

卷积 1：普通卷积，输入为  $64*64*256$ ，输出为  $64*64*256$ ，卷积核大小为  $3*3$ 。参数为  $3*3*256*256=590K$ ，计算量为  $64*64*256*3*3*256=2.42G$ ，计算过程的工作集内存总量（输入输出数据+参数）为  $64*64*256*2 + 3*3*256*256 = 2.69M$ 。

卷积 2：dw 卷积，输入为  $64*64*256$ ，输出为  $64*64*256$ ，卷积核大小为  $3*3$ 。参数为  $3*3*256=2.3K$  个，计算量为  $64*64*256*3*3=9.44M$ ，计算过程的工作集内存总量为  $64*64*256*2 + 3*3*256=2.10M$ 。

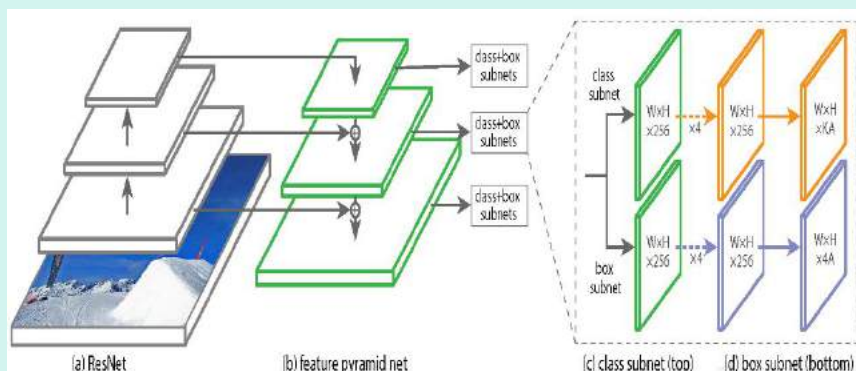
卷积 3：普通卷积，输入为  $64*64*16$ ，输出为  $64*64*16$ ，卷积核大小为  $3*3$ 。参数为  $3*3*16*16=2.3K$  个，计算量为  $64*64*16*3*3*16=9.44M$ ，计算过程的工作集内存总量为  $64*64*16*2 + 3*3*16*16=133K$ 。

可以看到卷积 2 肯定比卷积 1 快，因为计算量下降到  $1/256$  了，但卷积 2 实际上无法达到卷积 1 的 256 倍速度（我记得我测得结果大概是快 10 倍左右），因为工作集内存大小并没有显著降低。卷积 2 也无法达到卷积 3 的速度，因为虽然 FLOPS 相同，但工作集内存大小相差了很多倍，因此单位数据的计算密度小很多，很难充分利用 GPU 上的计算单元。

### 70、RetinaNet 的大致结构画一下

标签：深度学习

参考回答：





## 71、RetinaNet 为什么比 SSD 效果好

标签：深度学习

参考回答：

SSD 在训练期间重新采样目标类和背景类的比率，这样它就不会被图像背景淹没。RetinaNet 采用另一种方法来减少训练良好的类的损失。因此，只要该模型能够很好地检测背景，就可以减少其损失并重新增强对目标类的训练。所以 RetinaNet 比 SSD 效果好。

## 五、数据结构与算法

### 1、查找

#### 1、手写二分查找

考点：算法

详情：python

参考回答：

```
def binary_search(num_list, x):  
  
    num_list=sorted(num_list)  
  
    left, right = 0, len(num_list)  
  
    while left < right:  
  
        mid = (left + right) / 2  
  
        if num_list[mid] > x:  
  
            right = mid  
  
        elif num_list[mid] < x:  
  
            left = mid + 1  
  
        else:  
  
            return num_list[x]  
  
    return x
```

#### 2、算法题，单调函数求零点（简单的二分法）



参考回答：

```
#include <iostream>

#include <cstdio>

using namespace std;

double f(double x) {

    return x*x*x*x*x-15*x*x*x*x+85*x*x*x-225*x*x+274*x-121;

}

int main()

{

    double x=1.5,y=2.4; //边界条件

    while(y-x>=0.0000001) { //保留到6位小数 计算到7位

        double t=(x+y)/2; //二分法

        if(f(x)*f(t)<=0) y=t; //解 (x, t]

        else x=t; //解 (t, y)

    }

    printf("%.6f\n", x);

    return 0;

}
```

### 3、特别大的数据量，实现查找，排序

参考回答：

#### 1) 位图法

位图法是我在编程珠玑上看到的一种比较新颖的方法，思路比较巧妙效率也很高。

使用场景举例：对 2G 的数据量进行排序，这是基本要求。

数据：1、每个数据不大于 8 亿；2、数据类型位 int；3、每个数据最多重复一次。

内存：最多用 200M 的内存进行操作。



首先对占用的内存进行判断，每个数据不大于 8 亿，那么 8 亿是一个什么概念呢。

\*\*

1 byte = 8 bit (位)

1024 byte = 8\*1024 bit = 1k

1024 k = 8\*1024\*1024 bit = 1M = 8388608 bit

\*\*

也就是 1M=8388608 位

而位图法的基本思想就是利用一位代表一个数字，例如 3 位上为 1，则说明 3 在数据中出现过，若为 0，则说明 3 在数据中没有出现过。所以当题目中出现每个数据最多重复一次这个条件时，我们可以考虑使用位图法来进行大数据排序。

那么假如使用位图法来进行这题的排序，内存占用多少呢。由题目知道每个数据不大于 8 亿，那么我们就需要 8 亿位，占用  $800000000/8388608=95M$  的空间，满足最多使用 200M 内存进行操作的条件，这也是这题能够使用位图法来解决的一个基础。

## 2) 堆排序法

堆排序是 4 种平均时间复杂度为  $n\log n$  的排序方法之一，其优点在于当求 M 个数中的前 n 个最大数，和最小数的时候性能极好。所以当从海量数据中要找出前 m 个最大值或最小值，而对其他值没有要求时，使用堆排序法效果很好。

使用场景：从 1 亿个整数里找出 100 个最大的数

步骤：

(1) 读取前 100 个数字，建立最大值堆。(这里采用堆排序将空间复杂度讲得很低，要排序 1 亿个数，但一次性只需读取 100 个数字，或者设置其他基数，不需要 1 次性读完所有数据，降低对内存要求)

(2) 依次读取余下的数，与最大值堆作比较，维持最大值堆。可以每次读取的数量为一个磁盘页面，将每个页面的数据依次进堆比较，这样节省 IO 时间。

(3) 将堆进行排序，即可得到 100 个有序最大值。

堆排序是一种常见的算法，但了解其的使用场景能够帮助我们更好的理解它。

## 3) 较为通用的分治策略

分治策略师对常见复杂问题的一种万能的解决方法，虽然很多情况下，分治策略的解法都不是最优解，但是其通用性很强。分治法的核心就是将一个复杂的问题通过分解抽象成若干简单的问题。





应用场景：10G 的数据，在 2G 内存的单台机器上排序的算法

我的想法，这个场景既没有介绍数据是否有重复，也没有给出数据的范围，也不是求最大的个数。而通过分治虽然可能需要的 io 次数很多，但是对解决这个问题还是具有一定的可行性的。

步骤：

(1) 从大数据中抽取样本，将需要排序的数据切分为多个样本数大致相等的区间，例如：1-100, 101-300...

(2) 将大数据文件切分为多个小数据文件，这里要考虑 IO 次数和硬件资源问题，例如可将小数据文件数设定为 1G（要预留内存给执行时的程序使用）

(3) 使用最优的算法对小数据文件的数据进行排序，将排序结果按照步骤 1 划分的区间进行存储

(4) 对各个数据区间内的排序结果文件进行处理，最终每个区间得到一个排序结果的文件

(5) 将各个区间的排序结果合并。通过分治将大数据变成小数据进行处理，再合并。

## 2、哈希

### 1、Hash 表处理冲突的方法

考点：数据存储

详情：数据结构与算法

参考回答：

开放定址法

为产生冲突的地址  $H(\text{key})$  求得一个地址序列  $H_0, \dots, H_s (1 \leq s \leq m - 1)$ ，其中  $H_i = (H(\text{key}) + d_i) \text{MOD } m$ 。其中  $m$  为表的长度，而增量  $d_i$  有三种取值方法，线性探测再散列，平方探测再散列，随即探测再散列。

链地址法

将所有 Hash 地址相同的记录都链接在同一链表中

再 Hash 法

同时构造多个不同的 Hash 函数，当产生冲突时，计算另一个 Hash 函数地址直到不再发生冲突为止。



建立公共溢出区

将 Hash 表分为基本表和溢出表,若是与基本表发生冲突,都放入溢出表。

## 2、一致性哈希

参考回答：一致性哈希算法在 1997 年由麻省理工学院提出，设计目标是为了解决因特网中的热点(Hot pot)问题，初衷和 CARP（缓冲阵列路由协议，Cache Array Routing Protocol）十分类似。一致性哈希修正了 CARP 使用的简单哈希算法带来的问题，使得 DHT（Distributed Hash Table，分布式哈希）可以在 P2P 环境中真正得到应用。

标准：一致性哈希提出了在动态变化的 Cache 环境中，哈希算法应该满足的 4 个适应条件：

### 1）、平衡性(Balance)

平衡性也就是说负载均衡，是指客户端 hash 后的请求应该能够分散到不同的服务器上去。一致性 hash 可以做到每个服务器都进行处理请求，但是不能保证每个服务器处理的请求的数量大致相同。

### 2）、单调性(Monotonicity)

单调性是指如果已经有一些内容通过哈希分派到了相应的缓冲中，又有新的缓冲加入到系统中。哈希的结果应该能够保证已分配的内容可以被映射到新的缓冲中去，而不会被映射到旧的缓冲集合中的其他缓冲区。

### 3）、分散性(Spread)

分布式环境中，客户端请求时候可能不知道所有服务器的存在，可能只知道其中一部分服务器，在客户端看来他看到的部分服务器会形成一个完整的 hash 环。如果多个客户端都把部分服务器作为一个完整 hash 环，那么可能会导致，同一个用户的请求被路由到不同的服务器进行处理。这种情况显然是应该避免的，因为它不能保证同一个用户的请求落到同一个服务器。所谓分散性是指上述情况发生的严重程度。

### 4）、负载(Load)

负载问题实际上是从另一个角度看待分散性问题。既然不同的终端可能将相同的内容映射到不同的缓冲区中，那么对于一个特定的缓冲区而言，也可能被不同的用户映射为不同的内容。与分散性一样，这种情况也是应当避免的，因此好的哈希算法应能够尽量降低缓冲的负荷。

## 3、Hash 表处理冲突的方法

考点:数据存储

标签：数据结构与算法

参考回答：

开放定址法



为产生冲突的地址  $H(\text{key})$  求得一个地址序列  $H_0, \dots, H_s (1 \leq s \leq m - 1)$ , 其中  $H_i = (H(\text{key}) + d_i) \text{MOD } m$ 。其中  $m$  为表的长度, 而增量  $d_i$  有三种取值方法, 线性探测再散列, 平方探测再散列, 随即探测再散列。

链地址法

将所有 Hash 地址相同的记录都链接在同一链表中

再 Hash 法

同时构造多个不同的 Hash 函数, 当产生冲突时, 计算另一个 Hash 函数地址直到不再发生冲突为止。

建立公共溢出区

将 Hash 表分为基本表和溢出表, 若是与基本表发生冲突, 都放入溢出表。

#### 4、apriori

标签：数据结构与算法

参考回答：

##### 1) Apriori 原理

如果一个项集是频繁的, 则它的所有子集一定也是频繁的; 相反, 如果项集是非频繁的, 则它的所有超集也一定是非频繁的。

##### 2) 发现频繁项集

假定事务总数为  $N$ , 支持度阈值是  $\text{minsup}$ , 发现频繁项集的过程如下 (理论上, 存在许多产生候选项集的方法, 本例使用支持度阈值来产生):

①初始时每个项都被看作候选 1-项集。计数对它们的支持度之后, 将支持度少于阈值的候选项集丢弃, 生成频繁 1-项集。

②在第二次迭代, 依据 Apriori 原理 (即所有非频繁的 1-项集的超集都是非频繁的), 仅使用频繁 1-项集来产生候选 2-项集。此时生成的候选 2-项集有多个, 将支持度少于阈值的候选项集丢弃, 生成频繁 2-项集。

③经过多次迭代, 每次用上一次生成的频繁  $n$ -项集产生新的候选  $(n+1)$ -项集, 直至没有发现频繁  $(n+1)$ -项集, 则得到的频繁  $n$ -项集就是最终结果。

##### 3) 发现关联规则

发现关联规则是指找出支持度大于等于  $\text{minsup}$  并且置信度大于等于  $\text{minconf}$  的所有规则, 其中  $\text{minsup}$  和  $\text{minconf}$  是对应的支持度阈值和置信度阈值。



## 5、KM 算法

标签：数据结构与算法

参考回答：

匈牙利算法：求最大匹配，那么我们希望每一个在左边的点都尽量找到右边的一个点和它匹配。我们依次枚举左边的点  $x$  的所有出边指向的点  $y$ ，若  $y$  之前没有被匹配，那么  $(x, y)$  就是一对合法的匹配，我们将匹配数加一，否则我们试图给原来匹配  $y$  的  $x'$  重新找一个匹配，如果  $x'$  匹配成功，那么  $(x, y)$  就可以新增为一对合法的匹配。给  $x'$  寻找匹配的过程可以递归解决。

从一边的未饱和点出发，寻找增广路。复杂度： $O(VE)$

KM 算法：给定一个带权的二分图，求权值最大的完备匹配

相等子图的完备匹配=原图的最大权匹配

1) 初始化可行性顶标

2) 对  $n$  个点在相等子图中寻找增广路

(1) 初始化访问标记

(2) 寻找增广路

(3) 若增广路不存在，则修改交错路中的顶标，直到对某个点而言找到一条增广路为止

3) 求得最大权

算法时间复杂度  $O(n^3)O(n^3)$

## 3、表达式、字符串

### 1、中缀表达式转后缀表达式

考点：后缀表达式

详情：数据结构与算法

参考回答：

对于中缀表达式，遇到操作数直接将其输出，如果遇到操作符和左括号全部压入栈中，若遇到右括号则将栈中元素全部弹出，直到遇到左括号为止。压栈过程中，若遇到其它操作符，从栈中弹出元素直到遇到更低优先级的操作符为止。

### 2、算法题：翻转中间由各种符号隔开的字符串

参考回答：



```
import java.util.Scanner;

import java.util.StringTokenizer;

//完成字符串的反转，如输入 we; tonight; you;输出 ew; thginot; uoy;

public class FanZhuan {

    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);

        System.out.println("请输入字符串:");

        String str = sc.nextLine();

        //考虑字符串中的逗号

        StringTokenizer st = new StringTokenizer(str,";");

        //将 string 对象变成可改变的 StringBuffer 对象

        while(st.hasMoreTokens()){

            String streverse = new
StringBuffer(st.nextToken()).reverse().toString();

            System.out.print(streverse);

            System.out.print(";");

        }

    }

}
```

### 3、问题：A+B\* (C-D)/E 的后缀表达式。

知识点：后缀表达式/逆波兰表达式

详情：数据结构与算法

参考回答：ABCD-\*E/+

解析：

转换过程需要用到栈，具体过程如下：

1) 如果遇到操作数，我们就直接将其输出。



- 2) 如果遇到操作符，则我们将其放入到栈中，遇到左括号时我们也将之放入栈中。
- 3) 如果遇到一个右括号，则我们将栈元素弹出，将弹出的操作符输出直到遇到左括号为止。注意，左括号只弹出并不输出。
- 4) 如果遇到任何其他的操作符，如（“+”，“\*”，“（”）等，从栈中弹出元素直到遇到发现更低优先级的元素(或者栈为空)为止。弹出完这些元素后，才将遇到的操作符压入到栈中。有一点需要注意，只有在遇到“)”的情况下我们才弹出“(”，其他情况我们都不会弹出“(”。
- 5) 如果我们读到了输入的末尾，则将栈中所有元素依次弹出。

## 4、栈与堆

### 1、大顶堆怎么插入删除

考点:堆

参考回答:

插入:在一个大顶堆之后插入新的元素可能会破坏堆的结构,此时需要找到新插入节点的父节点,对堆进行自下而上的调整使其变成一个大顶堆。

删除:将堆的最后一个元素填充到删除元素的位置,然后调整堆结构构造出新的大顶堆

### 2、堆栈区别

参考回答:

堆和栈的区别:

一、堆栈空间分配区别:

1)、栈(操作系统):由操作系统自动分配释放,存放函数的参数值,局部变量的值等。其操作方式类似于数据结构中的栈;

2)、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。

二、堆栈缓存方式区别:

1)、栈使用的是一级缓存,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;





2)、堆是存放在二级缓存中，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：内存中，存储的是引用数据类型，引用数据类型无法确定大小，堆实际上是一个在内存中使用到内存中零散空间的链表结构的存储空间，堆的大小由引用类型的大小直接决定，引用类型的大小的变化直接影响到堆的变化

栈：是内存中存储值类型的，大小为 2M，超出则会报错，内存溢出

三、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。

特点：先进后出

### 3、栈溢出有哪些情况

参考回答：

- 1) 局部数组过大。当函数内部的数组过大时，有可能导致堆栈溢出。
- 2) 递归调用层次太多。递归函数在运行时会执行压栈操作，当压栈次数太多时，也会导致堆栈溢出。
- 3)、指针或数组越界。这种情况最常见，例如进行字符串拷贝，或处理用户输入等等。

