比如C++的virtual关键字，

C的friend关键字，

inline关键字的作用和具体是怎么实现的

了一下LR的损失函数，SVM的损失函数

矩阵秩的几何意义，这个概念具体有什么用

写一个链表的数据结构，链表和数组的区别，在内存中的存放

走台阶问题，n个台阶，一次走一步或者两步，最后走n阶有多少种方法。让用两种方法实现，递归和非递归。

数据库

Linux的知识

网络的7层协议，帧在哪一层，socket在哪一层

应用 表示（API） 会话（socket）SPI是服务提供者接口，管理用户间的会话和对话；控制用户间的连接和挂断连接；报告上层错误 传输TCP 网络NDIS 数据链路（帧） 物理

hadoop，以wordcount为例子来说一下hadoop的数据走的流程，在sort阶段发生在什么时候

Hadoop是一个开发和运行处理大规模数据的软件平台,是Appach的一个用java语言实现开源软件框架，实现在大量计算机组成的集群中对海量数据进行分布式计算.

哪些聚类方法

k-means

CLARA

LR，问LR的损失函数

第一个问题是GBDT和随机森林的区别  
第二个问题是如何判断函数凸或非凸  
第三个问题是项目中特征选择都是怎么做的  
第四个问题是为什么会产生过拟合有哪些方法可以预防或克服过拟合

**欠拟合**

就是模型没有很好地捕捉到数据特征，不能够很好地拟合数据

添加其他特征项，有时候我们模型出现欠拟合的时候是因为特征项不够导致的，可以添加其他特征项来很好地解决。例如，“组合”、“泛化”、“相关性”三类特征是特征添加的重要手段，无论在什么场景，都可以照葫芦画瓢，总会得到意想不到的效果。除上面的特征之外，“上下文特征”、“平台特征”等等，都可以作为特征添加的首选项。

2）添加多项式特征，这个在机器学习算法里面用的很普遍

**过拟合**

通俗一点地来说过拟合就是模型把数据学习的太彻底，以至于把噪声数据的特征也学习到了，这样就会导致在后期测试的时候不能够很好地识别数据，即不能正确的分类，模型泛化能力太差。

我们用于训练的数据量太小导致的，训练数据占总数据的比例过小。

重新清洗数据，导致过拟合的一个原因也有可能是数据不纯导致的，如果出现了过拟合就需要我们重新清洗数据。

采用dropout方法。这个方法在神经网络里面很常用。

随机森林、

1. 决策树

分类树以C4.5算法为例，C4.5分类树在每次分枝时，是穷举每一个feature的每一个阈值，找到使得按照feature<=阈值，和feature>阈值分成的两个分枝的熵最大的阈值(熵最大的概念可理解成尽可能每个分枝的男女比例都远离1:1)，按照该标准分枝得到两个新节点，用同样方法继续分枝直到所有人都被分入性别唯一的叶子节点，或达到预设的终止条件，若最终叶子节点中的性别不唯一，则以多数人的性别作为该叶子节点的性别。

回归树总体流程类似于分类树，区别在于，回归树的每一个节点都会得一个预测值，以年龄为例，该预测值等于属于这个节点的所有人年龄的平均值。分枝时穷举每一个feature的每个阈值找最好的分割点，但衡量最好的标准不再是最大熵，而是最小化平方误差。也就是被预测出错的人数越多，错的越离谱，平方误差就越大，通过最小化平方误差能够找到最可靠的分枝依据。分枝直到每个叶子节点上人的年龄都唯一或者达到预设的终止条件(如叶子个数上限)，若最终叶子节点上人的年龄不唯一，则以该节点上所有人的平均年龄做为该叶子节点的预测年龄。

**线程进程关系**

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动

线程是进程的一个实体,是CPU调度和分派的基本单位

一个线程可以创建和撤销另一个线程;同一个进程中的多个线程之间可以并发执行.

线程是一个更加接近于执行体的概念

**1) 简而言之,一个程序至少有一个进程,一个进程至少有一个线程.**

2) 线程的划分尺度小于进程，使得多线程程序的并发性高。

3) 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

4) 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。**但是线程不能够独立执行，**必须依存在应用程序中，由应用程序提供多个线程执行控制。

5) 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。**这就是进程和线程的重要区别。**

应用层

App HTTP超文本传输协议 ftp文件传输协议

### 表示层

链接会话层和应用层 保密解码 assic码

### 会话层

什么时候恢复结束通信 socket应用层与传输层间的一个抽象层，是个编程接口

### 传输层

传报文 tcp udp（全是传输协议）dns TCP协议的封包处理是在这一层进行的

控制流量 数据包

### 网络层

里面是数据包 路径选择 寻址 ipv4 ipv6

### 数据链路层

### 怎么传数据 也是帧 物理 看介质 数据单位帧

### 物理层

光纤

TCP与UDP的区别

Tcp是面向连接 可靠 把数据流切成报文段有序号可以排序

滑动窗口协议：两个窗口不一样大确认接收才再发

Udp不可靠 把结果包传给ip层 dns

三次握手，四次挥手

建立连接要三次

断开要四次



Dns

是个域名解析 去找域名

Ospf

迪杰斯特拉算法

有个表让我一层一层按着找

找路 路由协议 最短路径优先

多态，虚函数

快速排序

随机拿个数 比她小的左边比他大的右边 然后递归 直到排完

ls　　        显示文件或目录

     -l           列出文件详细信息l(list)

     -a          列出当前目录下所有文件及目录，包括隐藏的a(all)

mkdir         创建目录

     -p           创建目录，若无父目录，则创建p(parent)

cd               切换目录

touch          创建空文件

echo            创建带有内容的文件。

cat              查看文件内容

cp                拷贝

mv               移动或重命名

rm               删除文件

     -r            递归删除，可删除子目录及文件

     -f            强制删除

find              在文件系统中搜索某文件

wc                统计文本中行数、字数、字符数

grep             在文本文件中查找某个字符串

rmdir           删除空目录

tree             树形结构显示目录，需要安装tree包

pwd              显示当前目录

ln                  创建链接文件

more、less  分页显示文本文件内容

head、tail    显示文件头、尾内容

ctrl+alt+F1  命令行全屏模式

const定义的常量在超出其作用域之后其空间会被释放，而static定义的静态常量在函数执行后不会释放其存储空间。

      static表示的是静态的。类的静态成员函数、静态成员变量是和类相关的，而不是和类的具体对象相关的。即使没有具体对象，也能调用类的静态成员函数和成员变量。一般类的静态函数几乎就是一个全局函数，只不过它的作用域限于包含它的文件中。

      在C++中，static静态成员变量不能在类的内部初始化。在类的内部只是声明，定义必须在类定义体的外部，通常在类的实现文件中初始化，如：double Account::Rate=2.25;static关键字只能用于类定义体内部的声明中，定义时不能标示为static

      在C++中，const成员变量也不能在类定义处初始化，只能通过构造函数初始化列表进行，并且必须有构造函数。

      const数据成员 只在某个对象生存期内是常量，而对于整个类而言却是可变的。因为类可以创建多个对象，不同的对象其const数据成员的值可以不同。所以不能在类的声明中初始化const数据成员，因为类的对象没被创建时，编译器不知道const数据成员的值是什么。

      const数据成员的初始化只能在类的构造函数的初始化列表中进行。要想建立在整个类中都恒定的常量，应该用类中的枚举常量来实现，或者static cosnt。

strcpy, strncpy, memcpy