有监督学习和无监督学习的区别

有监督学习：对具有标记的训练样本进行学习，以尽可能对训练样本集外的数据进行分类预测。（LR,SVM,BP,RF,GBDT）
无监督学习：对未标记的样本进行训练学习，比发现这些样本中的结构知识。(KMeans,DL)

正则化

正则化是针对过拟合而提出的，以为在求解模型最优的是一般优化最小的经验风险，现在在该经验风险上加入模型复杂度这一项（正则化项是模型参数向量的范数），并使用一个 rate 比率来权衡模型复杂度与以往经验风险的权重，如果模型复杂度越高，结构化的经验风险会越大，现在的目标就变为了结构经验风险的最优化，可以防止模型训练过度复杂，有效的降低过拟合的风险。奥卡姆剃刀原理，能够很好的解释已知数据并且十分简单才是最好的模型。

过拟合

如果一味的去提高训练数据的预测能力，所选模型的复杂度往往会很高，这种现象称为过拟合。所表现的就是模型训练时候的误差很小，但在测试的时候误差很大。
产生的原因
过拟合原因
1. 样本数据的问题。
样本数量太少
抽样方法错误，抽出的样本数据不能有效足够代表业务逻辑或业务场景。比如样本符合正态分布，却按均分分布抽样，或者样本数据不能代表整体数据的分布
样本里的噪音数据干扰过大
2. 模型问题
模型复杂度高 、参数太多
决策树模型没有剪枝
权值学习迭代次数足够多(Overtraining),拟合了训练数据中的噪声和训练样例中没有代表性的特征.
解决方法
1. 样本数据方面。
增加样本数量，对样本进行降维，添加验证数据
抽样方法要符合业务场景
清洗噪声数据
2. 模型或训练问题
控制模型复杂度，优先选择简单的模型，或者用模型融合技术。

利用先验知识，添加正则项。L1 正则更加容易产生稀疏解、L2 正则倾向于让参数 w 趋向于 0.
交叉验证
不要过度训练，最优化求解时，收敛之前停止迭代。
决策树模型没有剪枝
权值衰

线性分类器与非线性分类器的区别以及优劣

如果模型是参数的线性函数，并且存在线性分类面，那么就是线性分类器，否则不是。
常见的线性分类器有：LR,贝叶斯分类，单层感知机、线性回归
常见的非线性分类器：决策树、RF、GBDT、多层感知机 SVM 两种都有(看线性核还是高斯核)
线性分类器速度快、编程方便，但是可能拟合效果不会很好
非线性分类器编程复杂，但是效果拟合能力强

为什么 LR 要使用 Sigmod 函数？

说到底源于 sigmoid，或者说 exponential family 所具有的最佳性质，即 maximum entropy 的性质。虽然不清楚历史上孰先孰后，但这并不妨碍 maximum entropy 给了 logistic regression 一个很好的数学解释。为什么 maximum entropy 好呢？entropy 翻译过来就是熵，所以 maximum entropy 也就是最大熵。熵原本是 information theory 中的概念，用在概率分布上可以表示这个分布中所包含的不确定度，熵越大不确定度越大。所以大家可以想象到，均匀分布熵最大，因为基本新数据是任何值的概率都均等。而我们现在关心的是，给定某些假设之后，熵最大的分布。也就是说这个分布应该在满足我假设的前提下越均匀越好。比如大家熟知的正态分布，正是假设已知 mean 和 variance 后熵最大的分布。回过来看 logistic regression，这里假设了什么呢？首先，我们在建模预测 Y|X，并认为 Y|X 服从 bernoulli distribution, 所以我们只需要知道 P(Y|X) ;其次我们需要一个线性模型, 所以 P(Y|X) = f(wx)。接下来我们就只需要知道 f 是什么就行了。而我们可以通过最大熵原则推出的这个 f，就是 sigmoid。其实前面也有人剧透了 bernoulli 的 exponential family 形式，也即是 1/ (1 + e^-z)

LR 与 Liner SVM 区别

- Linear SVM 和 LR 都是线性分类器
- Linear SVM 不直接依赖数据分布，分类平面不受一类点影响；LR 则受所有数据点的影响，如果数据不同类别 strongly unbalance 一般需要先对数据做 balancing。
- Linear SVM 依赖数据表达的距离测度，所以需要对数据先做 normalization(归一化)；LR 不受其影响 Linear SVM 依赖 penalty 的系数，实验中需要做 validation
- Linear SVM 和 LR 的 performance 都会收到 outlier 的影响，其敏感程度而言，谁更好很

难下明确结论。

EM 算法的基本概念和应用场景？

最大期望（EM）算法是在概率（probabilistic）模型中寻找参数最大似然估计或者最大后验估计的算法，其中概率模型依赖于无法观测的隐藏变量（Latent Variable）。假设我们估计知道 A 和 B 两个参数，在开始状态下二者都是未知的，并且知道了 A 的信息就可以得到 B 的信息，反过来知道了 B 也就得到了 A。可以考虑首先赋予 A 某种初值，以此得到 B 的估计值，然后从 B 的当前值出发，重新估计 A 的取值，这个过程一直持续到收敛为止。
参考链接：http://www.tuicool.com/articles/Av6NVzy
最大期望经常用在机器学习和计算机视觉的数据聚类领域。

常见的分类算法有哪些？

SVM、神经网络、随机森林、逻辑回归、KNN、贝叶斯

请描述极大似然估计 MLE 和最大后验估计 MAP 之间的区别。请解释为什么 MLE 比 MAP 更容易过拟合。

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，即：“模型已定，参数未知”。简单而言，假设我们要统计全国人口的身高，首先假设这个身高服从服从正态分布，但是该分布的均值与方差未知。我们没有人力与物力去统计全国每个人的身高，但是可以通过采样，获取部分人的身高，然后通过最大似然估计来获取上述假设中的正态分布的均值与方差。最大后验估计是根据经验数据获得对难以观察的量的点估计。与最大似然估计类似，但是最大的不同时，最大后验估计的融入了要估计量的先验分布在其中。故最大后验估计可以看做规则化的最大似然估计。

SVM 为什么引入对偶问题？什么是对偶？

- 对偶问题往往更加容易求解(结合拉格朗日和 kkt 条件)

可以很自然的引用核函数（拉格朗日表达式里面有内积，而核函数也是通过内积进行映射的）

- 在优化理论中，目标函数 f(x) 会有多种形式：如果目标函数和约束条件都为变量 x 的线性函数，称该问题为线性规划； 如果目标函数为二次函数，约束条件为线性函数，称该最优化问题为二次规划；如果目标函数或者约束条件均为非线性函数，称该最优化问题为非线性规划。每个线性规划问题都有一个与之对应的对偶问题，对偶问题有非常良好的性质，以下列举几个：
  对偶问题的对偶是原问题；
  无论原始问题是否是凸的，对偶问题都是凸优化问题；
  对偶问题可以给出原始问题一个下界；
  当满足一定条件时，原始问题与对偶问题的解是完全等价的