## 5、树

### 1、问题： 手撕代码，根据前序，中序创建二叉树。

知识点：二叉树

详情：数据结构与算法

参考回答：

```
public class Solution {  
  
    public TreeNode func(int [] pre, int s1, int e1, int [] in, int s2, int e2)  
    {  
  
        if(s1>e1||s2>e2) return null;  
  
        TreeNode p = new TreeNode(pre[s1]);
```



```
        for(int i=s2;i<=e2;i++){

            if(pre[s1]==in[i]){

                p.left = func(pre, s1+1, s1+i-s2, in, s2, i-1);

                p.right = func(pre, s1+i-s2+1, e1, in, i+1, e2);

            }

        }

        return p;

    }

    public TreeNode reConstructBinaryTree(int [] pre,int [] in) {

        return func(pre, 0, pre.length-1, in, 0, in.length-1);

    }

}
```

解析：剑指 OFFER 原题：构建二叉树。通过前序第一个节点（根节点）分割后序遍历的字符串。分别递归的根节点的左右节点。

## 2、算法题：从右边看被遮挡的二叉树，求露出的 node

参考回答：

```
import java.util.Deque;

import java.util.LinkedList;

import java.util.List;
```

//树节点类

```
class TreeElement {

    int val;

    TreeElement left;

    TreeElement right;
```



```
TreeElement(int x) {  
  
    val = x;  
  
}  
  
}  
  
public class RightSideView {  
  
    public List<Integer> rightSideView(TreeElement root) {  
  
        List<Integer> result = new LinkedList<>();  
  
        if(root != null) {  
  
            Deque<TreeElement> deque = new LinkedList<>();  
  
            //当前层的节点数  
  
            int current = 1;  
  
            //下一层的节点数  
  
            int next = 0;  
  
            TreeElement node;  
  
            deque.addLast(root);  
  
            while(deque.size() > 0) {  
  
                //取第一个节点  
  
                node = deque.removeFirst();  
  
                current--;  
  
                //添加非空的节点  
  
                if(node.left != null) {  
  
                    next++;  
  
                    deque.addLast(node.left);  
  
                }  
  
            }  
  
        }  
  
    }  
  
}
```



```
        //添加非空的节点

        if(node.right != null) {

            next++;

            deque.addLast(node.right);

        }

        //如果当前层已经处理完了

        if(current == 0) {

            //保存此层的最右一个节点值

            result.add(node.val);

            //设置下一层的元素个数

            current = next;

            next = 0;

        }

    }

    return result;

}
```

### 3、算法题，给前序和中序，求出二叉树

参考回答：

```
class TreeNode {

    int val;

    TreeNode left;

    TreeNode right;

    TreeNode(int x) {
```



```
        val = x;

    }

}

public class TestRecoverBinaryTree {

    public TreeNode reConstructBinaryTree(int [] preOrder,int [] inOrder)

    {

        int pLen = preOrder.length;

        int iLen = inOrder.length;

        if(pLen==0 && iLen==0)

        {

            return null;

        }

        return btConstruct( preOrder, inOrder, 0, pLen-1,0, iLen-1);

    }

    //构建方法，pStart 和 pEnd 分别是前序遍历序列数组的第一个元素和最后一个元素：

    //iStart 和 iEnd 分别是中序遍历序列数组的第一个元素和最后一个元素。

    public TreeNode btConstruct(int[] preOrder, int[] inOrder, int pStart, int pEnd,int iStart,int iEnd)

    {

        //建立根节点

        TreeNode tree = new TreeNode(preOrder[pStart]);

        tree.left = null;

        tree.right = null;

        if(pStart == pEnd && iStart == iEnd)

        {
```



```
        return tree;

    }

    int root = 0;

    //找中序遍历中的根节点

    for(root=iStart; root<iEnd; root++)

    {

        if(preOrder[pStart] == inOrder[root])

        {

            break;

        }

    }

    //划分左右子树

    int leftLength = root - iStart;//左子树

    int rightLength = iEnd - root;//右子树

    //遍历左子树

    if(leftLength>0)

    {

        tree.left = btConstruct(preOrder, inOrder,  pStart+1,

pStart+leftLength, iStart, root-1);

    }

    //遍历右子树

    if(rightLength>0)

    {

        tree.right = btConstruct(preOrder, inOrder,  pStart+leftLength+1,

pEnd, root+1, iEnd);

    }

}
```





```
        return tree;

    }

}
```

#### 4、算法题，trim 二叉搜索树

参考回答：

```
class Solution {

public:

    TreeNode* trimBST(TreeNode* root, int L, int R) {

        if(root==NULL)

            return NULL;

        if(root->val<L)

            return trimBST(root->right,L,R);

        else if(root->val>R)

            return trimBST(root->left,L,R);

        else

        {

            root->left=trimBST(root->left,L,R);

            root->right=trimBST(root->right,L,R);

        }

        return root;

    }

};
```

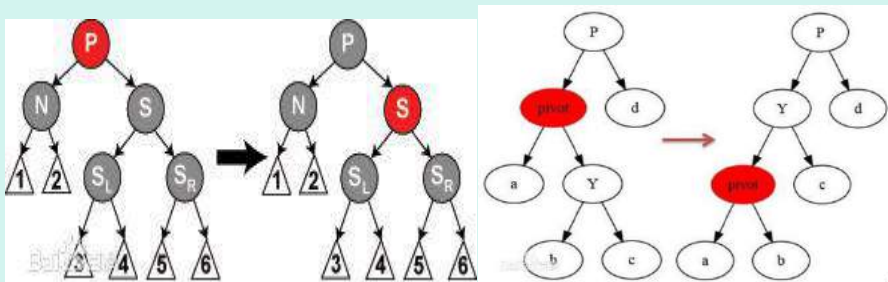
#### 5、红黑树

参考回答：

红黑树（Red Black Tree）是一种自平衡二叉查找树，是在计算机科学中用到的一种数据结构，典型的用途是实现关联数组。红黑树和 AVL 树类似，都是在进行插入和删除操作时通过特定操作保持二叉查找树的平衡，从而获得较高的查找性能。

它虽然是复杂的，但它的最坏情况运行时间也是非常良好的，并且在实践中是高效的：它可以在  $O(\log n)$  时间内做查找，插入和删除，这里的  $n$  是树中元素的数目。

它的统计性能要好于平衡二叉树，因此，红黑树在很多地方都有应用。在 C++ STL 中，很多部分(包括 set, multiset, map, multimap)应用了红黑树的变体(SGI STL 中的红黑树有一些变化，这些修改提供了更好的性能，以及对 set 操作的支持)。其他平衡树还有：AVL，SBT，伸展树，TREAP 等等。



红黑树是每个节点都带有颜色属性的二叉查找树，颜色或红色或黑色。在二叉查找树强制一般要求以外，对于任何有效的红黑树我们增加了如下的额外要求：

性质 1. 节点是红色或黑色。

性质 2. 根节点是黑色。

性质 3 每个叶节点（NIL 节点，空节点）是黑色的。

性质 4 每个红色节点的两个子节点都是黑色。（从每个叶子到根的所有路径上不能有两个连续的红色节点）

性质 5. 从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点。

这些约束强制了红黑树的关键性质：从根到叶子的最长的可能路径不多于最短的可能路径的两倍长。结果是这个树大致上是平衡的。因为操作比如插入、删除和查找某个值的最坏情况时间都要求与树的高度成比例，这个在高度上的理论上限允许红黑树在最坏情况下都是高效的，而不同于普通的二叉查找树。

要知道为什么这些特性确保了这个结果，注意到性质 4 导致了路径不能有两个毗连的红色节点就足够了。最短的可能路径都是黑色节点，最长的可能路径有交替的红色和黑色节点。因为根据性质 5 所有最长的路径都有相同数目的黑色节点，这就表明了没有路径能多于任何其他路径的两倍长。

在很多树数据结构的表示中，一个节点有可能只有一个子节点，而叶子节点不包含数据。用这种范例表示红黑树是可能的，但是这会改变一些属性并使算法复杂。为此，本文中我们



使用“nil 叶子”或“空(null)叶子”，如上图所示，它不包含数据而只充当树在此结束的指示。这些节点在绘图中经常被省略，导致了这些树好象同上述原则相矛盾，而实际上不是这样。与此有关的结论是所有节点都有两个子节点，尽管其中的一个或两个可能是空叶子。

## 6、排序

1、对一千万个整数排序, 整数范围在 $[-1000, 1000]$ 间, 用什么排序最快?

考点: 排序算法

详情: 数据结构与算法

参考回答:

在以上的情景下最好使用计数排序, 计数排序的基本思想为在排序前先统计这组数中其它数小于这个数的个数, 其时间复杂度为  $O(n + k)$ , 其中  $n$  为整数的个数,  $k$  为所有数的范围, 此场景下的  $n \gg k$ , 所以计数排序要比其他基于的比较排序效果要好。

2、堆排序的思想

考点: 排序算法

详情: 数据结构与算法

参考回答:

将待排序的序列构成一个大顶堆, 这个时候整个序列的最大值就是堆顶的根节点, 将它与末尾节点进行交换, 然后末尾变成了最大值, 然后剩余  $n-1$  个元素重新构成一个堆, 这样得到这  $n$  个元素的次大值, 反复进行以上操作便得到一个有序序列。

3、冒泡排序

考点: 排序算法

参考回答:

```
def bubble_sort(lst):  
  
    count=len(lst)  
  
    for i in range(0, count):  
  
        for j in range(i, count):  
  
            if lst[i]>lst[j]:
```



```
lst[i],lst[j]=lst[j],lst[i]
```

#### 4、快速排序的最优情况

考点:排序

参考回答:

快速排序的最优情况是 Partition 每次划分的都很均匀, 当排序的元素为  $n$  个, 则递归树的深度为  $\log n + 1$ 。在第一次做 Partition 的时候需对所有元素扫描一遍, 获得的枢纽元将所有元素一分为二, 不断的划分下去直到排序结束, 而在此情况下快速排序的最优时间复杂度为  $n \log n$ 。

#### 5、抽了两道面试题目两道。8 个球，1 个比较重，天平，几步找到重的？

参考回答：两次。分成 3-3-2。

详情：数据结构与算法

#### 6、求一些数里的第 $n$ 大的数

参考回答:

两种解法:

1. 快排的思想，满足前一部分为  $k-1$  个数，第  $k$  个数就是第  $k$  大的
2. 堆排序，维护一个  $n$  大小的小顶堆。比小顶堆的值大，顶部出堆，新值进堆。最终遍历后，堆顶部的就是第  $k$  大的数。

注意数组的长度和  $n$  的比较，即非法查询。

#### 7、算法题： topK 给出 3 种解法

参考回答:

1) 局部淘汰法 -- 借助“冒泡排序”获取 TopK

思路：（1）可以避免对所有数据进行排序，只排序部分；（2）冒泡排序是每一轮排序都会获得一个最大值，则  $K$  轮排序即可获得 TopK。

时间复杂度空间复杂度：（1）时间复杂度：排序一轮是  $O(N)$ ，则  $K$  次排序总时间复杂度为： $O(KN)$ 。（2）空间复杂度： $O(K)$ ，用来存放获得的 topK，也可以  $O(1)$  遍历原数组的最后  $K$  个元素即可。

2) 局部淘汰法 -- 借助数据结构“堆”获取 TopK



思路：（1）堆：分为大顶堆（堆顶元素大于其他所有元素）和小顶堆（堆顶其他元素小于所有其他元素）。（2）我们使用小顶堆来实现。（3）取出 K 个元素放在另外的数组中，对这 K 个元素进行建堆。（4）然后循环从 K 下标位置遍历数据，只要元素大于堆顶，我们就将堆顶赋值为该元素，然后重新调整为小顶堆。（5）循环完毕后，K 个元素的堆数组就是我们所需要的 TopK。

时间复杂度与空间复杂度：（1）时间复杂度：每次对 K 个元素进行建堆，时间复杂度为： $O(K\log K)$ ，加上  $N-K$  次的循环，则总时间复杂度为  $O((K+(N-K))\log K)$ ，即  $O(N\log K)$ ，其中 K 为想要获取的 TopK 的数量 N 为总数据量。（2）空间复杂度： $O(K)$ ，只需要新建一个 K 大小的数组用来存储 topK 即可

### 3) 分治法 —— 借助”快速排序“方法获取 TopK

思路：（1）比如有 10 亿的数据，找出 Top1000，我们先将 10 亿的数据分成 1000 份，每份 100 万条数据。（2）在每一份中找出对应的 Top 1000，整合到一个数组中，得到 100 万条数据，这样过滤掉了 999% 的数据。（3）使用快速排序对这 100 万条数据进行”一轮“排序，一轮排序之后指针的位置指向的数字假设为 S，会将数组分为两部分，一部分大于 S 记作  $S_i$ ，一部分小于 S 记作  $S_j$ 。（4）如果  $S_i$  元素个数大于 1000，我们对  $S_i$  数组再进行一轮排序，再次将  $S_i$  分成了  $S_i$  和  $S_j$ 。如果  $S_i$  的元素小于 1000，则我们需要在  $S_j$  中获取  $1000 - \text{count}(S_i)$  个元素的，也就是对  $S_j$  进行排序（5）如此递归下去即可获得 TopK。

时间复杂度与空间复杂度：（1）时间复杂度：一份获取前 TopK 的时间复杂度： $O((N/n)\log K)$ 。则所有份数为： $O(N\log K)$ ，但是分治法我们会使用多核多机的资源，比如我们有 S 个线程同时处理。则时间复杂度为： $O((N/S)\log K)$ 。之后进行快排序，一次的时间复杂度为： $O(N)$ ，假设排序了 M 次之后得到结果，则时间复杂度为： $O(MN)$ 。所以，总时间复杂度大约为  $O(MN + (N/S)\log K)$ 。（2）空间复杂度：需要每一份一个数组，则空间复杂度为  $O(N)$ 。

## 8、快排

参考回答：

通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

```
#include <iostream>

using namespace std;

void Qsort(int a[], int low, int high)

{

    if(low >= high)

    {
```



```
        return;

    }

    int first = low;

    int last = high;

    int key = a[first];/*用字表的第一个记录作为枢轴*/

    while(first < last)

    {

        while(first < last && a[last] >= key)

        {

            --last;

        }

        a[first] = a[last];/*将比第一个小的移到低端*/

        while(first < last && a[first] <= key)

        {

            ++first;

        }

        a[last] = a[first];

        /*将比第一个大的移到高端*/

    }

    a[first] = key;/*枢轴记录到位*/

    Qsort(a, low, first-1);

    Qsort(a, first+1, high);

}

int main()

{
```

```
int a[] = {57, 68, 59, 52, 72, 28, 96, 33, 24};

Qsort(a, 0, sizeof(a) / sizeof(a[0]) - 1);/*这里原文第三个参数要减 1 否则
内存越界*/

for(int i = 0; i < sizeof(a) / sizeof(a[0]); i++)

{

    cout << a[i] << " ";

}

return 0;

}
```

## 9、说一下小顶堆的调整过程

标签：数据结构与算法

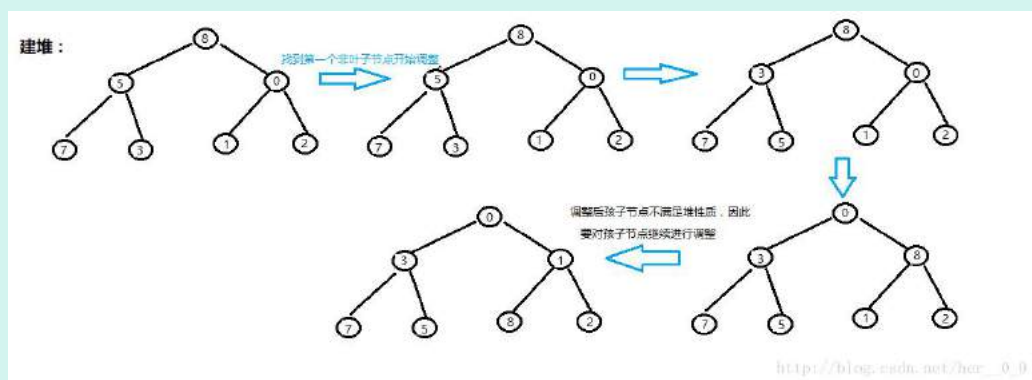
参考回答：

堆排序的步骤分为三步：

1) 建堆；2) 交换数据；3) 向下调整。

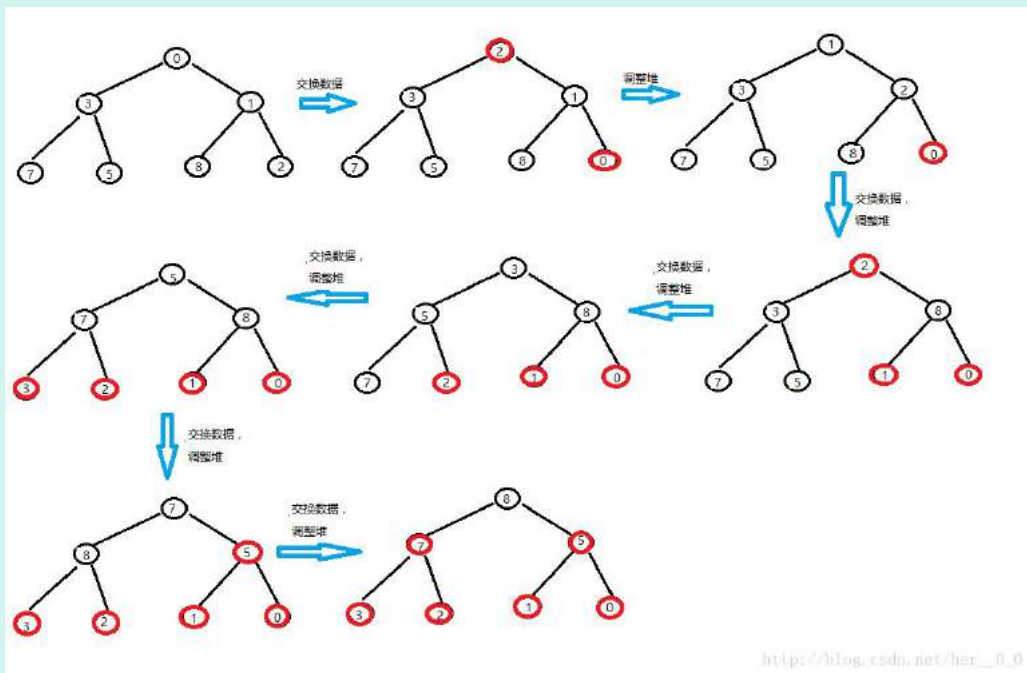
假设我们现在要对数组 `arr[] = {8, 5, 0, 3, 7, 1, 2}` 进行排序（降序）：