判别式模型和生成式模型

判别方法：由数据直接学习决策函数 Y = f（X），或者由条件分布概率 P（Y|X）作为预测模型，即判别模型。
生成方法：由数据学习联合概率密度分布函数 P（X,Y）,然后求出条件概率分布 P(Y|X)作为预测的模型，即生成模型。
由生成模型可以得到判别模型，但由判别模型得不到生成模型。
常见的判别模型有：K 近邻、SVM、决策树、感知机、线性判别分析（LDA）、线性回归、传统的神经网络、逻辑斯蒂回归、boosting、条件随机场
常见的生成模型有 :朴素贝叶斯、隐马尔可夫模型、高斯混合模型、文档主题生成模型(LDA)、限制玻尔兹曼机

SVM 如何实现，SMO 算法如何实现？

SMO 是用于快速求解 SVM 的。
它选择凸二次规划的两个变量，其他的变量保持不变，然后根据这两个变量构建一个二次规划问题，这个二次规划关于这两个变量解会更加的接近原始二次规划的解，通过这样的子问题划分可以大大增加整个算法的计算速度，关于这两个变量：
其中一个是严重违反 KKT 条件的一个变量
另一个变量是根据自由约束确定，好像是求剩余变量的最大化来确定的。

如果训练样本数量少于特征数量，怎么办？

如果训练集很小，那么高偏差/低方差分类器（如朴素贝叶斯分类器）要优于低偏差/高方差分类器（如 k 近邻分类器），因为后者容易过拟合。然而，随着训练集的增大，低偏差/高方差分类器将开始胜出（它们具有较低的渐近误差），因为高偏差分类器不足以提供准确的模型。你也可以认为这是生成模型与判别模型的区别。维度高用非线性分类器，维度低用线性分类器。

Xgboost 是如何调参的？

XGBoost 的参数

- General Parameters：
  - booster：所使用的模型，gbtree 或 gblinear
  - silent：1 则不打印提示信息，0 则打印，默认为 0
  - nthread：所使用的线程数量，默认为最大可用数量
- Booster Parameters（gbtree）：
  - eta：学习率，默认初始化为 0.3，经多轮迭代后一般衰减到 0.01 至 0.2
  - min_child_weight：每个子节点所需的最小权重和，默认为 1
  - max_depth：树的最大深度，默认为 6，一般为 3 至 10
  - max_leaf_nodes：叶结点最大数量，默认为 2^6
  - gamma：拆分节点时所需的最小损失衰减，默认为 0
  - max_delta_step：默认为 0
  - subsample：每棵树采样的样本数量比例，默认为 1，一般取 0.5 至 1
  - colsample_bytree：每棵树采样的特征数量比例，默认为 1，一般取 0.5 至 1
  - colsample_bylevel：默认为 1
  - lambda：L2 正则化项，默认为 1
  - alpha：L1 正则化项，默认为 1
  - scale_pos_weight：加快收敛速度，默认为 1
- Learning Task Parameters：
  - objective：目标函数，默认为 reg:linear，还可取 binary:logistic、multi:softmax、multi:softprob
  - eval_metric：误差函数，回归默认为 rmse，分类默认为 error，其他可取值包括 rmse、mae、logloss、merror、mlogloss、auc
  - seed：随机数种子，默认为 0

解释一下 LDA?

主题模型是一个比较广的领域。Spark 1.3 加入了隐含狄利克雷分布（LDA），差不多是现今最成功的主题模型。最初被开发用于文本分析和群体遗传学，LDA 之后被不断拓展，应用到从时间序列分析到图片分析等问题。首先，我们从文本分析的角度描述 LDA。

什么是主题？主题不是 LDA 的输入，所以 LDA 必须要从纯文本中推断主题。LDA 将主题定义为词的分布。例如，当我们在一个 20 个新闻组的文章数据集上运行 MLlib 的 LDA，开始的几个主题是：

Adaboost、GBDT 和 Xgboost 的区别？

1. 传统 GBDT 以 CART 作为基分类器，xgboost 还支持线性分类器，这个时候 xgboost 相当于带 L1 和 L2 正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。
2. 2 传统 GBDT 在优化时只用到一阶导数信息，xgboost 则对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。顺便提一下，xgboost 工具支持自定义代价函数，只要函数可一阶和二阶求导。

3．xgboost 在代价函数里加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出的 score 的 L2 模的平方和。从 Bias-variance tradeoff 角度来讲，正则项降低了模型的 variance，使学习出来的模型更加简单，防止过拟合，这也是 xgboost 优于传统 GBDT 的一个特性。

4. Shrinkage（缩减），相当于学习速率（xgboost 中的 eta）。xgboost 在进行完一次迭代后，会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。实际应用中，一般把 eta 设置得小一点，然后迭代次数设置得大一点。（补充：传统 GBDT 的实现也有学习速率）

5. 列抽样（column subsampling）。xgboost 借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算，这也是 xgboost 异于传统 gbdt 的一个特性。

6. 对缺失值的处理。对于特征的值有缺失的样本，xgboost 可以自动学习出它的分裂方向。

7. xgboost 工具支持并行。boosting 不是一种串行的结构吗?怎么并行的？注意 xgboost 的并行不是 tree 粒度的并行，xgboost 也是一次迭代完才能进行下一次迭代的（第 t 次迭代的代价函数里包含了前面 t-1 次迭代的预测值）。xgboost 的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），xgboost 在训练之前，预先对数据进行了排序，然后保存为 block 结构，后面的迭代中重复地使用这个结构，大大减小计算量。这个 block 结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。

8. 可并行的近似直方图算法。树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以 xgboost 还提出了一种可并行的近似直方图算法，用于高效地生成候选的分割点。

Adaboost 和 gbdt 的区别是 adaboost 对于每个样本有一个权重，样本预估误差越大，权重越大。gradient boosting 则是直接用梯度拟合残差，没有样本权重的概念。

无监督和有监督算法的区别？

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

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

SVM 的推导，特性？多分类怎么处理？

SVM 是最大间隔分类器，几何间隔和样本的误分次数之间存在关系。
从线性可分情况下，原问题，特征转换后的 dual 问题，引入 kernel(线性 kernel,多项式，高斯),最后是 soft margin。
线性：简单，速度快，但是需要线性可分
多项式：比线性核拟合程度更强，知道具体的维度，但是高次容易出现数值不稳定，参数选择比较多。
高斯：拟合能力最强，但是要注意过拟合问题。不过只有一个参数需要调整。
多分类问题，一般将二分类推广到多分类的方式有三种，一对一，一对多，多对多。
一对一：将 N 个类别两两配对，产生 N(N-1)/2 个二分类任务，测试阶段新样本同时交给所有的分类器，最终结果通过投票产生。
一对多：每一次将一个例作为正例，其他的作为反例，训练 N 个分类器，测试时如果只有一个分类器预测为正类，则对应类别为最终结果，如果有多个，则一般选择置信度最大的。从分类器角度一对一更多，但是每一次都只用了 2 个类别，因此当类别数很多的时候一对一开销通常更小(只要训练复杂度高于 O(N)即可得到此结果)。