首先要先建小堆：



堆建好了下来就要开始排序了：





现在这个数组就已经是有序的了。

#### 10、算法题：2sum，3sum

参考回答：

2sum:

```
vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, vector<int>> mark;

    for(int i=0;i<nums.size();i++)
        mark[nums[i]].push_back(i);

    for(int i = 0;i<nums.size();i++){
        if(target-nums[i] == nums[i]){
            if(mark[nums[i]].size() > 1){
                vector<int> tmp{i,mark[nums[i]][1]};
                return tmp;
            }
        }else{

```



```
        if(mark.find(target-nums[i]) != mark.end()){

            vector<int> tmp{i,mark[target-nums[i]][0]};

            return tmp;

        }

    }

}

}

3sum:

void two_sum(vector<int>& nums, int i, int target, vector<vector<int>> &result) {

    int j = nums.size()-1;

    int b = i-1;

    while(i<j) {

        if(nums[i]+nums[j] == target) {

            result.push_back(vector<int>{nums[b],nums[i],nums[j]});

            i++;

            j--;

            while(i<j && nums[i] == nums[i-1]) i++;

            while(i<j && nums[j+1] == nums[j]) j--;

        } else {

            if(nums[i]+nums[j] < target)

                i++;

            else

                j--;

        }

    }

}
```



```
        return;

    }

    vector<vector<int>> threeSum(vector<int>& nums) {

        set<vector<int>> result;

        vector<vector<int>> result2;

        sort(nums.begin(), nums.end());

        for(int i=0;i<nums.size();i++)

            if(i>0&&nums[i-1]== nums[i])

                continue;

            else

                two_sum(nums, i+1, -nums[i], result2);

        return result2;

    }
```

## 7、高级算法

1、手撕代码：以概率  $p$  生成 1、概率  $1-p$  生成 0 的 rand 函数，得到 0-1 等概率的 rand 函数，计算新的 rand 函数中：调用一次，while 循环的期望次数

标签：数据结构与算法

参考回答：

```
function g(x)

{

    int v = f(x) + f(x)>0?0:1;

    if(v==0)

    {

        return 0; //1.f(x), f'(x)同时为 0

    }

}
```



```
else if(v==2)

{

    return 1;  //2. f(x), f'(x) 同时为 1

}

else

{

    g(x);  //3. f(x), f'(x) 一个为 0 一个为 1, 重新生成随机数

}

}
```

新的 rand 函数中：调用一次，while 循环的期望次数是 2。

## 2、Kruskal 算法的基本过程

考点：最小代价生成树

详情：数据结构与算法

参考回答：

Kruskal 算法是以边为主导地位，始终选取当前可用的拥有最小权值的边，所选择的边不能构成回路。首先构造一个只有  $n$  个顶点没有边的非连通图，给所有的边按值以从小到大的顺序排序，选择一个最小权值边，若该边的两个顶点在不同的连通分量上，加入到有效边中，否则舍去这条边，重新选择权值最小的边，以此重复下去，直到所有顶点在同一连通分量上。

## 3、BFS 和 DFS 的实现思想

考点：搜索算法

标签：数据结构与算法

参考回答：

BFS: (1) 顶点  $v$  入队列 (2) 当队列为非空时继续执行否则停止 (3) 从队列头取顶点  $v$ ，查找顶点  $v$  的所有子节点并依次从队列尾插入 (4) 跳到步骤 2

DFS: (1) 访问顶点  $v$  并打印节点 (2) 遍历  $v$  的子节点  $w$ ，若  $w$  存在递归的执行该节点。



#### 4、关联规则具体有哪两种算法, 它们之间的区别

参考回答:

Apriori 和 FP-growth 算法

Apriori 多次扫描交易数据库, 每次利用候选频繁集产生频繁集, 而 FP-growth 则利用树形结构, 无需产生候选频繁集而直接得到频繁集, 减少了扫描交易数据库的次数, 提高算法的效率但是 Apriori 有较好的扩展性可用于并行计算。一般使用 Apriori 算法进行关联分析, FP-growth 算法来高效发现频繁项集。

#### 5、贪婪算法

公司: 美团

标签: 数据结构与算法

参考回答:

贪婪算法(贪心算法)是指在对问题进行求解时, 在每一步选择中都采取最好或者最优(即最有利)的选择, 从而希望能够导致结果是最好或者最优的算法。贪婪算法所得到的结果往往不是最优的结果(有时候会是最优解), 但是都是相对近似(接近)最优解的结果。贪婪算法并没有固定的算法解决框架, 算法的关键是贪婪策略的选择, 根据不同的问题选择不同的策略。

必须注意的是策略的选择必须具备无后效性, 即某个状态的选择不会影响到之前的状态, 只与当前状态有关, 所以对采用的贪婪的策略一定要仔细分析其是否满足无后效性。

其基本的解题思路为: 1) 建立数学模型来描述问题; 2) 把求解的问题分成若干个子问题; 3) 对每一子问题求解, 得到子问题的局部最优解; 4) 把子问题对应的局部最优解合成原来整个问题的一个近似最优解。

#### 6、模拟退火, 蚁群对比

参考回答:

模拟退火算法: 退火是一个物理过程, 粒子可以从高能状态转换为低能状态, 而从低能转换为高能则有一定的随机性, 且温度越低越难从低能转换为高能。就是在物体冷却的过程中, 物体内部的粒子并不是同时冷却下来的。因为在外围粒子降温的时候, 里边的粒子还会将热量传给外围粒子, 就可能导致局部粒子温度上升, 当然, 最终物体的温度还是会下降到环境温度的。

先给出一种求最优解的方法, 在寻找一个最优解的过程中采取分段检索的方式, 在可行解中随机找出来一个, 然后在这个解附近进行检索, 找到更加优化的解, 放弃原来的, 在新得到的解附近继续检索, 当无法继续找到一个更优解的时候就认为算法收敛, 最终得到的是一个局部最优解。



蚁群算法:很显然,就是模仿蚂蚁寻找食物而发明的算法,主要用途就是寻找最佳路径。先讲一下蚂蚁是怎么寻找食物,并且在寻找到了食物后怎么逐渐确定最佳路径的。

事实上,蚂蚁的目光很短,它只会检索它附近的很小一部分空间,但是它在寻路过程中不会去重复它留下信息素的空间,也就是它不会往回走。当一群蚂蚁遇到障碍物了之后,它们会随机分为两拨,一波往左一波往右,但是因为环境会挥发掉它们的信息素,于是,较短的路留下的信息素多,而较长的路因为挥发较多,也就留下得少了。接下来,蚁群就会趋向于行走信息素较多的路径,于是大部分蚂蚁就走了较短的路了。但是蚁群中又有一些特别的蚂蚁,它们并不会走大家走的路,而是以一定概率寻找新路,如果新路更短,信息素挥发更少,渐渐得蚁群就会走向更短的路径。

以上就是蚂蚁寻路的具体过程。把这个过程映射到计算机算法上,计算机在单次迭代过程中,会在路径的节点上留下信息素(可以使用数据变量来表示)。每次迭代都做信息素蒸发处理,多次迭代后聚集信息素较多的路径即可认为是较优路径。

## 7、 算法题：名人问题，给出最优解法

参考回答：

问题描述：

有  $n$  个人他们之间认识与否用邻接矩阵表示(1 表示认识, 0 表示不认识), 并 A 认识 B 并不意味着 B 认识 A, 也就意味着是个有向图。如果一个人是名人, 他必须满足两个条件, 一个是他不认识任何人, 另一个是所有人必须都认识他。

解决问题：

用一个数组保存所有没检查的人的编号, 遍历数组中的两个人  $i, j$ 。如果  $i$  认识  $j$ , 则  $i$  必定不是名人, 删除  $i$ , 如果  $i$  不认识  $j$ , 则  $j$  必定不是名人, 删除  $j$ , 最终会只剩下一个人, 我们检查一下这个人是否是名人即可, 如果是, 返回这个人, 如果不是, 那么这  $n$  个人中并无名人。

代码：

//初始化数组编号

```
for(int i=a;i<n;i++){
```

```
    a[i]=i;
```

```
}
```

```
while(n>1){
```

```
    if(known[a[0]][a[1]]){
```

```
        //如果 a[0]号认识 a[1]号
```



```
//删除 a[0]，删除操作用 a[n] 替换掉 a[0] 即可，再将 n 下标减 1

a[0]=a[--n];

}else{

    //如果 a[0] 号不认识 a[1] 号

    //删除 a[1]

    a[1]=a[--n];

}

}

//最终检查 a[0] 是否为名人

for(int i=0;i<n;++i){

    //不考虑自身，并且检查他是否认识某个人，如果认识，那么不是名人

    //检查其他人是否认识他，如果有人不认识他，那么他也不是名人

    if((a[0]!=i)&&(known[a[0]][i]||!known[i][a[0]]))

        return -1;

}

return a[0];

}
```

算法优化：

以上算法需要用一个数组来保存没有检查的人的编号，意味着空间复杂度为  $O(n)$ ，是否可以让空间复杂度降低到  $O(1)$  呢，答案是肯定的，解决方法就是用一头扫的方法

这里我们就不需要用一个数组来保存没有检查的人的编号了，我们直接对  $n$  个人进行遍历

遍历的方式是定义两个下标，用两个下标一起往后扫，对于两个下标  $i, j$  而言，保证  $[0 \sim i-1]$  没有名人，并且  $[i \sim j-1]$  也没有名人，如果  $i$  认识  $j$ ，那么  $i$  不是名人，删掉  $i$ ，删除的方法就是  $i=j, j++$ ，如果  $i$  不认识  $j$ ，删除  $j$ ，删除的方式是  $j++$ ，遍历的时候让  $j$  一直加就可以了。

```
int i=0,j=1;

for(;j<n;j++){
```





```
        if (known[a[i]a[j]])

            i=j;

    }

    for (j=0; j<n; j++) {

        if ((i!=j)&&(known[i][j] || !known[j][i]))

            return -1;

    }

    return i;

}
```

算法优化 2:

除了一头扫的优化方式，也可以用两头扫的方式优化以上的算法，保证  $0 \sim i-1$  没有名人，并且  $j+1 \sim n$  也没有名人，如果  $i$  认识  $j$ ，删除  $i$ ，如果  $i$  不认识  $j$ ，删除  $j$

```
i=0; j=n-1;

while (i<j) {

    if (known[i][j]) {

        ++i;

    } else {

        --j;

    }

}

for (j=0; j<n; ++j) {

    if (i!=j&&(known[i][j] || !known[j][i]))

        return -1

}

return i;
```



```
}
```

#### 8、代码题：股票最大值。

标签：数据结构与算法

参考回答：

最大利润无外乎就是计算后面的数字减去前面的数字得到的一个最大的差值；

求总体的最大差值，需要的数据：当前的最小值，当前的最大差值；遍历求解即可。

```
class Solution {  
  
public:  
  
    int maxProfit(vector<int> prices)  
  
    {  
  
        int length = prices.size();  
  
        if (length < 2)  
  
            return 0;  
  
        int minPrice = prices[0];  
  
        int maxDiff = prices[1] - minPrice;  
  
        for (int i = 2; i < length; ++i)  
  
        {  
  
            if (prices[i - 1] < minPrice)  
  
                minPrice = prices[i - 1];  
  
            int currentDiff = prices[i] - minPrice;  
  
            if (currentDiff > maxDiff)  
  
                maxDiff = currentDiff;  
  
        }  
  
        return maxDiff;  
  
    }  
}
```

```
};
```

## 9、编辑距离

参考回答：

概念

编辑距离的作用主要是用来比较两个字符串的相似度的

编辑距离，又称 Levenshtein 距离（莱文斯坦距离也叫做 Edit Distance），是指两个字符串之间，由一个转成另一个所需的最少编辑操作次数，如果它们的距离越大，说明它们越是不同。许可的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。

在概念中，我们可以看出一些重点那就是，编辑操作只有三种。插入，删除，替换这三种操作，我们有两个字符串，将其中一个字符串经过上面的这三种操作之后，得到两个完全相同的字符串付出的代价是什么就是我们要讨论和计算的。

例如：

我们有两个字符串：kitten 和 sitting：

现在我们要将 kitten 转换成 sitting

我们可以做如下的一些操作：

kitten -> sitten 将 k 替换成 s    sitten -> sittin 将 e 替换成 i

sittin -> sitting 添加 g

在这里我们设置每经过一次编辑，也就是变化（插入，删除，替换）我们花费的代价都是 1。

例如：

如果 str1="ivan"，str2="ivan"，那么经过计算后等于 0。没有经过转换。相似度=1-0/Math.Max(str1.length, str2.length)=1

如果 str1="ivan1"，str2="ivan2"，那么经过计算后等于 1。str1 的 "1" 转换 "2"，转换了一个字符，所以距离是 1，相似度=1-1/Math.Max(str1.length, str2.length)=0.8

算法过程

1. str1 或 str2 的长度为 0 返回另一个字符串的长度。 if(str1.length==0) return str2.length; if(str2.length==0) return str1.length;

2. 初始化 (n+1)\*(m+1) 的矩阵 d，并让第一行和列的值从 0 开始增长。扫描两字符串 (n\*m 级的)，如果：str1[i] == str2[j]，用 temp 记录它，为 0。否则 temp 记为 1。然后在矩阵 d[i, j] 赋于 d[i-1, j]+1、d[i, j-1]+1、d[i-1, j-1]+temp 三者的最小值。

3. 扫描完后，返回矩阵的最后一个值  $d[n][m]$  即是它们的距离。

计算相似度公式：1-它们的距离/两个字符串长度的最大值。

我们用字符串 “ivan1” 和 “ivan2” 举例来看看矩阵中值的状况：

1、第一行和第一列的值从 0 开始增长

		i	v	a	n	1
	0	1	2	3	4	5
i	1					
v	2					
a	3					
n	4					
2	5					

图一

首先我们先创建一个矩阵，或者说是我们的二维数列，假设有两个字符串，我们的字符串的长度分别是  $m$  和  $n$ ，那么，我们矩阵的维度就应该是  $(m+1)*(n+1)$ 。

注意，我们先给数列的第一行第一列赋值，从 0 开始递增赋值。我们就得到了图一的这个样子

之后我们计算第一列，第二列，依次类推，算完整个矩阵。

我们的计算规则就是：

$d[i, j] = \min(d[i-1, j]+1, d[i, j-1]+1, d[i-1, j-1]+temp)$  这三个当中的最小值。

其中：  $str1[i] == str2[j]$ ，用  $temp$  记录它，为 0。否则  $temp$  记为 1

我们用  $d[i-1, j]+1$  表示增加操作

$d[i, j-1]+1$  表示我们的删除操作

$d[i-1, j-1]+temp$  表示我们的替换操作

2、举证元素的产生  $Matrix[i-1, j] + 1$  ;  $Matrix[i, j-1] + 1$  ;  $Matrix[i-1, j-1] + t$  三者当中的最小值



		i	v	a	n	1
	0+t=0	1+1=2	2	3	4	5
i	1+1=2	取三者最小值=0				
v	2	依次类推：1				
a	3	2				
n	4	3				
2	5	4				

3. 依次类推直到矩阵全部生成

		i	v	a	n	1
	0	1	2			
i	1	0	1			
v	2	1	0			
a	3	2	1			
n	4	3	2			
2	5	4	3			

		i	v	a	n	1
	0	1	2	3	4	5
i	1	0	1	2	3	4
v	2	1	0	1	2	3
a	3	2	1	0	1	2
n	4	3	2	1	0	1
2	5	4	3	2	1	1

这个就得到了我们的整个完整的矩阵。

## 8、链表

### 1、如何判断单链表是否是循环链表

参考回答：

时间复杂度：O(n)

空间复杂度：O(1)

两个指针，一个每次走一步，一个每次走两步，如果有环，两者会相遇。相遇后，让一个指针从头结点再次出发，两个指针每次都走一步，直到相遇点即为环入口。

```
public class Solution {
    public ListNode EntryNodeOfLoop(ListNode pHead) {
```



```
        ListNode left = pHead;

        ListNode right = pHead;

        while(right!=null && right.next!=null){

            left = left.next;

            right = right.next.next;

            if(left==right) break;

        }

        if(right==null || right.next == null){

            return null;

        }

        left = pHead;

        while(left!=right){

            right = right.next;

            left = left.next;

        }

        return left;

    }

}
```

## 2、反转链表

参考回答：

```
public class ListNode {

    public int data;

    public ListNode next;

}

public ListNode reverseList(ListNode pHead) {
```



```
ListNode pReversedHead = null; //反转过后的单链表存储头结点

ListNode pNode = pHead; //定义 pNode 指向 pHead;

ListNode pPrev = null; //定义存储前一个结点

while(pNode != null){

    ListNode pNext = pNode.next; //定义 pNext 指向 pNode 的下一个结点

    if(pNext==null){ //如果 pNode 的下一个结点为空，则 pNode 即为结果

        pReversedHead = pNode;

    }

    pNode.next = pPrev; //修改 pNode 的指针域指向 pPrev

    pPrev = pNode; //将 pNode 结点复制给 pPrev

    pNode = pNext; //将 pNode 的下一个结点复制给 pNode

}

return pReversedHead;

}
```

### 3、算法题，反转链表

参考回答：

```
public class ListNode {

    public int data;

    public ListNode next;

}

public ListNode reverseList(ListNode pHead) {

    ListNode pReversedHead = null; //反转过后的单链表存储头结点

    ListNode pNode = pHead; //定义 pNode 指向 pHead;

    ListNode pPrev = null; //定义存储前一个结点

    while(pNode != null) {
```





```
        ListNode pNext = pNode.next; //定义 pNext 指向 pNode 的下一个结点

        if(pNext==null){ //如果 pNode 的下一个结点为空，则 pNode 即为结果

            pReversedHead = pNode;

        }

        pNode.next = pPrev; //修改 pNode 的指针域指向 pPrev

        pPrev = pNode; //将 pNode 结点复制给 pPrev

        pNode = pNext; //将 pNode 的下一个结点复制给 pNode

    }

    return pReversedHead;

}
```

#### 4、算法题，单链表判断是否有环 (leetcode easy)，以及判断环入口

参考回答：

是否有环：

```
while (faster.next != null && faster.next.next != null) {

    faster = faster.next.next;

    slower = slower.next;

    if (faster == slower) {

        return true;

    }

}

return false;
```

判断环入口：

```
Node meet = null;

while (faster.next != null && faster.next.next != null) {

    faster = faster.next.next;
```



```
        slower = slower.next;

        if (faster == slower) {

            meet = faster;

        }

    }

    if (meet != null) {

        newSlower = head;

        while (newSlower != slower) {

            newSlower = newSlower.next;

            slower = slower.next;

        }

        return newSlower;

    }

    return null;
```

## 9、数组

### 1、找出数组中只出现 1 次的数，其余数均出现 2 次，扩展，其余数出现 2 次以上

参考回答：

位运算题目

位运算中异或的性质：两个相同数字异或=0，一个数和 0 异或还是它本身。当只有一个数出现一次时，我们把数组中所有的数，依次异或运算，最后剩下的就是落单的数，因为成对儿出现的都抵消了。

```
public static int find1From2(int[] a){

    int len = a.length, res = 0;

    for(int i = 0; i < len; i++){

        res = res ^ a[i];

    }
```



```
    }

    return res;
}

扩展：

public static int find1From3(int[] a){

    int[] bits = new int[32];

    int len = a.length;

    for(int i = 0; i < len; i++){

        for(int j = 0; j < 32; j++){

            bits[j] = bits[j] + ( (a[i]>>>j) & 1);

        }

    }

    int res = 0;

    for(int i = 0; i < 32; i++){

        if(bits[i] % 3 !=0){

            res = res | (1 << i);

        }

    }

    return res;

}
```

## 10、动态规划

1、最短描述数，10 的最短描述数是  $3^2+1^2$  所以是 2，求一个数的最短描述数

参考回答：

动态规划问题，参考牛客网“拼凑面额”题解

**2、跳台阶问题，每次只能跳 1 个台阶或者 2 个台阶，n 个台阶共有多少种方式**

参考回答：

最基础的动态规划问题

```
public class Solution {  
  
    public int JumpFloor(int target) {  
  
        if(target<=1) return target;  
  
        int[] a = new int[target+1];  
  
        a[1]=1; a[2]=2;  
  
        for(int i=3;i<=target;i++){  
  
            a[i] = a[i-1] + a[i-2];  
  
        }  
  
        return a[target];  
  
    }  
  
}
```

**3、动态规划和带记忆递归的区别**

考点:算法

参考回答：

自顶而下和自底而上

**4、手撕代码：0-1 矩阵的最大正方形**

参考回答：

```
public int maxSquare(int[][] matrix) {  
  
    int row = matrix.length; //行大小  
  
    int line = matrix[0].length; //列大小  
  
    //一个与 matrix 相同大小的辅助数组  
  
    int[][] tmp = new int[row][line];
```



```
//将 matrix 的第一行和第一列元素直接存放到

for(int i=0;i<row;i++){

    tmp[i][0] = matrix[i][0];

}

for(int i=0;i<line;i++){

    tmp[0][i] = matrix[0][i];

}

for(int i=1;i<row;i++){

    for(int j=1;j<line;j++){

        if(matrix[i][j] == 1){

            tmp[i][j] =

                Math.min(Math.min(tmp[i-1][j], tmp[i][j-1]), tmp[i-1][j-1]) +

1;

        }

        if(matrix[i][j] == 0){

            tmp[i][j] = 0;

        }

    }

}

int max=0; //记录 tmp 中最大元素的值（tmp 中元素值表示正方形的边长）

for(int i=0;i<row;i++){

    for(int j=0;j<line;j++){

        if(tmp[i][j] > max){

            max = tmp[i][j];

        }

    }

}
```

```
    }  
  
    return max*max;  
  
}
```

## 11、遍历

### 1、代码题：股票最大值。

标签：遍历

参考回答：最大利润无外乎就是计算后面的数字减去前面的数字得到的一个最大的差值；求总体的最大差值，需要的数据：当前的最小值，当前的最大差值；遍历求解即可。

```
class Solution {  
  
public:  
  
    int maxProfit(vector<int> prices)  
  
    {  
  
        int length = prices.size();  
  
        if (length < 2)  
  
            return 0;  
  
        int minPrice = prices[0];  
  
        int maxDiff = prices[1] - minPrice;  
  
        for (int i = 2; i < length; ++i)  
  
        {  
  
            if (prices[i - 1] < minPrice)  
  
                minPrice = prices[i - 1];  
  
            int currentDiff = prices[i] - minPrice;  
  
            if (currentDiff > maxDiff)  
  
                maxDiff = currentDiff;  
  
        }  
  
    }  
  
};
```



```
    }  
  
    return maxDiff;  
  
}  
  
};
```

## 六、编程语言，工具和环境

### 1、编程语言

#### 1、什么是 python 的生成器？

考点:python 基础

参考回答:

python 生成器是一个返回可以迭代对象的函数,可以被用作控制循环的迭代行为。生成器类似于返回值为数组的一个函数,这个函数可以接受参数,可以被调用,一般的函数会返回包括所有数值的数组,生成器一次只能返回一个值,这样消耗的内存将会大大减小。

#### 2、Java 抽象类和接口的区别？

考点:Java 基础

参考回答:

Java 中,一个类可以实现多个接口,但是一个类只能继承一个抽象类。接口中只包含接口签名,而抽象类可以提高默认实现,子类可以重载抽象类方法。接口中默认的所有方法均为 public 修饰,而抽象类可以用 protected 修饰符。接口中不允许定义任何属性,而抽象类中可以定义属性和常量在应用场景中。在应用场景中使用接口来完成同一方法的不同实现,若不同的方法实现需要共享同样的行为或状态则用抽象类。

#### 3、python 中 is 和 == 的区别

考点:python 基础

参考回答:

is 是用来判断两个变量引用的对象是否为同一个,==用于判断引用对象的值是否相等。可以通过 id() 函数查看引用对象的地址。





#### 4、python 方法解析顺序

考点:python 基础

参考回答:

Python 的方法解析顺序优先级从高到低为:实例本身→类→继承类(继承关系越近,越先定义,优先级越高)

#### 5、strcpy 函数

考点:C 语言基础

详情: c 语言

#### 6、Ctrl+C 程序挂掉还是抛出异常, 如何判断两个 dict 是否一样, list 头上删除元素, 字符串拼接?

考点:python 基础

参考回答:

Ctrl+C 会挂掉程序

通过 is 可以判断两个 dict 是否相同

list.pop(0) 删除 list 第一个元素

join() 函数进行字符串拼接

#### 7、pytorch 中 cuda() 作用, 两个 Tensor, 一个加了 cuda(), 一个没加, 相加后很怎样?

考点:GPU 基础

参考回答:

cuda() 将操作对象放在 GPU 内存中, 加了 cuda() 的 Tensor 放在 GPU 内存中而没加的 Tensor 放在 CPU 内存中, 所以这两个 Tensor 相加会报错。

#### 8、python 中 dict 和 list 的区别, dict 的内部实现

考点:python 基础



参考回答：

dict 查找速度快, 占用的内存较大, list 查找速度慢, 占用内存较小, dict 不能用来存储有序集合。Dict 用 {} 表示, list 用 [] 表示。

dict 是通过 hash 表实现的, dict 为一个数组, 数组的索引键是通过 hash 函数处理后得到的, hash 函数的目的是使键值均匀的分布在数组中。

## 9、C++析构函数

参考回答：

是执行与构造函数相反的操作：释放对象使用的资源，并销毁非 static 成员。

1. 函数名是在类名前加上~，无参数且无返回值。
2. 一个类只能有且有一个析构函数，如果没有显式的定义，系统会生成一个缺省的析构函数（合成析构函数）。
3. 析构函数不能重载。每有一次构造函数的调用就会有一次析构函数的调用。

## 10、C++的 delete, delete[]的区别

参考回答：

略

## 11、C++相关的问题虚函数

参考回答：

拥有 Virtual 关键字的函数称之为虚函数. 要成为虚函数必须满足两点，一就是这个函数依赖于对象调用，因为虚函数就是依赖于对象调用，因为虚函数是存在于虚函数表中，有一个虚函数指针指向这个虚表，所以要调用虚函数，必须通过虚函数指针，而虚函数指针是存在于对象中的。二就是这个函数必须可以取地址，因为我们的虚函数表中存放的是虚函数函数入口地址，如果函数不能寻址，就不能成为虚函数。

## 12、如何写多线程的代码

参考回答：

略

## 13、是否关注过 caffe 和 pytorch 是怎么写的吗？pytorch 调用多 GPU 函数内核



参考回答：

略

#### 14、Java 虚拟机内存的划分

考点:Java 基础

参考回答：

程序计数器,本地方法栈,虚拟机栈,堆,方法区和运行时的常量池。

#### 15、python dict 按照 value 进行排序

考点:python 字典基础

参考回答：

按照 value 从大到小排序：

```
sorted(d.items(),key = lambda x:x[1],reverse = True)
```

按照 value 从小到大排序：

```
sorted(d.items(),key = lambda x:x[1],reverse =False)
```

#### 16、C++中 static 关键字的作用

考点:C++基础

参考回答：

同时编译多个文件时,所有未加 static 前缀的全局变量和函数都具有全局可见性,所以加了 static 关键字的变量和函数可对其它源文件隐藏。还可以保持变量内容的持久性。用 static 前缀作为关键字的变量默认的初始值为 0。

#### 17、虚函数和纯虚函数的区别

考点:C++基础

参考回答：

含有纯虚函数的类称为抽象类,只含有虚函数的类不能称为抽象类。虚函数可以直接被使用,也可以被子类重载以后以多态形式调用,而纯虚函数必须在子类中实现该函数才可使



用,因为纯虚函数在基类中只有声明而没有定义。虚函数必须实现,对虚函数来说父类和子类都有各自的版本。

## 18、Python 多进程

参考回答:

方式一: `os.fork()`

方式二: 使用 `multiprocessing` 模块: 创建 `Process` 的实例, 传入任务执行函数作为参数

方式三: 使用 `multiprocessing` 模块: 派生 `Process` 的子类, 重写 `run` 方法

方式四: 使用进程池 `Pool`

## 19、深拷贝, 浅拷贝, 写一个出来(写了个自己认为对的版本)

参考回答:

深拷贝与浅拷贝的区别

深复制和浅复制最根本的区别在于是否是真正获取了一个对象的复制实体, 而不是引用。

浅复制 —— 只是拷贝了基本类型的数据, 而引用类型数据, 复制后也是会发生引用, 我们把这种拷贝叫做“(浅复制)浅拷贝”, 换句话说, 浅复制仅仅是指向被复制的内存地址, 如果原地址中对象被改变了, 那么浅复制出来的对象也会相应改变。

深复制 —— 在计算机中开辟了一块新的内存地址用于存放复制的对象。

浅拷贝实例

//此递归方法不包含数组对象

```
var obj = { a:1, arr: [2,3] };
```

```
var shallowObj = shallowCopy(obj);
```

```
function shallowCopy(src) {
```

```
    var newObj = {};
```

```
    for (var prop in src) {
```

```
        if (src.hasOwnProperty(prop)) {
```



```
        newObj[prop] = src[prop];

    }

}

return newObj;

}
```

#### 深拷贝实例

而深复制则不同，它不仅将原对象的各个属性逐个复制出去，而且将原对象各个属性所包含的对象也依次采用深复制的方法递归复制到新对象上。这就不会存在上面 obj 和 shallowObj 的 arr 属性指向同一个对象的问题。

```
var obj = {

    a:1,

    arr: [1,2],

    nation : '中国',

    birthplaces:['北京','上海','广州']

};

var obj2 = {name:'杨'};

obj2 = deepCopy(obj,obj2);

console.log(obj2);

//深复制，要想达到深复制就需要用递归

function deepCopy(o,c){

    var c = c || {};

    for(var i in o){

        if(typeof o[i] === 'object'){

            //要考虑深复制问题了

            if(o[i].constructor === Array){

                //这是数组
```



```
        c[i] = [];\n\n        }else{\n\n            //这是对象\n\n            c[i] = {};\n\n        }\n\n        deepCopy(o[i],c[i]);\n\n    }else{\n\n        c[i] = o[i];\n\n    }\n\n}\n\nreturn c\n\n}
```

## 20、在程序里面智能指针的名字是啥？

参考回答：

一个是 shared\_ptr 允许多个指针指向同一个对象；一个是 unique\_ptr 独占所指向的对象；还有一种伴随类 weak\_ptr 他是一种弱引用 指向 shared\_ptr 所指向的对象。

## 21、new，malloc 区别

参考回答：

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 malloc/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



## 22、纯虚函数怎么定义，写一个出来

参考回答：

纯虚函数是一种特殊的虚函数，在许多情况下，在基类中不能对虚函数给出有意义的实现，而把它声明为纯虚函数，它的实现留给该基类的派生类去做。这就是纯虚函数的作用。

纯虚函数是一种特殊的虚函数，它的一般格式如下：

```
class <类名>

{

virtual <类型><函数名>(<参数表>)=0;

...

};
```

## 23、函数后面接 const 是什么意思？

参考回答：

这是把整个函数修饰为 const，意思是“函数体内不能对成员数据做任何改动”。如果你声明这个类的一个 const 实例，那么它就只能调用有 const 修饰的函数。

## 24、写一个函数指针

参考回答：

```
#include<iostream>

using namespace std;

int compute(int a, int b, int(*func)(int, int)) { //函数指针

    return func(a, b);

}

int max(int a, int b) {

    return (a > b) ? a : b;

}

int min(int a, int b) {

    return(a < b) ? a : b;
```





```
    }

    int add(int a, int b) {

        return a + b;

    }

    int main() {

        int m, n, res_max, res_min, res_add;

        cin >> m >> n;

        res_max = compute(m, n, max);

        res_min = compute(m, n, &min);

        res_add = compute(m, n, add);

        cout << res_max << res_min << res_add << endl;

        return 0;

    }
```

25、抽象类和接口的区别，慢慢说

参考回答：



类型	abstract class	Interface
定义	abstract class 关键字	Interface 关键字
继承	抽象类可以继承一个类和实现多个接口；子类只可以继承一个抽象类	接口只可以继承接口（一个或多个）；子类可以实现多个接口
访问修饰符	抽象方法可以有 <b>public</b> 、 <b>protected</b> 和 <b>default</b> 这些修饰符	接口方法默认修饰符是 <b>public</b> 。不可以使用其它修饰符
方法实现	可定义构造方法，可以有抽象方法和具体方法	接口完全是抽象的，没构造方法，且方法都是抽象的，不存在方法的实现
实现方式	子类使用 <b>extends</b> 关键字来继承抽象类。如果子类不是抽象类的话，它需要提供抽象类中所有声明的方法的实现	子类使用关键字 <b>implements</b> 来实现接口。它需要提供接口中所有声明的方法的实现
作用	了把相同的东西提取出来, <b>即重用</b>	为了把程序模块进行固化的契约, 是 <b>为了降低耦合</b>

26、有看过 c++ 的一些库吗？

参考回答：

标准库

C++标准库，包括了 STL 容器，算法和函数等。

C++ Standard Library：是一系列类和函数的集合，使用核心语言编写，也是 C++ISO 自身标准的一部分。

Standard Template Library：标准模板库

C POSIX library： POSIX 系统的 C 标准库规范

ISO C++ Standards Committee：C++标准委员会

框架

C++通用框架和库

Apache C++ Standard Library：是一系列算法，容器，迭代器和其他基本组件的集合

ASL：Adobe 源代码库提供了同行的评审和可移植的 C++源代码库。

Boost：大量通用 C++库的集合。

BDE：来自于彭博资讯实验室的开发环境。



Cinder：提供专业品质创造性编码的开源开发社区。

Cxxomfort：轻量级的，只包含头文件的库，将 C++ 11 的一些新特性移植到 C++03 中。

Dlib：使用契约式编程和现代 C++ 科技设计的通用的跨平台的 C++ 库。

EASTL：EA-STL 公共部分

ffead-cpp：企业应用程序开发框架

Folly：由 Facebook 开发和使用的开源 C++ 库

JUCE：包罗万象的 C++ 类库，用于开发跨平台软件

libPhenom：用于构建高性能和高度可扩展性系统的事件框架。

LibSourcey：用于实时的视频流和高性能网络应用程序的 C++11 evented IO

LibU：C 语言写的多平台工具库

Loki：C++ 库的设计，包括常见的设计模式和习语的实现。

MiLi：只含头文件的小型 C++ 库

openFrameworks：开发 C++ 工具包，用于创意性编码。

Qt：跨平台的应用程序和用户界面框架

Reason：跨平台的框架，使开发者能够更容易地使用 Java，.Net 和 Python，同时也满足了他们对 C++ 性能和优势的需求。

ROOT：具备所有功能的一系列面向对象的框架，能够非常高效地处理和分析大量的数据，为欧洲原子能研究机构所用。

STLport：是 STL 具有代表性的版本

STXXL：用于额外的大型数据集的标准模板库。

Ultimate++：C++ 跨平台快速应用程序开发框架

Windows Template Library：用于开发 Windows 应用程序和 UI 组件的 C++ 库

Yomml1：C++11 的开放 multi-methods.