多对多：若干各类作为正类，若干个类作为反类。注意正反类必须特殊的设计

LR 的推导，特性？

决策树的特性？

决策树基于树结构进行决策，与人类在面临问题的时候处理机制十分类似。其特点在于需要选择一个属性进行分支，在分支的过程中选择信息增益最大的属性，在划分中我们希望决策树的分支节点所包含的样本属于同一类别，即节点的纯度越来越高。决策树计算量简单，可解释性强，比较适合处理有缺失属性值的样本，能够处理不相关的特征，但是容易过拟合，需要使用剪枝或者随机森林。信息增益是熵减去条件熵，代表信息不确定性较少的程度，信息增益越大，说明不确定性降低的越大，因此说明该特征对分类来说很重要。由于信息增益准则会对数目较多的属性有所偏好，因此一般用信息增益率(c4.5)　其中分母可以看作为属性自身的熵。取值可能性越多，属性的熵越大。Cart 决策树使用基尼指数来选择划分属性，直观的来说，Gini(D)反映了从数据集 D 中随机抽取两个样本，其类别标记不一致的概率，因此基尼指数越小数据集 D 的纯度越高，一般为了防止过拟合要进行剪枝，有预剪枝和后剪枝，一般用 cross validation 集进行剪枝。连续值和缺失值的处理，对于连续属性 a,将 a 在 D 上出现的不同的取值进行排序，基于划分点 t 将 D 分为两个子集。一般对每一个连续的两个取值的中点作为划分点，然后根据信息增益选择最大的。与离散属性不同，若当前节点划分属性为连续属性，该属性还可以作为其后代的划分属性。

SVM、LR、决策树的对比？

SVM 既可以用于分类问题，也可以用于回归问题，并且可以通过核函数快速的计算，LR 实现简单，训练速度非常快，但是模型较为简单，决策树容易过拟合，需要进行剪枝等。从优化函数上看，soft margin 的 SVM 用的是 hinge loss,而带 L2 正则化的 LR 对应的是 cross entropy loss，另外 adaboost 对应的是 exponential loss。所以 LR 对远点敏感，但是 SVM 对 outlier 不太敏感，因为只关心 support vector，SVM 可以将特征映射到无穷维空间，但是 LR 不可以，一般小数据中 SVM 比 LR 更优一点，但是 LR 可以预测概率，而 SVM 不可以，SVM 依赖于数据测度，需要先做归一化，LR 一般不需要，对于大量的数据 LR 使用更加广泛，LR 向多分类的扩展更加直接，对于类别不平衡 SVM 一般用权重解决，即目标函数中对正负样本代价函数不同，LR 可以用一般的方法，也可以直接对最后结果调整(通过阈值)，一般小数据下样本维度比较高的时候 SVM 效果要更优一些。SVM 通过映射到高维在做回归使用的。

GBDT 和 决策森林的区别？

随机森林采用的是 bagging 的思想，bagging 又称为 bootstrap aggreagation，通过在训练样本集中进行有放回的采样得到多个采样集，基于每个采样集训练出一个基学习器，再将基学习器结合。随机森林在对决策树进行 bagging 的基础上，在决策树的训练过程中引入了随机属性选择。传统决策树在选择划分属性的时候是在当前节点属性集合中选择最优属性，而随机森林则是对结点先随机选择包含 k 个属性的子集，再选择最有属性，k 作为一个参数控制了随机性的引入程度。
另外，GBDT 训练是基于 Boosting 思想，每一迭代中根据错误更新样本权重，因此是串行生成的序列化方法，而随机森林是 bagging 的思想，因此是并行化方法。

如何判断函数凸或非凸？

首先定义凸集，如果 x,y 属于某个集合 C,并且所有的$\theta x+(1-\theta)y$也属于 c,那么 c 为一个凸集，进一步，如果一个函数其定义域是凸集，并且

$$f(\theta x+(1-\theta)y)\leq\theta f(x)+(1-\theta)f(y)$$

则该函数为凸函数。上述条件还能推出更一般的结果，

$$f(\sum_{i=1}^{k}\theta_i x_i)\leq\sum_{i=1}^{k}\theta_i f(x_i),\sum_{i=1}^{k}\theta_i=1,\ \theta_i\geq 0$$

如果函数有二阶导数，那么如果函数二阶导数为正，或者对于多元函数，Hessian 矩阵半正定则为凸函数。

(也可能引到 SVM，或者凸函数局部最优也是全局最优的证明，或者上述公式期望情况下的 Jessen 不等式)

解释对偶的概念

一个优化问题可以从两个角度进行考察，一个是 primal 问题，一个是 dual 问题，就是对偶问题，一般情况下对偶问题给出主问题最优值的下界，在强对偶性成立的情况下由对偶问题可以得到主问题的最优下界，对偶问题是凸优化问题，可以进行较好的求解，SVM 中就是将 primal 问题转换为 dual 问题进行求解，从而进一步引入核函数的思想。

如何进行特征选择？

特征选择是一个重要的数据预处理过程，主要有两个原因，首先在现实任务中我们会遇到维数灾难的问题(样本密度非常稀疏)，若能从中选择一部分特征，那么这个问题能大大缓解，另外就是去除不相关特征会降低学习任务的难度，增加模型的泛化能力。冗余特征指该特征包含的信息可以从其他特征中推演出来，但是这并不代表该冗余特征一定没有作用，例如在欠拟合的情况下也可以用过加入冗余特征，增加简单模型的复杂度。
在理论上如果没有任何领域知识作为先验假设那么只能遍历所有可能的子集。但是这显然是不可能的，因为需要遍历的数量是组合爆炸的。一般我们分为子集搜索和子集评价两个过程，子集搜索一般采用贪心算法，每一轮从候选特征中添加或者删除，分别成为前向和后先搜索。或者两者结合的双向搜索。子集评价一般采用信息增益，对于连续数据往往排序之后选择中点作为分割点。
常见的特征选择方式有过滤式，包裹式和嵌入式，filter,wrapper 和 embedding。Filter 类型先对数据集进行特征选择，再训练学习器。Wrapper 直接把最终学习器的性能作为特征子集的评价准则，一般通过不断候选子集，然后利用 cross-validation 过程更新候选特征，通常计算量比较大。嵌入式特征选择将特征选择过程和训练过程融为了一体，在训练过程中自动进行了特征选择，例如 L1 正则化更易于获得稀疏解，而 L2 正则化更不容易过拟合。L1 正则化可以通过 PGD，近端梯度下降进行求解。