## 27、c++ 你看的最久的一章是哪一章，c++ primer 最熟哪一章

参考回答：

指针那一章看的最久，数组那一章最熟。



## 28、开发环境、语言的掌握

参考回答：

熟悉 Linux 常用命令，熟练使用 SVN、FTP，熟练使用 Keras 及常见 IDE；熟练使用 Java，Python，了解 C，C++，Matlab

## 29、Python 多进程

参考回答：

方式一：os.fork()

方式二：使用 multiprocessing 模块：创建 Process 的实例，传入任务执行函数作为参数

方式三：使用 multiprocessing 模块：派生 Process 的子类，重写 run 方法

方式四：使用进程池 Pool

## 30、Python 锁

参考回答：

Python 中的各种锁：

### 一、全局解释器锁（GIL）

#### 1、什么是全局解释器锁

每个 CPU 在同一时间只能执行一个线程，那么其他的线程就必须等待该线程的全局解释器，使用权消失后才能使用全局解释器，即使多个线程直接不会相互影响在同一个进程下也只有一个线程使用 cpu，这样的机制称为全局解释器锁(GIL)。GIL 的设计简化了 CPython 的实现，使的对象模型包括关键的内置类型，如：字典等，都是隐含的，可以并发访问的，锁住全局解释器使得比较容易的实现多线程的支持，但也损失了多处理器主机的并行计算能力。

#### 2、全局解释器锁的好处

1)、避免了大量的加锁解锁的好处

2)、使数据更加安全，解决多线程间的数据完整性和状态同步

#### 3、全局解释器的缺点

多核处理器退化成单核处理器，只能并发不能并行。

#### 4、GIL 的作用：



多线程情况下必须存在资源的竞争，GIL 是为了保证在解释器级别的线程唯一使用共享资源（cpu）。

## 二、同步锁

### 1、什么是同步锁？

同一时刻的一个进程下的一个线程只能使用一个 cpu，要确保这个线程下的程序在一段时间内被 cpu 执，那么就要用到同步锁。

### 2、为什么用同步锁？

因为有可能当一个线程在使用 cpu 时，该线程下的程序可能会遇到 io 操作，那么 cpu 就会切到别的线程上去，这样就有可能影响到该程序结果的完整性。

### 3、怎么使用同步锁？

只需要在对公共数据的操作前后加上上锁和释放锁的操作即可。

### 4、同步锁的所用：

为了保证解释器级别下的自己编写的程序唯一使用共享资源产生了同步锁。

## 三、死锁

### 1、什么是死锁？

指两个或两个以上的线程或进程在执行程序的过程中，因争夺资源或者程序推进顺序不当而相互等待的一个现象。

### 2、死锁产生的必要条件？

互斥条件、请求和保持条件、不剥夺条件、环路等待条件

### 3、处理死锁的基本方法？

预防死锁、避免死锁（银行家算法）、检测死锁（资源分配）、解除死锁：剥夺资源、撤销进程

## 四、什么是递归锁？

在 Python 中为了支持同一个线程中多次请求同一资源，Python 提供了可重入锁。这个 RLock 内部维护着一个 Lock 和一个 counter 变量，counter 记录了 acquire 的次数，从而使得资源可以被多次 require。直到一个线程所有的 acquire 都被 release，其他的线程才能获得资源。递归锁分为可递归锁与非递归锁。

## 五、什么是乐观锁？

假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。



六、什么是悲观锁？

假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作。

七、python 常用的加锁方式？

互斥锁、可重入锁、迭代死锁、互相调用死锁、自旋锁。

## 2、大数据相关

### 1、Spark 性能如何调优

考点:大数据基础

详情: Spark

参考回答:

避免创建重复的 RDD, 尽量复用同一 RDD, 尽量避免使用 shuffle 类算子, 优化数据结构, 使用 Hive ETL 预处理数据, 过滤少数导致倾斜的 key, 提高 shuffle 操作的并行度, 两阶段聚合, 将 reduce join 转为 map join。

### 2、map reduce 实现笛卡尔乘积

考点:大数据

参考回答:

在 Map 阶段, 将来自矩阵 A 的元素  $a_{ij}$  标识成 1 条  $\langle \text{key}, \text{value} \rangle$  的形式,  $\text{key}=(i, k), k=1, 2, \dots, l, \text{value}=(j, a_{ij})$ 。将来自矩阵 B 的元素  $b_{jk}$  标识成 1 条  $\langle \text{key}, \text{value} \rangle$  的形式,  $\text{key}=(i, k), k=1, 2, \dots, m, \text{value}=(j, b_{jk})$ 。

Shuffle 阶段将 key 相同的 value 放在相同的列表中

在 reduce 的时候将 key 相同来自不同矩阵的 value 做笛卡尔积

再将结果 map 成  $((i, j), \text{value})$  的形式

Shuffle 将 key 相同的 value 放在同一列表中

Reduce 时 key 相同的 value 求和

### 3、是否写过 udf, 问 udaf, udtf 区别和一些细节



参考回答：

udf:user defined function

特点：input:output=1:1

实例函数：md5, split, ltrim

应用场景：1:1 的情况，比如 md5...

实现方法：简单 udf 实现

extends UDF

方法名 evaluate

udaf:user defined aggregation function

特点：input:output=n:1

示例函数：sum, count, max, min.....

实现方法：extends UDAF, 内部静态类实现接口 UDAFEvaluator

五大方法：init:初始化 map 或是 reduce 需用到的变量

iterate:迭代处理每条数据，true

terminatePartial:相当于 mr 的 combiner

merge:其输入一定是 terminatePartial 的输出

terminate:处理的是 merge 的结果

udtf:user defined table function

特点:input:output=1:n

示例函数：explode

实现方法：udf+explode



## 七、自然语言处理

### 1、Word2vec

#### 1、Word2Vec 中 skip-gram 是什么, Negative Sampling 怎么做

考点: 词向量基础

详情: Word2Vec

参考回答:

Word2Vec 通过学习文本然后用词向量的方式表征词的语义信息, 然后使得语义相似的单词在嵌入式空间中的距离很近。而在 Word2Vec 模型中有 Skip-Gram 和 CBOW 两种模式, Skip-Gram 是给定输入单词来预测上下文, 而 CBOW 与之相反, 是给定上下文来预测输入单词。Negative Sampling 是对于给定的词, 并生成其负采样词集合的一种策略, 已知有一个词, 这个词可以看做一个正例, 而它的上下文词集可以看做是负例, 但是负例的样本太多, 而在语料库中, 各个词出现的频率是不一样的, 所以在采样时可以要求高频词选中的概率较大, 低频词选中的概率较小, 这样就转化为一个带权采样问题, 大幅度提高了模型的性能。

#### 2、FastText 和 Glovec 原理

考点: 词向量

详情: Word2Vec

参考回答:

FastText 是将句子中的每个词通过一个 lookup 层映射成词向量, 对词向量叠加取平均作为句子的向量, 然后直接用线性分类器进行分类, FastText 中没有非线性的隐藏层, 结构相对简单而且模型训练的更快。

Glovec 融合了矩阵分解和全局统计信息的优势, 统计语料库的词-词之间的共现矩阵, 加快模型的训练速度而且又可以控制词的相对权重。

#### 3、word2vec 实施过程

参考回答:

词向量其实是将词映射到一个语义空间, 得到的向量。而 word2vec 是借用神经网络的方式实现的, 考虑文本的上下文关系, 有两种模型 CBOW 和 Skip-gram, 这两种模型在训练的过程中类似。Skip-gram 模型是用一个词语作为输入, 来预测它周围的上下文, CBOW 模型是拿一个词语的上下文作为输入, 来预测这个词语本身。

词向量训练的预处理步骤:

1. 对输入的文本生成一个词汇表，每个词统计词频，按照词频从高到低排序，取最频繁的  $V$  个词，构成一个词汇表。每个词存在一个 one-hot 向量，向量的维度是  $V$ ，如果该词在词汇表中出现过，则向量中词汇表中对应的位置为 1，其他位置全为 0。如果词汇表中不出现，则向量为全 0

2. 将输入文本的每个词都生成一个 one-hot 向量，此处注意保留每个词的原始位置，因为是上下文相关的

3. 确定词向量的维数  $N$

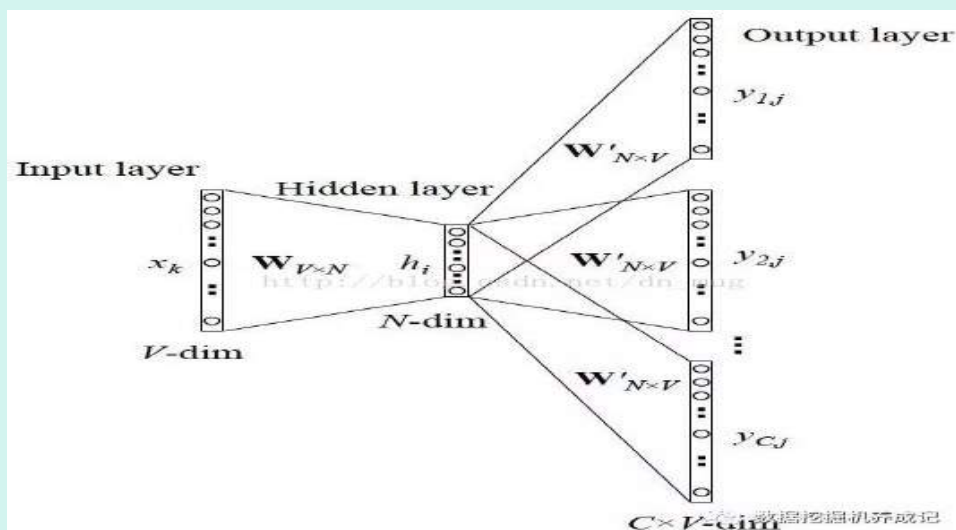
Skip-gram 处理步骤：

1. 确定窗口大小  $window$ ，对每个词生成  $2*window$  个训练样本， $(i, i-window)$ ， $(i, i-window+1)$ ， $\dots$ ， $(i, i+window-1)$ ， $(i, i+window)$

2. 确定  $batch\_size$ ，注意  $batch\_size$  的大小必须是  $2*window$  的整数倍，这确保每个 batch 包含了一个词汇对应的所有样本

3. 训练算法有两种：层次 Softmax 和 Negative Sampling

4. 神经网络迭代训练一定次数，得到输入层到隐藏层的参数矩阵，矩阵中每一行的转置即是对应词的词向量



CBOW 的处理步骤：

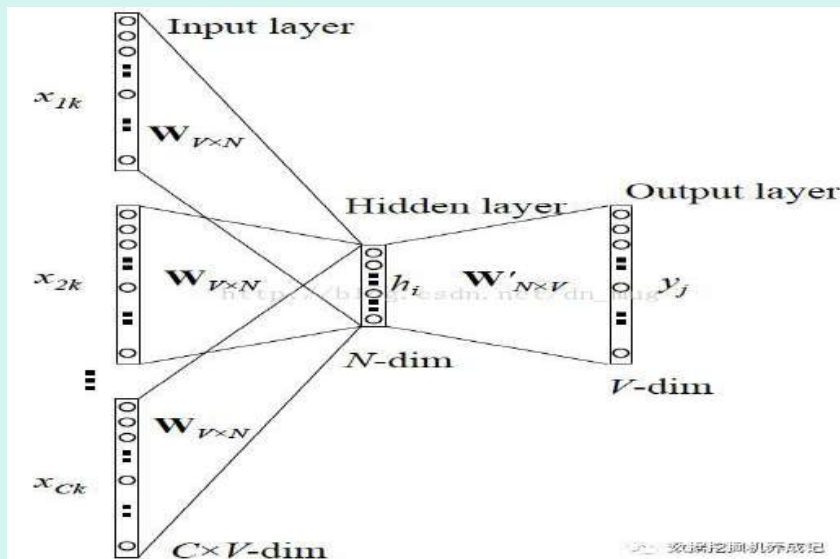
1. 确定窗口大小  $window$ ，对每个词生成  $2*window$  个训练样本， $(i-window, i)$ ， $(i-window+1, i)$ ， $\dots$ ， $(i+window-1, i)$ ， $(i+window, i)$

2. 确定  $batch\_size$ ，注意  $batch\_size$  的大小必须是  $2*window$  的整数倍，这确保每个 batch 包含了一个词汇对应的所有样本

3. 训练算法有两种：层次 Softmax 和 Negative Sampling



4. 神经网络迭代训练一定次数，得到输入层到隐藏层的参数矩阵，矩阵中每一行的转置即是对应词的词向量



参数矩阵解释：

对输入层到隐藏层的参数包含  $W$  和  $b$ ，我们需要的是  $W$ ，这里的  $W$  是一个矩阵， $\text{shape}=(N, V)$ 。其中  $V$  是上文所述的词表的大小， $N$  是需要生成的词向量的维数。 $N$  同样也是隐藏层（第一层）中的隐藏节点个数。

每次一个 `batch_size` 输入其实一个矩阵(`batch_size`,  $V$ )，记为  $X$ ，隐藏层输出为  $Y$ ，公式为  $Y = XW^T + b$ 。所有的输入共享一个  $W$ ，每次迭代的时候都在修改  $W$ ，由于 one-hot 的性质，每次修改  $W$  只修改 1 对应的那一行。而这一行也就是词向量（转置后）

神经网络像是一个黑盒子，这其中的概念很难理解，这里给出我对词向量训练的个人理解：对于每个词  $s$ ，训练数据对应的标记是另一个词  $t$ ，训练其实是想找到一种映射关系，让  $s$  映射到  $t$ 。但很显然我们不是希望找到一个线性函数，使得给定  $s$  一定能得到  $t$ ，我们希望通过  $s$  得到一类词  $T$ ，包含  $t$ 。对于  $T$  中的每个  $t$ ，由于在  $s$  上下文中出现的频次不同，自然能得到一个概率，频次越高说明  $s$  与  $t$  相关性越高。

对于词向量，或者说参数矩阵  $W$ ，可以认为是一个将词映射到语义空间的桥梁， $s$  与  $t$  相关性越高，则认为其在语义空间中越近，那么对应的桥梁也越靠近。如果用向量来理解的话就是向量之前的夹角越小，我们使用向量来表示这个词的信息，重要的是得到了语义信息。在实际应用中，生成一段文本，我们可以判断词与词的向量之间相似度，如果过低则需要怀疑是否正确了。

#### 4、softmax 的原理了解

参考回答：

考虑一个多分类问题，即预测变量  $y$  可以取  $k$  个离散值中的任何一个。比如一个邮件分类系统将邮件分为私人邮件，工作邮件和垃圾邮件。由于  $y$  仍然是一个离散值，只是相对于二分类的逻辑回归多了一些类别。下面将根据多项式分布建模。

考虑将样本共有  $k$  类，每一类的概率分别为  $\phi_1, \dots, \phi_k$ ，由于  $\sum_{i=1}^k \phi_i = 1$ ，所以通常我们只需要  $k-1$  个参数  $\phi_1, \dots, \phi_{k-1}$  即可

$$\phi_i = p(y = i; \phi), \quad p(y = k; \phi) = 1 - \sum_{i=1}^{k-1} \phi_i$$

为了推导，引入表达式：

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(3) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, T(k-1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, T(k) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

上面  $T(y)$  是  $k-1$  维列向量，其中  $y = 1, 2, \dots, k$ 。

$T(y)_i$  表示向量  $T(y)$  的第  $i$  个元素。

还要引入表达式  $1\{\cdot\}$ ，如果大括号里面为真，则真个表达式就为 1，否则为 0。例如： $1\{2=3\} = 0$  和  $1\{3=3\} = 1$ 。

则上面的  $k$  个向量就可以表示为  $(T(y))_i = 1\{y = i\}$ 。

以为  $y$  只能属于某一个类别，于是  $T(y)$  中只能有一个元素为 1 其他元素都为 0，可以求出  $k-1$  个元素的期望： $E[(T(y))_i] = P(y = i) = \phi_i$ 。

定义：
$$\eta_i = \log \frac{\phi_i}{\phi_k}$$

其中  $i = 1, 2, \dots, k$ 。则有：

$$\begin{aligned} e^{\eta_i} &= \frac{\phi_i}{\phi_k} \\ \phi_k e^{\eta_i} &= \phi_i \\ \phi_k \sum_{i=1}^k e^{\eta_i} &= \sum_{i=1}^k \phi_i = 1 \end{aligned}$$