为什么会产生过拟合，有哪些方法可以预防或克服过拟合？

一般在机器学习中，将学习器在训练集上的误差称为训练误差或者经验误差，在新样本上的误差称为泛化误差。显然我们希望得到泛化误差小的学习器，但是我们事先并不知道新样本，因此实际上往往努力使经验误差最小化。然而，当学习器将训练样本学的太好的时候，往往可能把训练样本自身的特点当做了潜在样本具有的一般性质。这样就会导致泛化性能下降，称之为过拟合，相反，欠拟合一般指对训练样本的一般性质尚未学习好，在训练集上仍然有较大的误差。
欠拟合：一般来说欠拟合更容易解决一些，例如增加模型的复杂度，增加决策树中的分支，

增加神经网络中的训练次数等等。
过拟合：一般认为过拟合是无法彻底避免的，因为机器学习面临的问题一般是 np-hard,但是一个有效的解一定要在多项式内可以工作，所以会牺牲一些泛化能力。过拟合的解决方案一般有增加样本数量，对样本进行降维，降低模型复杂度，利用先验知识(L1,L2 正则化)，利用 cross-validation，early stopping 等等。

采用 EM 算法求解的模型有哪些，为什么不用牛顿法或梯度下降法？

用 EM 算法求解的模型一般有 GMM 或者协同过滤，k-means 其实也属于 EM。EM 算法一定会收敛，但是可能收敛到局部最优。由于求和的项数将随着**隐变量的数目指数上升**，会给梯度计算带来麻烦。

用 EM 算法推导解释 Kmeans

k-means 算法是高斯混合聚类在混合成分方差相等，且每个样本仅指派一个混合成分时候的特例。注意 k-means 在运行之前需要进行归一化处理，不然可能会因为样本在某些维度上过大导致距离计算失效。k-means 中每个样本所属的类就可以看成是一个隐变量，在 E 步中，我们固定每个类的中心，通过对每一个样本选择最近的类优化目标函数，在 M 步，重新更新每个类的中心点，该步骤可以通过对目标函数求导实现，最终可得新的类中心就是类中样本的均值。

常见聚类算法比较

(1) k-means
优点：简单，易于理解和实现；时间复杂度低，每轮迭代负载度为 O(n*k)
缺点： 需要对均值给出定义；需要指定聚类的数目；一些过大的异常值会带来很大影响；需要指定初始聚类中心，算法对初始值敏感；适合球形类簇。
(2) 层次聚类（试图在不同层次对数据集进行划分，从而形成树形的聚类结构。AGNES 是一种采用自底向上聚合策略的层次聚类算法）
优点：距离和规则的相似度容易定义，限制少；不需要预先指定聚类数目；可以发现类的层次关系；可以聚类成其他形状
缺点：计算复杂度高；奇异值也能产生很大影响；算法很可能聚类成链状
(3) 基于密度的聚类
(4) 基于网格的聚类
(5) 基于平方误差的迭代重分配聚类
(6) 基于约束的聚类

用过哪些聚类算法，解释密度聚类算法

k-means 算法，聚类性能的度量一般分为两类，一类是聚类结果与某个参考模型比较(外部指标)，另外是直接考察聚类结果(内部指标)。后者通常有 DB 指数和 DI，DB 指数是对每个类，找出类内平均距离/类间中心距离最大的类，然后计算上述值，并对所有的类求和，越小越好。类似 k-means 的算法仅在类中数据构成簇的情况下表现较好，密度聚类算法从样本密度的角度考察样本之间的可连接性，并基于可连接样本不断扩展聚类蔟得到最终结果。DBSCAN(density-based spatial clustering of applications with noise)是一种著名的密度聚类算法，基于一组邻域参数 $(\varepsilon, MinPts)$ 进行刻画，包括 ε 邻域，核心对象(邻域内至少包含 $MinPts$ 个对象)，密度直达(j 由 i 密度直达，表示 j 在 i 的邻域内，且 i 是一个核心对象)，密度可达(j 由 i 密度可达，存在样本序列使得每一对都密度直达)，密度相连(xi,xj 存在 k,i,j 均有 k 可达)，先找出样本中所有的核心对象，然后以任一核心对象作为出发点，找出由其密度可达的样本生成聚类簇，直到所有核心对象被访问过为止。

聚类算法中的距离度量有哪些

聚类算法中的距离度量一般用闽科夫斯基距离，在 p 取不同的值下对应不同的距离，例如 p=1 的时候对应曼哈顿距离，p=2 的情况下对应欧式距离，p=inf 的情况下变为切比雪夫距离，还有 jaccard 距离，幂距离(闽科夫斯基的更一般形式),余弦相似度，加权的距离，马氏距离(类似加权)作为距离度量需要满足非负性，同一性，对称性和直递性，闽科夫斯基在 p>=1 的时候满足读来那个性质，对于一些离散属性例如{飞机，火车，轮船}则不能直接在属性值上计算距离，这些称为无序属性，可以用 VDM(Value Diffrence Metrix)，属性 u 上两个离散值 a,b 之间的 VDM 距离定义为

$$VDM_p(a,b)=\sum_{i=1}^{k}\left|\frac{m_{u,a,i}}{m_{u,a}}-\frac{m_{u,b,i}}{m_{u,b}}\right|^p$$

其中表示在第 i 个簇中属性 u 上 a 的样本数，样本空间中不同属性的重要性不同的时候可以采用加权距离，一般如果认为所有属性重要性相同则要对特征进行归一化。一般来说距离需要的是相似性度量，距离越大，相似度越小，用于相似性度量的距离未必一定要满足距离度量的所有性质，例如直递性。比如人马和人，人马和马的距离较近，然后人和马的距离可能就很远。

如何进行实体识别？

解释贝叶斯公式和朴素贝叶斯分类求解方法

贝叶斯公式

$$P(c \mid x)=\frac{P(c, x)}{P(x)}=\frac{P(x \mid c) P(c)}{P(x)}$$

最小化分类错误的贝叶斯最优分类器等价于最大化后验概率

基于贝叶斯公式来估计后验概率的主要困难在于，条件概率 $P(x \mid c)$ 是所有属性上的联合概率，难以从有限的训练样本直接估计得到。朴素贝叶斯分类器采用了属性条件独立性假设，对于已知的类别，假设所有属性相互独立。这样，朴素贝叶斯分类则定义为

$$h_{nb}(x)=\arg \max P(c) \prod_{i=1}^{d} P(x_i \mid c)$$

如果有足够多的独立同分布样本，那么 $P(c)$ 可以根据每个类中的样本数量直接估计出来。

在离散情况下先验概率可以利用样本数量估计或者离散情况下根据假设的概率密度函数进行最大似然估计。朴素贝叶斯可以用于同时包含连续变量和离散变量的情况。如果直接基于出现的次数进行估计，会出现一项为 0 而乘积为 0 的情况，所以一般会用一些平滑的方法，例如拉普拉斯修正