也很容易得出： $\phi_k = 1 / \sum_{i=1}^k e^{\eta_i}$ ，由该式和上面使得等式： $\phi_k e^{\eta_i} = \phi_i$  一起可以得到： $\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}}$  这个函数就是 softmax 函数。

然后假设  $\eta_i$ 's 和  $x$ 's 具有线性关系，即  $\eta_i = \theta_i^T x$  (for  $i = 1, \dots, k-1$ )

于是从概率的角度出发：

$$\begin{aligned} p(y = i|x; \theta) &= \phi_i \\ &= \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}} \\ &= \frac{e^{\theta_i^T x}}{\sum_{j=1}^k e^{\theta_j^T x}} \end{aligned}$$

其中  $y \in \{1, \dots, k\}$  这个模型就是 softmax 回归 (softmax regression)，它是逻辑回归的泛化。

这样我们的输出：

$$\begin{aligned} h_{\theta}(x) &= E[T(y)|x; \theta] \\ &= E \left[ \begin{bmatrix} 1\{y=1\} \\ 1\{y=2\} \\ \vdots \\ 1\{y=k-1\} \end{bmatrix} \middle| x; \theta \right] \\ &= \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{k-1} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\exp(\theta_1^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \frac{\exp(\theta_2^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \vdots \\ \frac{\exp(\theta_{k-1}^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \end{bmatrix} \end{aligned}$$

就是输出了  $x$  属于  $(1, 2, \dots, k-1)$  中每一类的概率，当然属于第  $k$  类的概率就是：  
 $1 - \sum_{i=1}^{k-1} \phi_i$

下面开始拟合参数

同样使用最大化参数  $\theta$  的对数似然函数：

$$\begin{aligned} \ell(\theta) &= \sum_{i=1}^m \log p(y^{(i)}|x^{(i)}; \theta) \\ &= \sum_{i=1}^m \log \prod_{l=1}^k \left( \frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}} \right)^{1\{y^{(i)}=l\}} \end{aligned}$$

这里使用梯度下降和牛顿法均可。

## 5、Word2vec 公式

参考回答：

Hierarchical Softmax

CBOW

$$\mathbf{v}(\tilde{w}) := \mathbf{v}(\tilde{w}) + \eta \sum_{j=2}^{l^w} \frac{\partial \mathcal{L}(w, j)}{\partial \mathbf{x}_w}, \quad \tilde{w} \in \text{Context}(w).$$

Skip-gram

$$\mathbf{v}(w) := \mathbf{v}(w) + \eta \sum_{u \in \text{Context}(w)} \sum_{j=2}^{l^u} \frac{\partial \mathcal{L}(w, u, j)}{\partial \mathbf{v}(w)}$$

Negative Sampling

CBOW

$$\mathbf{v}(\tilde{w}) := \mathbf{v}(\tilde{w}) + \eta \sum_{u \in \{w\} \cup \text{NEG}(w)} \frac{\partial \mathcal{L}(w, u)}{\partial \mathbf{x}_w}, \quad \tilde{w} \in \text{Context}(w).$$

Skip-gram

$$\mathbf{v}(\tilde{w}) := \mathbf{v}(\tilde{w}) + \eta \sum_{u \in \{w\} \cup \text{NEG}(\tilde{w})} [L^w(u) - \sigma(\mathbf{v}(\tilde{w})^\top \theta^u)] \theta^u.$$

## 6、Word2vec 公式

参考回答：

Hierarchical Softmax

CBOW

$$\mathbf{v}(\tilde{w}) := \mathbf{v}(\tilde{w}) + \eta \sum_{j=2}^{l^w} \frac{\partial \mathcal{L}(w, j)}{\partial \mathbf{x}_w}, \quad \tilde{w} \in \text{Context}(w).$$

Skip-gram

$$\mathbf{v}(w) := \mathbf{v}(w) + \eta \sum_{u \in \text{Context}(w)} \sum_{j=2}^{l^u} \frac{\partial \mathcal{L}(w, u, j)}{\partial \mathbf{v}(w)}$$

Negative Sampling

CBOW

$$\mathbf{v}(\tilde{w}) := \mathbf{v}(\tilde{w}) + \eta \sum_{u \in \{w\} \cup NEG(w)} \frac{\partial \mathcal{L}(w, u)}{\partial \mathbf{x}_w}, \quad \tilde{w} \in Context(w)$$

Skip-gram

$$\mathbf{v}(\tilde{w}) := \mathbf{v}(\tilde{w}) + \eta \sum_{u \in \{w\} \cup NEG(w)} [L^w(u) - \sigma(\mathbf{v}(\tilde{w})^\top \theta^u)] \theta^u$$

## 7、使用 gensim 的 word similar 方法预测句子

参考回答：

利用 gensim 训练 Word2vec 向量，得到词向量空间，通过词向量空间预测词之间的相似度，从而去预测由词组成的句子之间的相似度。

# 八、计算机基础

## 1、linux

### 1、ELF 的 bss 段

考点:Linux 基础

参考回答：

ELF 是 Linux 系统下的一种可执行可链接文件的格式,而 bss 段则是用于存放程序中未初始化的全局变量和静态局部变量。

## 2、计算机网络

### 1、ip 报文经过一个路由器改变哪些字段?

考点:计算机网络基础

参考回答：

源和目的 IP 地址,源和目的 MAC 地址以及 TTL 值。

### 2、TCP/IP 算法, IP 寻址

考点:计算机网络





参考回答：

略

### 3、操作系统

#### 1、如何将小端存储模式转为大端存储模式

考点:数据存储方式

参考回答：

```
def swap(val):  
  
    val=((val<<8) & 0xFF00FF00 | ((val>>8) & 0x00FF00FF)  
  
    return (val<<16)|(val>>16)
```

#### 2、Python 锁

参考回答：

Python 中的各种锁：

##### 一、全局解释器锁（GIL）

##### 1、什么是全局解释器锁

每个 CPU 在同一时间只能执行一个线程，那么其他的线程就必须等待该线程的全局解释器，使用权消失后才能使用全局解释器，即使多个线程直接不会相互影响在同一个进程下也只有一个线程使用 cpu,这样的机制称为全局解释器锁(GIL)。GIL 的设计简化了 CPython 的实现，使的对象模型包括关键的内置类型，如：字典等，都是隐含的，可以并发访问的，锁住全局解释器使得比较容易的实现多线程的支持，但也损失了多处理器主机的并行计算能力。

##### 2、全局解释器锁的好处

1)、避免了大量的加锁解锁的好处

2)、使数据更加安全，解决多线程间的数据完整性和状态同步

##### 3、全局解释器的缺点

多核处理器退化成单核处理器，只能并发不能并行。

##### 4、GIL 的作用：

多线程情况下必须存在资源的竞争，GIL 是为了保证在解释器级别的线程唯一使用共享资源（cpu）。

## 二、同步锁

### 1、什么是同步锁？

同一时刻的一个进程下的一个线程只能使用一个 cpu，要确保这个线程下的程序在一段时间内被 cpu 执，那么就要用到同步锁。

### 2、为什么用同步锁？

因为有可能当一个线程在使用 cpu 时，该线程下的程序可能会遇到 io 操作，那么 cpu 就会切到别的线程上去，这样就有可能影响到该程序结果的完整性。

### 3、怎么使用同步锁？

只需要在对公共数据的操作前后加上上锁和释放锁的操作即可。

### 4、同步锁的所用：

为了保证解释器级别下的自己编写的程序唯一使用共享资源产生了同步锁。

## 三、死锁

### 1、什么是死锁？

指两个或两个以上的线程或进程在执行程序的过程中，因争夺资源或者程序推进顺序不当而相互等待的一个现象。

### 2、死锁产生的必要条件？

互斥条件、请求和保持条件、不剥夺条件、环路等待条件

### 3、处理死锁的基本方法？

预防死锁、避免死锁（银行家算法）、检测死锁（资源分配）、解除死锁：剥夺资源、撤销进程

## 四、什么是递归锁？

在 Python 中为了支持同一个线程中多次请求同一资源，Python 提供了可重入锁。这个 RLock 内部维护着一个 Lock 和一个 counter 变量，counter 记录了 acquire 的次数，从而使得资源可以被多次 require。直到一个线程所有的 acquire 都被 release，其他的线程才能获得资源。递归锁分为可递归锁与非递归锁。

## 五、什么是乐观锁？

假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。

六、什么是悲观锁？

假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作。

七、python 常用的加锁方式？

互斥锁、可重入锁、迭代死锁、互相调用死锁、自旋锁。

## 4、数据库

1、count(\*), count(1) 和 count(列名) 的区别

考点:mysql 基础

参考回答：

count(\*), count(1) 在统计的时候不会忽略 Null, count(列名) 在统计的时候会忽略 Null。若列名为主键, count(列名) 会比 count(1), count(\*) 快, 反之则 count(1), count(\*) 更快。表中有多列且没有主键, 则 count(1) 执行效率优于 count(\*)。若表中只有一个字段则 count(\*) 最快。

## 九、场景题

1、如何对 10 亿个词语进行排序, 找出频率最高的 100 个

考点: 大数据处理

参考回答：

顺序读取每个词  $x$ , 取  $\text{hash}(x)/m$ , 然后按照该值存到这  $m$  个小文件中, 对于每个小文件, 依次统计每个文件中出现的词以及相应的频率, 统计出出现频率最高的 100 个词, 并将这 100 个词和频率存入文件, 从而得到  $m$  个新的文件然后将这  $m$  个文件进行归并找出出现频率最高的 100 个词。

2、10 亿个 32 位的数据, 放到 4G 的空间里, 怎么找出只出现一次的数据的个数。

参考回答：

32 位数据共有  $2^{32}$  个数, 因此设置一个  $2^{32}$  大小的数组 (4g)。遍历数据, 把每一个数放到数组中对应下标的位置, 个数加一。数组中最终值等于一的下标即为要找的数据。计算的过程中注意数据正负值和下标的对换。

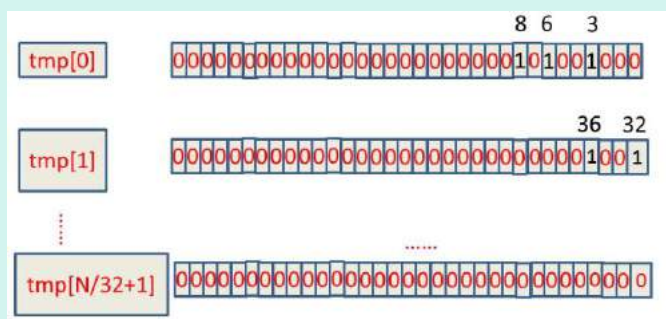
3、算法题, 10 亿个 32 位正整数, 求不同值, 只给 1GB 内存...

参考回答：

位图法是基于 int 型数的表示范围这个概念的，用一个 bit 位来标识一个 int 整数，若该位为 1，则说明该数出现；若该位为 0，则说明该数没有出现。一个 int 整型数占 4 字节 (Byte)，也就是 32 位 (bit)。那么把所有 int 整型数字表示出来需要  $2^{32}$  bit 位的空间，为了方便，我们可以把这些信息每 8bit 分割保存为 byte 数组，换算成字节单位也就是  $2^{32} \text{ bit} / 8 = 2^{29} \text{ Byte}$ ，大约等于 512MB

具体方案：那么接下来我们只需要申请一个 int 数组长度为 `int tmp[N/32+1]` 即可存储完这些数据，其中 N 代表要进行查找的总数（这里也就是  $2^{32}$ ），tmp 中的每个元素在内存存在占 32 位可以对应表示十进制数  $0 \sim 31$ ，所以可得到 BitMap 表：`tmp[0]`：可表示  $0 \sim 31$ ，`tmp[1]`：可表示  $32 \sim 63$ ，`tmp[2]` 可表示  $64 \sim 95$ ，~~

假设这 10 亿 int 数据为：6, 3, 8, 32, 36, ……，那么具体的 BitMap 表示为：



(1). 如何判断 int 数字放在哪一个 tmp 数组中：将数字直接除以 32 取整数部分 ( $x/32$ )，例如：整数 8 除以 32 取整等于 0，那么 8 就在 `tmp[0]` 上；

(2). 如何确定数字放在 32 个位中的哪个位：将数字  $\text{mod} 32$  取模 ( $x\%32$ )。上例中我们如何确定 8 在 `tmp[0]` 中的 32 个位中的哪个位，这种情况直接  $\text{mod}$  上 32 就 ok，又如整数 8，在 `tmp[0]` 中的第  $8 \text{ mod } 32$  等于 8，那么整数 8 就在 `tmp[0]` 中的第八个 bit 位（从右边数起）。

然后我们怎么统计只出现一次的数呢？每一个数出现的情况我们可以分为三种：0 次、1 次、大于 1 次。也就是说我们需要用 2 个 bit 位才能表示每个数的出现情况。此时则三种情况分别对应的 bit 位表示是：00、01、11

我们顺序扫描这 10 亿的数，在对应的双 bit 位上标记该数出现的次数。最后取出所有双 bit 位为 01 的 int 型数就可以了。

```
class BitmapTest {  
  
    private static final int CAPACITY = 1000000000; // 数据容量  
  
    // 定义一个 byte 数组缓存所有的数据  
  
    private byte[] dataBytes = new byte[1 << 29];  
}
```



```
public static void main(String[] args) {

    BitmapTest ms = new BitmapTest();

    byte[] bytes = null;

    Random random = new Random();

    for (int i = 0; i < CAPACITY; i++) {

        int num = random.nextInt();

        System.out.println("读取了第 " + (i + 1) + "\t个数: " + num);

        bytes = ms.splitBigData(num);

    }

    System.out.println("");

    ms.output(bytes);

}

/**
 * 读取数据，并将对应数数据的 到对应的 bit 中，并返回 byte 数组
 *
 * @param num 读取的数据
 *
 * @return byte 数组  dataBytes
 */

private byte[] splitBigData(int num) {

    //获取 num 数据对应 bit 数组（虚拟）的索引

    long bitIndex = num + (11 << 31);

    int index = (int) (bitIndex / 8); //bit 数组（虚拟）在 byte 数组中的索引

    //bitIndex 在 byte[]数组索引 index 中的具体位置

    int innerIndex = (int) (bitIndex % 8);

    System.out.println("byte[" + index + "]" 中的索引: " + innerIndex);

    dataBytes[index] = (byte) (dataBytes[index] | (1 << innerIndex));

}
```



```
        return dataBytes;

    }

    /**

    * 输出数组中的数据

    * @param bytes byte 数组

    */

    private void output(byte[] bytes) {

        int count = 0;

        for (int i = 0; i < bytes.length; i++) {

            for (int j = 0; j < 8; j++) {

                if (!(((bytes[i]) & (1 << j)) == 0)) {

                    count++;

                    int number = (int) (((long) i * 8 + j) - (11 << 31));

                    System.out.println("取出的第"+ count+"\t个数: "+

number);

                }

            }

        }

    }

}
```

#### 4、AI 能用在游戏的哪些方面。

参考回答：

人工智能在游戏中的目标主要有五个：一是为玩家提供适合的挑战；二是使玩家处于亢奋状态；三是提供不可预知性结果；四是帮助完成游戏的故事情节；五是创造一个生动的世界。这个生动的世界可以是类似现实生活中的世界，也可以是与现实世界完全不同的世界。但不管何种世界都要求有一整套能够自圆其说的游戏规则。

#### 5、如果让我用 AI 技术怎么加入 AI 元素

参考回答：

用自然语言处理或者语义理解改变问答体系，文本理解生成相关 AR 画面。

#### 6、你觉得你的构想能实际实现吗？

参考回答：

不能，以我学习的专业知识来讲，有些功能不能完全商业化实现，确实对主线剧情没有影响，但是作为彩蛋副本是完全可以的。

#### 7、那这个技术加进去有什么实际上的意义？

参考回答：

改变与玩家的交互方式后实际获得的用户信息更为真实，采用选项问答的方式会影响玩家的真实选择，在剧情人物上导向性太强会丧失乐趣。

## 十、项目

#### 1、项目中涉及的算法有了解情况

2、模型的搭建，后处理，数据中发现的特征，发现的亮点。

3、数据量和涉及的算法，效果。

4、你是怎么处理数据中经常存在的数据不平衡的问题。

5、考察项目中的 roi-pooling

6、自我介绍

7、项目介绍

参考回答：

项目名称：NetStore 网上购书系统

项目描述：该系统分为首页、用户管理、购物车管理、订单管理、退出 5 个模块。首页模块实现了模糊分页查询、分类分页查询等功能；用户管理模块实现了登录、注册、修改、查看信息等功能；购物车模块实现了查看、购买、修改图书数量等功能；订单模块实现了查看、删除等功能。

责任描述：个人独立完成了整个项目设计，开发，测试等。此系统整体采用 MVC 模式的 Struts 框架，持久层使用的是 Spring 的 HibernateTemplate 实现，数据源利用的是 SpringIoC 注入；模型层严格按照 JavaBean 规范要求；用 Struts 进行流程的控制，并实现了





国际化，JSP 用纯标签进行页面显示。为达到用户名唯一的目的用户注册采用 Ajax 技术进行后台校验。运用 SpringIoC 的注入对各层解耦，大大提高了程序的可扩展性，易于维护。

开发环境：数据库：MySQL；

JAVA 应用服务器：tomcat6.0；

技术选型：Spring、Struts、Hibernate、JavaBeans、Jsp。

项目总结：通过这个项目使我对 MVC 模式的认识更加的清楚，看到了 SpringIoC 在解决程序松散耦合方面的强大之处。JSP 页面纯标签开发所带来的方便以及页面的美观。