频率学派和贝叶斯派的本质区别

使用随机事件的发生的频率描述概率的方法，就是通常说的古典概型，或者称为频率学派。另外有一个更加综合的观点就是贝叶斯学派，在贝叶斯学派的观点下概率表示的是事件的不确定性大小。

使用概率表示不确定性，虽然不是唯一的选择，但是是必然的，因为如果想使用比较自然的感觉进行合理的综合的推断的话。在模式识别领域，对概率有一个更综合的了解将会非常有帮助。例如在多项式曲线拟合的过程中，对观察的目标变量使用频率学派的观点来理解看起来比较合适。但是我们希望确定最佳模型的参数 w 的不确定性情况，于是我们可以看见在贝叶斯理论中不仅可以描述参数的不确定性，实际上选择模型本身也是不确定的

优化方法（随机梯度下降、拟牛顿法等优化算法）

两种算法都是通过对数据进行参数评估，然后进行调整，找到一组最小化损失函数的参数的方法。

在标准梯度下降中，您将评估每组参数的所有训练样本。这类似于为解决这个问题而采取

了大而缓慢的步骤。
在随机梯度下降中，在更新参数集之前，您只需评估 1 个训练样本。这类似于向解决方案迈出的小步骤。

特征比数据量还大时，选择什么样的分类器

如果训练集很小，那么高偏差/低方差分类器（如朴素贝叶斯分类器）要优于低偏差/高方差分类器（如 k 近邻分类器），因为后者容易过拟合。然而，随着训练集的增大，低偏差/高方差分类器将开始胜出（它们具有较低的渐近误差），因为高偏差分类器不足以提供准确的模型。你也可以认为这是生成模型与判别模型的区别。

L1 和 L2 正则的区别，如何选择 L1 和 L2 正则？L1 在 0 处不可导，怎么处理

他们都是可以防止过拟合，降低模型复杂度
L1 是在 loss function 后面加上 模型参数的 1 范数（也就是|xi|）L0 范数的最小化问题在实际应用中是 NP 难问题，无法实际应用
L2 是在 loss function 后面加上 模型参数的 2 范数（也就是 sigma(xi^2)），注意 L2 范数的定义是 sqrt(sigma(xi^2))，在正则项上没有添加 sqrt 根号是为了更加容易优化
L1 会产生稀疏的特征
L2 会产生更多地特征但是都会接近于 0
L1 会趋向于产生少量的特征，而其他的特征都是 0，而 L2 会选择更多的特征，这些特征都会接近于 0。L1 在特征选择时候非常有用，而 L2 就只是一种规则化而已。
L1 对应拉普拉斯分布，L2 对应高斯分布，L1 偏向于参数稀疏性，L1 不可导可以使用 Proximal Algorithms 或者 ADMM 来解决

随机森林的学习过程；随机森林中的每一棵树是如何学习的；随机森林学习算法中 CART 树的基尼指数是什么？

为什么一些机器学习模型需要对数据进行归一化？

http://blog.csdn.net/xbmatrix/article/details/56695825

归一化化就是要把你需要处理的数据经过处理后（通过某种算法）限制在你需要的一定范围内。
1）归一化后加快了梯度下降求最优解的速度。等高线变得显得圆滑，在梯度下降进行求解时能较快的收敛。如果不做归一化，梯度下降过程容易走之字，很难收敛甚至不能收敛
2）把有量纲表达式变为无量纲表达式，有可能提高精度。一些分类器需要计算样本之间的距离（如欧氏距离），例如 KNN。如果一个特征值域范围非常大，那么距离计算就主要取决于这个特征，从而与实际情况相悖（比如这时实际情况是值域范围小的特征更重要）
3) 逻辑回归等模型先验假设数据服从正态分布。
归一化的类型有线性归一化、标准差归一化、非线性归一化

归一化和标准化的区别？

归一化：
1）把数据变成(0， 1)之间的小数
2）把有量纲表达式变成无量纲表达
常见的有线性转换、对数函数转换、反余切函数转换等

标准化：
数据的标准化（normalization）是将数据按比例缩放，使之落入一个小的特定区间。在某些比较和评价的指标处理中经常会用到，去除数据的单位限制，将其转化为无量纲的纯数值，便于不同单位或量级的指标能够进行比较和加权。
1）最小－最大规范化(线性变换)
y=((x-MinValue) / (MaxValue-MinValue))(new_MaxValue-new_MinValue)+new_minValue
2）z-score 规范化(或零－均值规范化)
y=(x-X 的平均值) / X 的标准差
3）小数定标规范化：通过移动 X 的小数位置来进行规范化
y= x/10 的 j 次方　　(其中,j 使得 Max(|y|) <1 的最小整数
4）.对数 Logistic 模式：
新数据=1/（1+e^(-原数据)）
5）模糊量化模式
新数据=1/2+1/2sin[派 3.1415/（极大值-极小值）

特征向量的缺失值处理

1. 缺失值较多.直接将该特征舍弃掉，否则可能反倒会带入较大的 noise，对结果造成不良影响。
2. 缺失值较少,其余的特征缺失值都在 10%以内，我们可以采取很多的方式来处理:
1) 把 NaN 直接作为一个特征，假设用 0 表示；
2) 用均值填充；
3) 用随机森林等算法预测填充

决策树的停止条件

直到每个叶子节点都只有一种类型的记录时停止，(这种方式很容易过拟合)
另一种时当叶子节点的记录树小于一定的阈值或者节点的信息增益小于一定的阈值时停止

SVM、LR、决策树的对比？

模型复杂度：SVM 支持核函数，可处理线性非线性问题;LR 模型简单，训练速度快，适合处理线性问题;决策树容易过拟合，需要进行剪枝
损失函数：SVM hinge loss; LR L2 正则化; adaboost 指数损失
数据敏感度：SVM 添加容忍度对 outlier 不敏感，只关心支持向量，且需要先做归一化; LR 对远点敏感
数据量：数据量大就用 LR，数据量小且特征少就用 SVM 非线性核

GBDT 和随机森林的区别？

随机森林采用的是 bagging 的思想，bagging 又称为 bootstrap aggreagation，通过在训练样本集中进行有放回的采样得到多个采样集，基于每个采样集训练出一个基学习器，再将基学习器结合。随机森林在对决策树进行 bagging 的基础上，在决策树的训练过程中引入了随机属性选择。传统决策树在选择划分属性的时候是在当前节点属性集合中选择最优属性，而随机森林则是对结点先随机选择包含 k 个属性的子集，再选择最有属性，k 作为一个参数控制了随机性的引入程度。
另外，GBDT 训练是基于 Boosting 思想，每一迭代中根据错误更新样本权重，因此是串行生成的序列化方法，而随机森林是 bagging 的思想，因此是并行化方法。

监督学习一般使用两种类型的目标变量

标称型和数值型
标称型：标称型目标变量的结果只在有限目标集中取值，如真与假(标称型目标变量主要用于分类)
数值型：数值型目标变量则可以从无限的数值集合中取值，如 0.100，42.001 等 (数值型目标变量主要用于回归分析)

为什么说朴素贝叶斯是高偏差低方差？

它简单的假设了各个特征之间是无关的，是一个被严重简化了的模型。所以，对于这样一个简单模型，大部分场合都会 bias 部分大于 variance 部分，也就是高偏差，低方差

决策树的父节点和子节点的熵的大小？请解释原因。

父节点的熵>子节点的熵

最好是在项目/实习的大数据场景里用过，比如推荐里用过 CF、LR，分类里用过 SVM、GBDT；一般用法是什么，是不是自己实现的，有什么比较知名的实现，使用过程中踩过哪些坑；

KMeans 怎么选择聚类中心？如果存在空块怎么办？
线性 SVM 算法的基本原理？核函数干什么用的？
如何进行实体识别？
常见优化算法的比较，看微信收藏的？最优化里面的
讲讲 DBN 模型
损失函数如何去选择
出了 sigmoid 函数还了解哪些，其他的激活函数
神经网络有哪些优化算法？
概率论：均匀分布如何转变为高斯分布？
如何理解白噪声？图像的白噪声是如何形成的？
指针和结构体的区别？使用指针需要注意什么？
谈谈内存泄露的避免方法？
类的大小计算。C 语言中变量存储为什么要内存对齐？
STL 相关的问题：vector 增长；map 是如何实现的？(红黑树)红黑树是什么？有什么性质？红黑树与 AVL 树的区别？
在函数传参时候正常对象，指针和引用分别如何使用，并简述他们之间的区别。
main 函数当中变量，全局变量还有宏，他们的执行顺序是什么样子的？
C++的三个特性，如何理解多态？虚函数是如何实现的？
链表查找环；判断两个单链表是否相交。
二叉镜像树。
TCP 和 UDP 哪些不同？
C++的多继承的缺点
两个编程题，判断单链表有没有环，并输出第一个环节点；找两个链表的公共节点。

进程和线程的区别有哪些？进程/线程间如何通信？如何防止线程死锁？TCP 三次握手四次挥手是个什么步骤？

给你公司内部群组的聊天记录，怎样区分出主管和员工？

如何评估网站内容的真实性（针对代刷、作弊类）？

深度学习在推荐系统上可能有怎样的发挥？

路段平均车速反映了路况，在道路上布控采集车辆速度，如何对路况做出合理估计？采集数据中的异常值如何处理？

如何根据语料计算两个词词义的相似度？

在百度贴吧里发布 APP 广告，问推荐策略？

如何判断自己实现的 LR、Kmeans 算法是否正确？

100 亿数字，怎么统计前 100 大的？

常见的聚类算法？

缺失数据的处理？

Bagging 和 Boosting 的区别？

校正 R2 或者 F 值是用来评估线性回归模型的。那用什么来评估逻辑回归模型？

1.由于逻辑回归是用来预测概率的，我们可以用 AUC-ROC 曲线以及混淆矩阵来确定其性能。

2.此外，在逻辑回归中类似于校正 R2 的指标是 AIC。AIC 是对模型系数数量惩罚模型的拟合度量。因此，我们更偏爱有最小 AIC 的模型。

3.空偏差指的是只有截距项的模型预测的响应。数值越低，模型越好。残余偏差表示由添加自变量的模型预测的响应。数值越低，模型越好。

推荐系统：

推荐系统的实现主要分为两个方面：基于内容的实现和协同滤波的实现。

基于内容的实现：

不同人对不同电影的评分这个例子，可以看做是一个普通的回归问题，因此每部电影都需要提前提取出一个特征向量(即 x 值)，然后针对每个用户建模，即每个用户打的分值作为 y 值，利用这些已有的分值 y 和电影特征值 x 就可以训

练回归模型了(最常见的就是线性回归)。这样就可以预测那些用户没有评分的电影的分数。（值得注意的是需对每个用户都建立他自己的回归模型）

从另一个角度来看，也可以是先给定每个用户对某种电影的喜好程度（即权值），然后学出每部电影的特征，最后采用回归来预测那些没有被评分的电影。

当然还可以是同时优化得到每个用户对不同类型电影的热爱程度以及每部电影的特征。具体可以参考 Ng 在 coursera 上的 ml 教程：

基于协同滤波的实现：

协同滤波（CF）可以看做是一个分类问题，也可以看做是矩阵分解问题。协同滤波主要是基于每个人自己的喜好都类似这一特征，它不依赖于个人的基本信息。比如刚刚那个电影评分的例子中，预测那些没有被评分的电影的分数只依赖于已经打分的那些分数，并不需要去学习那些电影的特征。

SVD 将矩阵分解为三个矩阵的乘积，公式如下所示：

$$Data_{mxn} = U_{mxm}\Sigma_{mxn}V^{T}_{nxn}$$

中间的矩阵 sigma 为对角矩阵，对角元素的值为 Data 矩阵的奇异值(注意奇异值和特征值是不同的)，且已经从大到小排列好了。即使去掉特征值小的那些特征，依然可以很好的重构出原始矩阵。如下图所示：

其中更深的颜色代表去掉小特征值重构时的三个矩阵。

果 m 代表商品的个数，n 代表用户的个数，则 U 矩阵的每一行代表商品的属性，现在通过降维 U 矩阵（取深色部分）后，每一个商品的属性可以用更低的维度表示（假设为 k 维）。这样当新来一个用户的商品推荐向量 X，则可以根据公式 X’*U1*inv(S1)得到一个 k 维的向量，然后在 V’中寻找最相似的那一个用户（相似度测量可用余弦公式等），根据这个用户的评分来推荐（主要是推荐新用户未打分的那些商品）。具体例子可以参考网页：SVD 在推荐系统中的应用。另外关于 SVD 分解后每个矩阵的实际含义可以参考 google 吴军的《数学之美》一书（不过个人感觉吴军解释 UV 两个矩阵时好像弄反了，不知道大家怎样认为）。或者参考 machine learning in action 其中的 svd 章节。

TF-IDF 是什么？

TF 指 Term frequecy,代表词频,IDF 代表 inverse document frequency,叫做逆

文档频率，这个算法可以用来提取文档的关键词，首先一般认为在文章中出现次数较多的词是关键词，词频就代表了这一项，然而有些词是停用词，例如的，是，有这种大量出现的词，首先需要进行过滤，比如过滤之后再统计词频出现了中国，蜜蜂，养殖且三个词的词频几乎一致，但是中国这个词出现在其他文章的概率比其他两个词要高不少，因此我们应该认为后两个词更能表现文章的主题，IDF 就代表了这样的信息，计算该值需要一个语料库，如果一个词在语料库中出现的概率越小，那么该词的 IDF 应该越大，一般来说 TF 计算公式为(某个词在文章中出现次数/文章的总词数)，这样消除长文章中词出现次数多的影响，IDF 计算公式为 **log(语料库文章总数/(包含该词的文章数)+1)**。将两者乘乘起来就得到了词的 TF-IDF。传统的 TF-IDF 对词出现的位置没有进行考虑，可以针对不同位置赋予不同的权重进行修正，注意这些修正之所以是有效的，正是因为人观测过了大量的信息，因此建议了一个先验估计，人将这个先验估计融合到了算法里面，所以使算法更加的有效。