## 8、问了下项目怎么做的

参考回答：项目是根据甲方提的需求，我们制定方案。以 TensorFlow 为框架，利用 LSTM+CNN 方法进行开发。

## 9、问了一下项目和简历

### 10、描述一个算法项目从 kickoff-落地的全过程

参考回答：

撰写项目指南，分析项目需求，制定项目方案，确定技术路线，开始实施，测试，反馈，改进，上线运行。

### 11、扣项目，问简历，其中涉及的算法和上面差不多

参考回答：

介绍了自己做的最好的项目，详细介绍了一下项目内容，说了一下项目成果，分析失败原因，自己得到的经验和教训。

### 12、对项目一些技术选型产生质疑，并友好的一起讨论了这个问题

参考回答：

详细讲解项目技术选择的原因，并讲解其优势和不足，有好讨论其他技术选型的优缺点。

### 13、扣简历的项目，扣的很细

参考回答：

介绍了自己做的最好的项目，详细介绍了一下项目内容，说了一下项目成果，分析失败原因，自己得到的经验和教训。

#### 14、扣简历

参考回答：

介绍了自己做的最好的项目，详细介绍了一下项目内容，说了一下项目成果，分析失败原因，自己得到的经验和教训。

#### 15、扣简历，问得太细了，每个项目都要回答如果再做一次，有什么改进的地方，both 算法上和模型选择上

参考回答：介绍了自己做的最好的项目，详细介绍了一下项目内容，说了一下项目成果，分析失败原因，自己得到的经验和教训。如果再做一遍，会有什么改变。

#### 16、聊简历项目，对搜索推荐算法的了解

参考回答：当用户输入关键词查询的时候，如何让用户查询更准确呢？我们设想，针对用户的输入，我们如果能给出若干个和用户输入的关键词相似度很近的词，以这些作为查询条件，如果我们的算法足够好，搜索出来的结果会大大增加检索的准确度。下面给出具体的算法思路：

从向量化的角度来看，每一篇文档都对应一个向量  $w(w_1, w_2, \dots, w_n)$ ，其中  $w_i$  表示特征项  $i$ 。  $w_i$  是一个向量由词、词的位置、TF 等义项来确定的。对于版本 1，我们只取了词、词的位置。我们先用分类规则，把文档分成若干类，基于每一类进行如下计算：

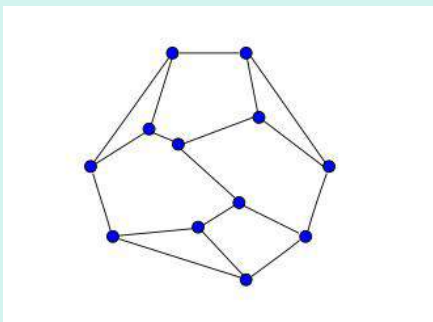
$r_{ij} = r(w_i, w_j)$  表示两个特征项的相似度。

我们定义一下距离公式

$$d_{ij} = d(w_i, w_j) = \min_{w \in D} \max_{w \in D} r(w_i, w_j)$$

我们对于每个文档的特征项，两两求出特征项的相似度。通过这个距离公式

我们可以得出，对于每一个分类，以这些特征项为顶点，以相似度距离为边，就构成了如下的无向图。





(lawnet)

类比于 wordnet 和知网的 hownet，我们称这个无向图，为 lawnnet。

那我们的设想问题就转化为：选取任意一个顶点，找出若干个（譬如 10 个）由这些顶点组成的最小生成树或者边权之和最小的最小子图。这是一个局部最优的随机问题。也就是说，我们只需要满足用户认可的体验程度即可，如果概率为 90%，也就是说，当用户输入 10000 次，我们能成功给出 9000 次的提示词就行了。

#### 17、简历上聚类项目用到的 ISODATA 算法比 kmeans 有哪些改进

参考回答：1) K-means 算法通常适合于分类数目已知的聚类，而 ISODATA 算法则更加灵活；2) 从算法角度看，ISODATA 算法与 K-means 算法相似，聚类中心都是通过样本均值的迭代运算来决定的；3) ISODATA 算法加入了一些试探步骤，并且可以结合成人机交互的结构，使其能利用中间结果所取得的经验更好地进行分类。

#### 18、自我介绍

参考回答：介绍了基本信息，说了一下自己的业务技能，讲了一下实习经历，自己做过的项目。

#### 19、然后让我说一下自己最印象深刻的项目。问我项目的最终成果，分析失败的原因。

#### 20、主要是问项目，根据项目里问一些细的技术点，比如 gan 在实际实现中的 loss 是什么

参考回答：GAN 同时要训练一个生成网络(Generator)和一个判别网络，前者输入一个 noise 变量  $z$ ，输出一个伪图片数据  $G(z; \theta_g)G(z; \theta_g)$ ，后者输入一个图片(real image)以及伪图片(fake image)数据  $x$ ，输出一个表示该输入是自然图片或者伪造图片的二分类置信度  $D(x; \theta_d)D(x; \theta_d)$ ，理想情况下，判别器  $D$  需要尽可能准确的判断输入数据到底是一个真实的图片还是某种伪造的图片，而生成器  $G$  又需要尽最大可能去欺骗  $D$ ，让  $D$  把自己产生的伪造图片全部判断成真实的图片。

根据上述训练过程的描述，我们可以定义一个损失函数：

$$Loss = \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^i)))$$

其中  $x_i, z_i$  分别是真实的图片数据以及 noise 变量。

而优化目标则是：

$$\min_G \max_D Loss$$

不过需要注意的一点是，实际训练过程中并不是直接在上述优化目标上对  $\theta_d, \theta_g$  计算梯度，而是分成几个步骤：训练判别器即更新  $\theta_d$ ：循环  $k$  次，每次准备一组 real image 数据  $x=x_1, x_2, \dots, x_m$  和一组 fake image 数据  $z=z_1, z_2, \dots, z_m$ ，计算



$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D(x^i) + \log(1 - D(G(z^i)))]$$

然后梯度上升法更新  $\theta_d$ ；训练生成器即更新  $\theta_g$ ：准备一组 fake image 数据  $z=z_1, z_2, \dots, z_m$ ，计算

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^i)))$$

然后梯度下降法更新  $\theta_g$ 。可以看出，第一步内部有一个 k 层的循环，某种程度上可以认为是因为我们的训练首先要保证判别器足够好然后才能开始训练生成器，否则对应的生成器也没有什么作用，然后第二步求提督时只计算 fake image 那部分数据，这是因为 real image 不由生成器产生，因此对应的梯度为 0。

## 21、主要是问项目

参考回答：详细讲解项目，项目的应用方面，落地，用到的主要技术。自己的职责，学到了什么，有什么缺点和不足。

项目名称：NetStore 网上购书系统

项目描述：该系统分为首页、用户管理、购物车管理、订单管理、退出 5 个模块。首页模块实现了模糊分页查询、分类分页查询等功能；用户管理模块实现了登录、注册、修改、查看信息等功能；购物车模块实现了查看、购买、修改图书数量等功能；订单模块实现了查看、删除等功能。

责任描述：个人独立完成了整个项目设计，开发，测试等。此系统整体采用 MVC 模式的 Struts 框架，持久层使用的是 Spring 的 HibernateTemplate 实现，数据源利用的是 SpringIoC 注入；模型层严格按照 JavaBean 规范要求；用 Struts 进行流程的控制，并实现了国际化，JSP 用纯标签进行页面显示。为达到用户名唯一的目的用户注册采用 Ajax 技术进行后台校验。运用 SpringIoC 的注入对各层解耦，大大提高了程序的可扩展性，易于维护。

开发环境：数据库：MySQL；

JAVA 应用服务器：tomcat6.0；

技术选型：Spring、Struts、Hibernate、JavaBeans、Jsp。

项目总结：通过这个项目使我对 MVC 模式的认识更加的清楚，看到了 SpringIoC 在解决程序松散耦合方面的强大之处。JSP 页面纯标签开发所带来的方便以及页面的美观。

## 22、 常规的自我介绍，两个面试官面试



参考回答：介绍了基本信息，说了一下自己的业务技能，讲了一下实习经历，自己做过的项目。

### 23、看过的论文，讨论论文

参考回答：讲一下和自己专业领域相关的论文，讨论一下主要方法，别紧张，平常心。

### 24、针对岗位需求和我简历里的内容进行提问

参考回答：如实回答简历内容相关问题，流利无卡顿，显得自信熟练，更增加印象分。

### 25、自我介绍

参考回答：介绍了基本信息，说了一下自己的业务技能，讲了一下实习经历，自己做过的项目。

### 26、项目经历介绍下

参考回答：该系统分为首页、用户管理、购物车管理、订单管理、退出 5 个模块。首页模块实现了模糊分页查询、分类分页查询等功能；用户管理模块实现了登录、注册、修改、查看信息等功能；购物车模块实现了查看、购买、修改图书数量等功能；订单模块实现了查看、删除等功能。

### 27、项目中遇到的最大困难

参考回答：项目需求讨论，确定方案，实施，一个项目从无到有，这个过程是艰辛的。最困难的就是方案制定。

### 28. 自我介绍

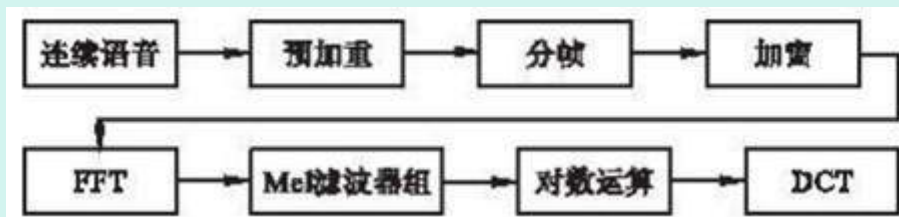
参考回答：介绍了基本信息，说了一下自己的业务技能，讲了一下实习经历，自己做过的项目。

### 29、针对简历里的第一个项目问的一些问题

- 1) 你分类的汽车类型有哪些
- 2) 基于信号特征还是神经网络进行分类的
- 3) MFCC 特征维数多少，有没有进行差分
- 4) 有没有试过使用其他声音信号来做分类？

5) MFCC 特征提取过程说下

参考回答:



30、针对项目 3，让解释下 DOA 估计

参考回答: DOA 估计 (或波达方向估计): 将接受信号进行空间傅里叶变换 (空间傅里叶变换和离散时间傅里叶变换的区别是, 空间傅里叶变换的求和是对阵元空间位置  $m$ , 而时域傅里叶变换的求和变量是离散时间  $n$ ), 进而取模的平方得到空间谱, 估计出信号的到达方向 (空间谱的最大值对应的相位  $\phi$ , 再根据定义  $\phi = 2\pi d \sin \theta / \lambda$ , 计算  $\theta$ )。

31、针对项目 4，问了以下几个问题

1) DTW 语音识别过程说下

参考回答: 在孤立词语音识别中, 最为简单有效的方法是采用 DTW (Dynamic Time Warping, 动态时间弯折) 算法, 该算法基于动态规划 (DP) 的思想, 解决了发音长短不一的模板匹配问题, 是语音识别中出现较早、较为经典的一种算法。

32、业界有哪些方法用来做语音识别?

参考回答: 第一种: 基于动态时间规整 (Dynamic Time Warping) 的算法; 第二种: 基于参数模型的隐马尔可夫模型 (HMM) 的方法; 第三种: 基于非参数模型的矢量量化 (VQ) 的方法。

3) 因为我说到了 HMM 语音识别, 所以面试官拓展问了下 Viterbi 算法解码语音和全路径搜索相比, 是有损还是无损? 最后的解码结果一样吗?

33、你的 C/C++ 怎么样?

参考回答: 如实回答自己的真是水平。

34、自我介绍?

参考回答: 介绍了基本信息, 说了一下自己的业务技能, 讲了一下实习经历, 自己做过的项目。

35、谈谈实习项目?



参考回答：讲了实习做的英语口语评测的工作，用到的语音识别框架（基于 WFST 的静态解码网络）

### 36、项目难点？

参考回答：讲了入门经历（万事开头难），详细的讲了强制对齐和 GMM 训练过程

### 37、说一下你简历里的图像识别的项目

参考回答：讲了项目里用到的网络以及训练方法，CNN 多说了一下，然后讲了 ResNet 的原理以及如何解决梯度消失的

### 38、来问我现在在做什么项目，然后我说 OCR，然后介绍了一下

参考回答：最近在做一个 OCR 相关的项目，OCR 技术是光学字符识别的缩写 (Optical Character Recognition)，是通过扫描等光学输入方式将各种票据、报刊、书籍、文稿及其它印刷品的文字转化为图像信息，再利用文字识别技术将图像信息转化为可以使用的计算机输入技术。

### 39、自我介绍

参考回答：介绍了基本信息，说了一下自己的业务技能，讲了一下实习经历，自己做过的项目。

### 40、项目经历详细介绍：两种预测方式区别，pair 的预测方式，整体项目有哪些可以提升的，遇到的困难之类的，整个项目用了哪些库？

参考回答：根据自己的项目情况详细介绍，介绍项目内容，项目成果，对比方法，用到了哪些方法，用的开源库等。

### 41、看过的论文，讨论论文

参考回答：讲一下和自己专业领域相关的论文，讨论一下主要方法，别紧张，平常心。

### 42、论文 flow 情况

参考回答：谈谈自己投稿的论文，论文投稿级别，论文内容，用到的方法，对比方法等

### 43、自我介绍



参考回答：介绍了基本信息，说了一下自己的业务技能，讲了一下实习经历，自己做过的项目。

#### 44、项目介绍：台球识别和分类使用的方法，Hough 变换原理、后处理

参考回答：

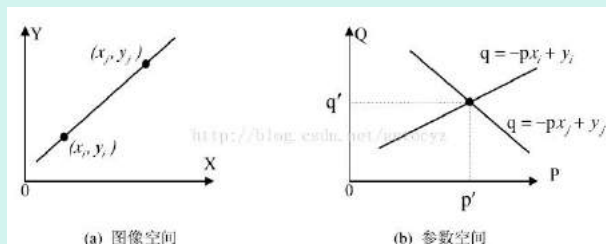
详细讲了一下项目背景，讲解了用到的检测方法。Hough 变换是基于点-线的对偶性思想。在图像 XY 里，所有过点  $(x, y)$  的直线的方程为

$$y = px + q \quad (1)$$

其中  $p$  为斜率， $q$  为截距，它也可以改写成如式(2)的形式：

$$q = -px + y \quad (2)$$

式(2)可以看作参数空间 PQ 中过点  $(p, q)$  的一条直线。如图下所示，在图像空间中 XY 中过点  $(x_i, y_i)$  的直线方程可以写成  $y_i = px_i + q$ ，也可以写成  $q = -px_i + y_i$ ，后者表示在参数空间 PQ 中的一条直线。同理对于  $(x_j, y_j)$  也可以写成上式形式。如下图：

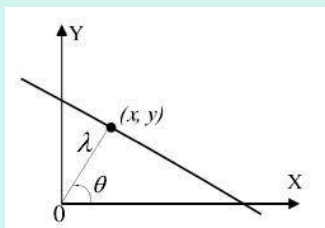


由此可知，在图像空间中共线的点对应在参数空间里面相交的线，反过来，在参数空间里面相交于同一个点的所有直线在图像空间里面都有共线的点与之对应，这就是点-线的对偶性。

Hough 变换就是根据这样的关系把空间里面的检测问题转换到参数空间，通过参数空间里面进行简单的累计统计来完成直线的检测任务。

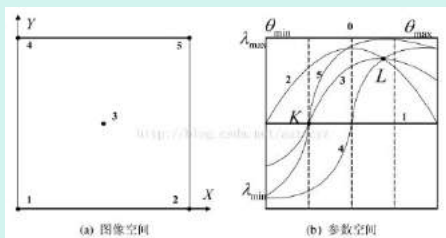
在运算式(2)直线方程时候，如果直线接近竖直方向，则会由于  $p$  的值都接近无穷而使得计算量增大，此时我们使用直线的极坐标方程来表示直线，如下图，其方程如式(3)，

$$\lambda = x \cos \theta + y \sin \theta \quad (3)$$





根据这个方程，对于任意一组  $(\lambda, \theta)$  都对应一条直线。即原来的点-线对偶性变成了现在的点-正弦曲线对偶性，如图所示，图(a)表示图像空间 XY 中的 5 个点，在图(b)参数空间中对应对应着 5 条曲线，这里的  $\theta$  为  $[-90, +90]$ ， $\lambda$  的取值范围为  $[-\frac{\sqrt{2}N}{2}, +\frac{\sqrt{2}N}{2}]$ ，N 为图像的宽度。



由图可知，图像空间中各个点可以看作它们在参数空间里面的对应曲线。在图(b)中，曲线 1, 3, 5 都过 K 点，这表示在图(a)中点 1, 3, 5 处于同一条直线上，同理，图(a)中点 2, 3, 4 处于同一条直线上，在图(b)中它们对应的曲线 2, 3, 4 都通过点 L。

Hough 变换的具体实现步骤如下：

- (1) 建立一个参数  $(\lambda, \theta)$  空间的二维的数组，该数组相当于一个累加器。
- (2) 顺序搜索图像中所有目标(黑色)像素，对于每一个目标像素，在参数空间中根据式(3)找到对应位置，然后在累加器的对应位置加 1。
- (3) 求出参数空间(累加器)中最大值，其位置为  $(\lambda', \theta')$ 。
- (4) 通过参数空间位置  $(\lambda', \theta')$ ，根据式(3)找到图像空间中相对应的直线参数。

对于直线 hough 变换过程，可以这么认为。依次遍历图像的所有像素，对每个像素判断是否满足某个特定条件，若满足，则经过该像素的所有直线区域的计数器加 1。为了得到经过某个像素的所有直线区域，可以依次用  $\theta$  的所有可能取值(例如  $\theta$  可以以 1 度为增量，从 -90 度取到 +90 度)，根据 (3) 式求出  $\lambda$  的值，从而得到很多组  $(\lambda, \theta)$ ，就对应了所有经过此像素的直线。

**45、Kaggle 比赛：背景介绍，数据清洗、数据增强、类别平衡，最终成绩，与前几名差距在哪，有没有尝试集成的方法。**

参考回答：详细介绍了比赛的背景，比赛项目，用到的数据处理过程，比赛成绩，名次，比赛经验总结，尝试过其他方法，当前方法是效果最好的。

**46、GAN 小论文：做了什么，最终效果**

参考回答：GAN 之所以是对抗的，是因为 GAN 的内部是竞争关系，一方叫 generator，它的主要工作是生成图片，并且尽量使得其看上去是来自于训练样本的。另一方是 discriminator，其目标是判断输入图片是否属于真实训练样本。更直白的讲，将 generator 想象成假币制造商，而 discriminator 是警察。generator 目的是尽可能把假币造的跟真的一样，从而能够骗过 discriminator，即生成样本并使它看上去好像来自于真实训练样本一样。生成对抗网络的一个简单解释如下：假设有两个模型，一个是生成模型 (Generative

Model，下文简称为 G)，一个是判别模型 (Discriminative Model，下文简称为 D)，判别模型 (D) 的任务就是判断一个实例是真实的还是由模型生成的，生成模型 (G) 的任务是生成一个实例来骗过判别模型 (D)，两个模型互相对抗，发展下去就会达到一个平衡，生成模型生成的实例与真实的没有区别，判别模型无法区分自然的还是模型生成的。以赝品商人为例，赝品商人 (生成模型) 制作出假的毕加索画作来欺骗行家 (判别模型 D)，赝品商人一直提升他的高仿水平来区分行家，行家也一直学习真的假的毕加索画作来提升自己的辨识能力，两个人一直博弈，最后赝品商人高仿的毕加索画作达到了以假乱真的水平，行家最后也很难区分正品和赝品了。

对于自己的论文，详细介绍一下，自己做了什么工作，有哪些提高和改进。并把自己的实验成果介绍一下。

#### 47、GAN 小论文，做了哪些工作，详细公式推一下，对 GAN 的具体应用有了解吗？

参考回答：GAN 之所以是对抗的，是因为 GAN 的内部是竞争关系，一方叫 generator，它的主要工作是生成图片，并且尽量使得其看上去是来自于训练样本的。另一方是 discriminator，其目标是判断输入图片是否属于真实训练样本。更直白的讲，将 generator 想象成假币制造商，而 discriminator 是警察。generator 目的是尽可能把假币造的跟真的一样，从而能够骗过 discriminator，即生成样本并使它看上去好像来自于真实训练样本一样。生成对抗网络的一个简单解释如下：假设有两个模型，一个是生成模型 (Generative Model，下文简称为 G)，一个是判别模型 (Discriminative Model，下文简称为 D)，判别模型 (D) 的任务就是判断一个实例是真实的还是由模型生成的，生成模型 (G) 的任务是生成一个实例来骗过判别模型 (D)，两个模型互相对抗，发展下去就会达到一个平衡，生成模型生成的实例与真实的没有区别，判别模型无法区分自然的还是模型生成的。以赝品商人为例，赝品商人 (生成模型) 制作出假的毕加索画作来欺骗行家 (判别模型 D)，赝品商人一直提升他的高仿水平来区分行家，行家也一直学习真的假的毕加索画作来提升自己的辨识能力，两个人一直博弈，最后赝品商人高仿的毕加索画作达到了以假乱真的水平，行家最后也很难区分正品和赝品了。

对于自己的论文，详细介绍一下，自己做了什么工作，有哪些提高和改进。并把自己的实验成果介绍一下。

GAN 同时要训练一个生成网络 (Generator) 和一个判别网络，前者输入一个 noise 变量  $z$ ，输出一个伪图片数据  $G(z; \theta_g)G(z; \theta_g)$ ，后者输入一个图片 (real image) 以及伪图片 (fake image) 数据  $x$ ，输出一个表示该输入是自然图片或者伪造图片的二分类置信度  $D(x; \theta_d)D(x; \theta_d)$ ，理想情况下，判别器 D 需要尽可能准确的判断输入数据到底是一个真实的图片还是某种伪造的图片，而生成器 G 又需要尽最大可能去欺骗 D，让 D 把自己产生的伪造图片全部判断成真实的图片。

根据上述训练过程的描述，我们可以定义一个损失函数：

$$Loss = \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^i)))$$

其中  $x_i, z_i$  分别是真实的图片数据以及 noise 变量。

而优化目标则是：



$$\min_G \max_D \text{Loss}$$

不过需要注意的一点是，实际训练过程中并不是直接在上述优化目标上对  $\theta_d$ ,  $\theta_g$  计算梯度，而是分成几个步骤：训练判别器即更新  $\theta_d$ ：循环  $k$  次，每次准备一组 real image 数据  $x=x_1, x_2, \dots, x_m$  和一组 fake image 数据  $z=z_1, z_2, \dots, z_m$ ，计算

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D(x^i) + \log(1 - D(G(z^i)))]$$

然后梯度上升法更新  $\theta_d$ ；训练生成器即更新  $\theta_g$ ：准备一组 fake image 数据  $z=z_1, z_2, \dots, z_m$ ，计算

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^i)))$$

然后梯度下降法更新  $\theta_g$ 。可以看出，第一步内部有一个  $k$  层的循环，某种程度上可以认为是因为我们的训练首先要保证判别器足够好然后才能开始训练生成器，否则对应的生成器也没有什么作用，然后第二步求提督时只计算 fake image 那部分数据，这是因为 real image 不由生成器产生，因此对应的梯度为 0。

#### 48、简历上项目为何适用 xgboost 和 lr，对比其他分类算法的场景优势。

参考回答：因为项目需要更高的效率，所以要用 xgboost 和 lr。

Xgboost:

优缺点：1) 在寻找最佳分割点时，考虑传统的枚举每个特征的所有可能分割点的贪心法效率太低，xgboost 实现了一种近似的算法。大致的思想是根据百分位法列举几个可能成为分割点的候选者，然后从候选者中根据上面求分割点的公式计算找出最佳的分割点。2) xgboost 考虑了训练数据为稀疏值的情况，可以为缺失值或者指定的值指定分支的默认方向，这能大大提升算法的效率，paper 提到 50 倍。3) 特征列排序后以块的形式存储在内存中，在迭代中可以重复使用；虽然 boosting 算法迭代必须串行，但是在处理每个特征列时可以做到并行。4) 按照特征列方式存储能优化寻找最佳的分割点，但是当以行计算梯度数据时会导致内存的不连续访问，严重时会导致 cache miss，降低算法效率。paper 中提到，可先将数据收集到线程内部的 buffer，然后再计算，提高算法的效率。5) xgboost 还考虑了当数据量比较大，内存不够时怎么有效的使用磁盘，主要是结合多线程、数据压缩、分片的方法，尽可能的提高算法的效率。

适用场景：分类回归问题都可以。

Lr:

优点：实现简单，广泛的应用于工业问题上；分类时计算量非常小，速度很快，存储资源低；便利的观测样本概率分数；对逻辑回归而言，多重共线性并不是问题，它可以结合 L2 正则化来解决该问题。

缺点：当特征空间很大时，逻辑回归的性能不是很好；容易欠拟合，一般准确度不太高

不能很好地处理大量多类特征或变量；只能处理两分类问题（在此基础上衍生出来的 softmax 可以用于多分类），且必须线性可分；对于非线性特征，需要进行转换。

适用场景：LR 同样是很多分类算法的基础组件，它的好处是输出值自然地落在 0 到 1 之间，并且有概率意义。因为它本质上是一个线性的分类器，所以处理不好特征之间相关的情况。虽然效果一般，却胜在模型清晰，背后的概率学经得住推敲。它拟合出来的参数就代表了每一个特征(feature)对结果的影响。也是一个理解数据的好工具。

#### 49、GAN 小论文，你做了什么，有哪些改进，在哪些数据集上做过实验，分辨率是多少？

参考回答：

GAN 之所以是对抗的，是因为 GAN 的内部是竞争关系，一方叫 generator，它的主要工作是生成图片，并且尽量使得其看上去是来自于训练样本的。另一方是 discriminator，其目标是判断输入图片是否属于真实训练样本。更直白的讲，将 generator 想象成假币制造商，而 discriminator 是警察。generator 目的是尽可能把假币造的跟真的一样，从而能够骗过 discriminator，即生成样本并使它看上去好像来自于真实训练样本一样。生成对抗网络的一个简单解释如下：假设有两个模型，一个是生成模型 (Generative Model, 下文简称为 G)，一个是判别模型 (Discriminative Model, 下文简称为 D)，判别模型 (D) 的任务就是判断一个实例是真实的还是由模型生成的，生成模型 (G) 的任务是生成一个实例来骗过判别模型 (D)，两个模型互相对抗，发展下去就会达到一个平衡，生成模型生成的实例与真实的没有区别，判别模型无法区分自然的还是模型生成的。以赝品商人为例，赝品商人（生成模型）制作出假的毕加索画作来欺骗行家（判别模型 D），赝品商人一直提升他的高仿水平来区分行家，行家也一直学习真的假的毕加索画作来提升自己的辨识能力，两个人一直博弈，最后赝品商人高仿的毕加索画作达到了以假乱真的水平，行家最后也很难区分正品和赝品了。

对于自己的论文，详细介绍一下，自己做了什么工作，有哪些提高和改进。并把自己的实验成果介绍一下。

#### 50、英特尔实习：1) 项目背景。台球检测和分类方法，球杆检测方法，球杆遮挡问题怎么处理，不用分类器，直接分割或计算图像差值会怎样？

参考回答：详细讲了一下项目背景，讲解了用到的检测方法，怎么处理的图像分类，直接分割或计算图像差值会变成什么样。

#### 51、有什么问题想了解一下

参考回答：问了一下岗位信息，和需要什么必备的专业技能，什么时候会有结果。

## 十一、hr 面

1. 自我介绍
2. 几点过来的？
3. 你有什么跟别人不一样的？



4. 你的缺点是什么？
5. 家里几口人？
6. 搞技术的你怎么很能说啊？
7. 为什么喜欢杭州？
8. 为什么喜欢菜鸟？
9. 如果你面试失败，你会怎么办？
10. 菜鸟跟你相关的部门？
11. 有女朋友吗？
12. 在哪里工作？
13. 从事什么行业？
14. 如果你的岗位有冲突，你会怎么处理？
15. 实习时间，具体什么时候可以开始？
16. 你有什么想问的吗？
17. 有没有男朋友呀，意向地点是哪？
18. 对公司有哪些了解
19. 兴趣爱好、意向地点
20. hr 面就是常见的问题，城市啊，薪资待遇啊，对部门的评价，对总监的评价
21. 自我评价优缺点，怎么改进

参考回答：优点：踏实肯干，有较强的团队协作意识，学习能力强，看待问题有独特的想法。缺点：过于自信，实践经验稍显不足。清楚明白的给自己定位，经常反思，不断进步。

22. hr 抠了简历细节，问了笔试试卷相关的问题（为什么工科生要选择做文案卷，笔试题目印象比较深的是什么呢等等）还有自己的职业规划等

参考回答：因为喜欢游戏，作为理科生，对于游戏很敏感，所以想做游戏策划，希望有朝一日自己策划的游戏得到大家的喜欢。对于自己的职业规划，希望自己在未来做好 AI 与游戏相结合，把自己的专业知识与游戏策划相融合，做出一款超棒的 AI 游戏。

23. 想去的工作城市

参考回答：北京，上海，大城市机会多，发展前景好。

24. 考研还是保研，为什么考到这个学校

参考回答：本科没把心思放到学习上，考研的。这个学校的专业很好，老师经验丰富，所在的城市也是我喜欢的。

25. 研究生期间最大的改变是什么

参考回答：开阔了视野，提高了自己的实践能力。

26. 四六级成绩



参考回答：如实回答。

27、讨论下工作地点和期望薪资

参考回答：期望在北京工作，期望薪资 20k/月。

28、你三年的职业规划是什么？

参考回答：在这三年中，在实践中提高自己的专业能力，争取成为领域专家。

29、你和周围的人比你的优势在哪？举个例子

参考回答：更善于学习一些新的技能，比较细心（结合实际吧，没有过于夸大）。

30、考研的还是保研的？为什么没有保研？

参考回答：（如实回答了）本科心思没全花在学习上，研究生阶段成绩还不错。

31、家庭成员？家里人对你的工作地点有要求吗？

参考回答：家里有什么人，家里人都赞同我的决定，没有要求我留在当地。

32、平时喜欢运动吗？

参考回答：喜欢，经常参加运动，各种球类都会。

33、设计模式会吗？

参考回答：会，本科开过这个课。

34、转岗吗？

参考回答：不转岗，我很喜欢我投递的岗位。

35、平时刷题吗？你刷过最有意思的题是啥，说一哈

参考回答：刷题，经常刷。最有意思的是做过一个动态规划的题目，和之前做过的很像，自己思考了很久才做出来，其实就差一个变量。

36、我看你之前还做过销售，聊一下

参考回答：是的，之前做过销售。是为了培养自己的社交能力，能与不同的人交流，学会随机应变，当然，顺便赚点生活费。

37、hr 面：确定并不是二面说的转岗，而是去做数据挖掘





参考回答：是的，确定不转岗。我想做数据挖掘，这也是我职业规划的一部分，我不想做自己不喜欢的工作

## 十二、数据挖掘

1、开放题:预测一位学生期末是否挂科，需要挖掘哪些信息。

参考回答：需要挖掘学生的以往上课时间和成绩，出勤率。

## 十三、学习与发展

1、如果有同学要学机器学习，你会建议他们从哪里做起

参考回答：

先打好数学基础：线性代数：矩阵/张量乘法、求逆，奇异值分解/特征值分解，行列式，范数等；统计与概率：概率分布，独立性与贝叶斯，最大似然 (MLE) 和最大后验估计 (MAP) 等；优化：线性优化，非线性优化 (凸优化/非凸优化) 以及其衍生的求解方法如梯度下降、牛顿法、基因算法和模拟退火等；微积分：偏微分，链式法则，矩阵求导等；信息论、数值理论等。选择合适的机器学习书籍，选择合适的编程语言和软件，实践最重要

2、个人发展，以及你对项目的深入思考，还有深度学习的发展等

参考回答：

掌握核心技术，提高实践能力，学会管理。项目中一定要协调好团队内的人与人之间的关系，同步推进项目，使用前沿技术，让项目更加完善。对于深度学习，下一步的发展重点在 迁移学习、强化学习、非监督学习 层面，对抗网络 (GAN) 也将会焕发新的光彩。而在应用领域，机器人将会在很大程度上推动深度学习的发展，也许在将来的 5-10 年，真的会诞生真正意义上的智能体，最近听说某某机器人通过了 图灵测试，很好奇，一看也只是在某个简单的层面。量子计算机、类脑、超脑 等一大批科技新星为之注入了强悍的体系支撑，全球在技术领域持续的资金和人才投入也将持续提供无限的动力，也许 AI 目前是泡沫，但若是支撑到某个领域的突破，必然会带来新的生产力变革，换句话说，科技泡沫总归比 房地产泡沫来得更实在一些。

3、讲很多关于深度学习以及机器学习的具体应用与实现，也谈到了自己在实际工作中遇到的困难。

参考回答：

讲述自己学习深度学习与机器学习的过程，理论与实现。具体谈谈自己在实际应用中遇到的困难，自己如何解决问题，克服困难等。

#### 4、问了些我对人工智能的看法，以及相关竞赛的情况

参考回答：

人工智能是为了模拟人类大脑的活动而产生的科学，人类已经可以用许多新技术新材料模拟人体的许多功能，诸如皮肤，毛发，骨骼等等，也就是说，人类可以创造出“类人体”。只要能够模拟人的大脑的功能，人就可以完成人工生命的研究工作，人创造自己，这不但在科学上，而且在哲学上都具有划时代的意义。这就是人工智能承担的历史使命。

如实回答自己的参赛情况，内容，获奖，名次等。

#### 5、自我介绍

参考回答：

介绍了基本信息，说了一下自己的业务技能，讲了一下实习经历，自己做过的项目。

#### 6、为什么选择做语音方向？

参考回答：

毕业设计要做这个方向，迫不得已；入门之后又觉得挺有意思，然后就决定找这方面的工作。

#### 7、为什么选择实习，而不是在学校做研究？

参考回答：

我实习的部门就是做研发的，还有跟着项目做研究可以把理论用到实际中，还有在公司实习的好处就是有数据，因为数据决定了机器/深度学习结果的上限，算法只是无限接近这个上限（校招王者）

## 十四、惊喜福利

此面试题库将根据当下面试形式大数据随时更新，如果你已获得下载权限，那么你可以终身在牛币兑换中心里去兑换此面试题库的电子版，如果电子版有更新，会通过牛客站内信进行通知（前提是你已获得下载权限）。

牛币兑换中心：<https://www.nowcoder.com/coin/index>

还能兑换各种惊喜周边哦



牛客定制



热门商品



名企周边



专业书籍



虚拟商品