文本中的余弦距离是什么，有哪些作用？

余弦距离是两个向量的距离的一种度量方式，其值在-1~1 之间，如果为 1 表示两个向量同相，0 表示两个向量正交，-1 表示两个向量反向。使用 TF-IDF 和余弦距离可以寻找内容相似的文章，例如首先用 TF-IDF 找出两篇文章的关键词，然后每个文章分别取出 k 个关键词(10-20 个)，统计这些关键词的词频，生成两篇文章的词频向量，然后用余弦距离计算其相似度。

如何解决类别不平衡问题？

2. 处理不平衡数据集的方法

2.1.1 随机欠采样（Random Under-Sampling）

2.1.2 随机过采样（Random Over-Sampling）

2.1.3 基于聚类的过采样（Cluster-Based Over Sampling）

在这种情况下，K-均值聚类算法独立地被用于少数和多数类实例。这是为了识别数据集中的聚类。随后，每一个聚类都被过采样以至于相同类的所有聚类有着同样的实例数量，且所有的类有着相同的大小。

2.1.4 信息性过采样：合成少数类过采样技术（SMOTE）

这一技术可用来避免过拟合——当直接复制少数类实例并将其添加到主数据集时。从少数类中把一个数据子集作为一个实例取走，接着创建相似的新合成的实例。这些合成的实例接着被添加进原来的数据集。新数据集被用作样本以训练分类模型。

2.15 改进的合成少数类过采样技术（MSMOTE）

2.2 算法集成技术（Algorithmic Ensemble Techniques）

Bagging boosting

什么是贝叶斯估计

new 和 malloc 的区别

hash 冲突是指什么？怎么解决？给两种方法，写出过程和优缺点。

是否了解线性加权、bagging、boosting、cascade 等模型融合方式

LDA 的原理和推导

做广告点击率预测，用哪些数据什么算法

推荐系统的算法中最近邻和矩阵分解各自适用场景

用户流失率预测怎么做（游戏公司的数据挖掘都喜欢问这个）

一个游戏的设计过程中该收集什么数据

如何从登陆日志中挖掘尽可能多的信息

推荐系统的冷启动问题如何解决

是否了解 A/B Test 以及 A/B Test 结果的置信度

给你一个有 1000 列和 1 百万行的训练数据集。这个数据集是基于分类问题的。经理要求你来降低该数据集的维度以减少模型计算时间。你的机器内存有限。你会怎么做？（你可以自由做各种实际操作假设。）

1.由于我们的 RAM 很小，首先要关闭机器上正在运行的其他程序，包括网页浏览器，以确保大部分内存可以使用。

2.我们可以随机采样数据集。这意味着，我们可以创建一个较小的数据集，比如有 1000 个变量和 30 万行，然后做计算。

3.为了降低维度，我们可以把数值变量和分类变量分开，同时删掉相关联的变量。对于数值变量，我们将使用相关性分析。对于分类变量，我们可以用卡方检验。

4.另外，我们还可以使用 PCA（主成分分析），并挑选可以解释在数据集中有最大偏差的成分。

5.利用在线学习算法，如 VowpalWabbit（在 Python 中可用）是一个可能的选择。

6.利用 Stochastic GradientDescent（随机梯度下降）法建立线性模型也很有帮助。

7.我们也可以用我们对业务的理解来估计各预测变量对响应变量的影响大小。但是，这是一个主观的方法，如果没有找出有用的预测变量可能会导致信息的显著丢失。

在 PCA 中有必要做旋转变换吗？如果有必要，为什么？如果你没有旋转变换那些成分，会发生什么情况？

旋转（正交）是必要的，因为它把由主成分捕获的方差之间的差异最大化。这使得主成分更容易解释。

但是不要忘记我们做 PCA 的目的是选择更少的主成分（与特征变量个数相较而言），那些选上的主成分能够解释数据集中最大方差。通过做旋转，各主成分的相对位置不发生变化，它只能改变点的实际坐标。

如果我们没有旋转主成分，PCA 的效果会减弱，那样我们会不得不选择更多个主成分来解释数据集里的方差。

给你一个数据集。这个数据集有缺失值，且这些缺失值分布在离中值有 1 个标准偏差的范围内。百分之多少的数据不会受到影响？为什么？

这个问题给了你足够的提示来开始思考！由于数据分布在中位数附近，让我们先假设这是一

个正态分布。我们知道，在一个正态分布中，约有 68%的数据位于跟平均数（或众数、中位数）1 个标准差范围内的，那样剩下的约 32%的数据是不受影响的。因此，约有 32%的数据将不受到缺失值的影响。

给你一个癌症检测的数据集。你已经建好了分类模型，取得了 96%的精度。为什么你还是不满意你的模型性能？你可以做些什么呢？

如果你分析过足够多的数据集，你应该可以判断出来癌症检测结果是不平衡数据。在不平衡数据集中，精度不应该被用来作为衡量模型的标准，因为 96%（按给定的）可能只有正确预测多数分类，但我们感兴趣是那些少数分类（4%），是那些被诊断出癌症的人。
因此，为了评价模型的性能，应该用灵敏度（真阳性率），特异性（真阴性率），F 值用来确定这个分类器的 “聪明” 程度。如果在那 4%的数据上表现不好，我们可以采取以下步骤：
1.我们可以使用欠采样、过采样或 SMOTE 让数据平衡。
2.我们可以通过概率验证和利用 AUC-ROC 曲线找到最佳阀值来调整预测阀值。
3.我们可以给分类分配权重，那样较少的分类获得较大的权重。
4.我们还可以使用异常检测。

解释朴素贝叶斯算法里面的先验概率、似然估计和边际似然估计？

先验概率就是因变量（二分法）在数据集中的比例。这是在你没有任何进一步的信息的时候，是对分类能做出的最接近的猜测。例如，在一个数据集中，因变量是二进制的（1 和 0）。例如，1（垃圾邮件）的比例为 70%和 0（非垃圾邮件）的为 30%。

因此，我们可以估算出任何新的电子邮件有 70%的概率被归类为垃圾邮件。似然估计是在其他一些变量的给定的情况下，一个观测值被分类为 1 的概率。例如，“FREE”这个词在以前的垃圾邮件使用的概率就是似然估计。边际似然估计就是，“FREE”这个词在任何消息中使用的概率。

花了几个小时后，现在你急于建一个高精度的模型。结果，你建了 5 个 GBM （Gradient Boosted Models），想着 boosting 算法会显示魔力。不幸的是，没有一个模型比基准模型表现得更好。最后，你决定将这些模型结合到一起。尽管众所周知，结合模型通常精度高，但你就很不幸运。你到底错在哪里？

据我们所知，组合的学习模型是基于合并弱的学习模型来创造一个强大的学习模型的想法。但是，只有当各模型之间没有相关性的时候组合起来后才比较强大。由于我们已经试了 5 个 GBM，但没有提高精度，表明这些模型是相关的。具有相关性的模型的问题是，所有的模型提供相同的信息。例如：如果模型 1 把 User1122 归类为 1，模型 2 和模型 3 很有可能会做有同样分类，即使它的实际值应该是 0，因此，只有弱相关的模型结合起来才会表现更好。

我知道校正 R？或者 F 值来是用来评估线性回归模型的。那用什么来评估逻辑回归模型？

1.由于逻辑回归是用来预测概率的，我们可以用 AUC-ROC 曲线以及混淆矩阵来确定其性能。

2.此外，在逻辑回归中类似于校正 R？的指标是 AIC。AIC 是对模型系数数量惩罚模型的拟合度量。因此，我们更偏爱有最小 AIC 的模型。

3.空偏差指的是只有截距项的模型预测的响应。数值越低，模型越好。残余偏差表示由添加自变量的模型预测的响应。数值越低，模型越好。

偏差和方差的平衡？

偏差误差在量化平均水平之上预测值跟实际值相差多远时有用。高偏差误差意味着我们的模型表现不太好，因为没有抓到重要的趋势。
而另一方面，方差量化了在同一个观察上进行的预测是如何彼此不同的。高方差模型会过度拟合你的训练集，而在训练集以外的数据上表现很差。

介绍卷积神经网络，和 DBN 有什么区别？

卷积神经网络的特点是卷积核，CNN 中使用了权共享，通过不断的上采用和卷积得到不同的特征表示，采样层又称为 pooling 层，基于局部相关性原理进行亚采样，在减少数据量的同时保持有用的信息。DBN 是深度信念网络，每一层是一个 RBM，整个网络可以视为 RBM 堆叠得到，通常使用无监督逐层训练，从第一层开始，每一层利用上一层的输入进行训练，等各层训练结束之后再利用 BP 算法对整个网络进行训练。

LSTM 结构推导，为什么比 RNN 好？

推导 forget gate，input gate，cell state， hidden information 等的变化；因为 LSTM 有进有出且当前的 cell informaton 是通过 input gate 控制之后叠加的，RNN 是叠乘，因此 LSTM 可以防止梯度消失或者爆炸；

为什么很多做人脸的 Paper 会最后加入一个 Local Connected Conv？

人脸在不同的区域存在不同的特征（眼睛 / 鼻子 / 嘴的分布位置相对固定），当不存在全局的局部特征分布时，Local-Conv 更适合特征的提取。

什么样的资料集不适合用深度学习?

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

对所有优化问题来说，有没有可能找到比現在已知算法更好的算法

没有免费的午餐定理

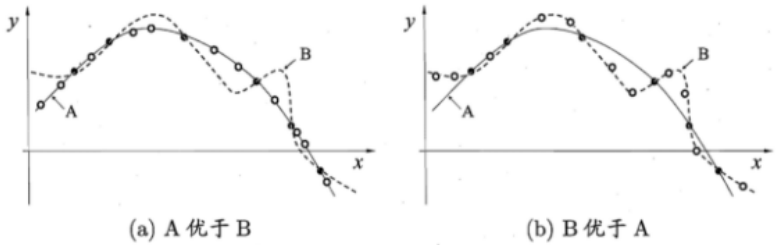


图 1.4 没有免费的午餐. (黑点: 训练样本; 白点: 测试样本)

对于训练样本（黑点），不同的算法 A/B 在不同的测试样本（白点）中有不同的表现，这表示：对于一个学习算法 A，若它在某些问题上比学习算法 B 更好，则必然存在一些问题，在那里 B 比 A 好。
也就是说：对于所有问题，无论学习算法 A 多聪明，学习算法 B 多笨拙，它们的期望性能相同。
但是：没有免费午餐定力假设所有问题出现几率相同，实际应用中，不同的场景，会有不同的问题分布，所以，在优化算法时，针对具体问题进行分析，是算法优化的核心所在。

用贝叶斯机率说明 Dropout 的原理

Dropout 是一种模型选择技术，其旨在避免在训练过程中出现过拟合现象，Dropout 的基本做法是在给定概率 p 的情况下随机移除输入数据 X 的维度。因此，探讨一下其如何影响潜在损失函数和最优化问题是有启发性的。

何为共线性，跟过拟合有啥关联?

共线性：多变量线性回归中，变量之间由于存在高度相关关系而使回归估计不准确。
共线性会造成冗余，导致过拟合。
解决方法：排除变量的相关性 / 加入权重正则

说明如何用支持向量机实现深度学习

支持向量机的原理大致是由感知器得到的，可以认为是神经网络的一个特例。

广义线性模型是怎被应用在深度学习中

1. 深度学习从统计学角度，可以看做递归的广义线性模型。
2. 广义线性模型相对于经典的线性模型(y=wx+b)，核心在于引入了连接函数 g(.)，形式变为：y=g−1(wx+b)。
3. 深度学习时递归的广义线性模型，神经元的激活函数，即为广义线性模型的链接函数。逻辑回归（广义线性模型的一种）的 Logistic 函数即为神经元激活函数中的 Sigmoid 函数，很多类似的方法在统计学和神经网络中的名称不一样，容易引起初学者（这里主要指我）的困惑。下图是一个对照表：

什么造成梯度消失问题？推导一下

1. 神经网络的训练中，通过改变神经元的权重，使网络的输出值尽可能逼近标签以降低误差值，训练普遍使用 BP 算法，核心思想是，计算出输出与标签间的损失函数值，然后计算其相对于每个神经元的梯度，进行权值的迭代。
2. 梯度消失会造成权值更新缓慢，模型训练难度增加。造成梯度消失的一个原因是，许多激活函数将输出值挤压在很小的区间内，在激活函数两端较大范围的定义域内梯度为 0。造成学习停止

$$a_1 = \sigma(z_1) = \sigma(w_1 a_0 + b_1)$$

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)\, w_2 \sigma'(z_2)\, w_3 \sigma'(z_3)\, w_4 \sigma'(z_4)\, \frac{\partial C}{\partial a_4}$$

Weights Initialization. 不同的方式，造成的后果。为什么会造成这样的结果。

权重初始化为 0，造成无法学习
lecun_uniform / glorot_normal / he_normal / batch_normal

神经网络过度拟合、规范化？

正则化、DroupOut

梯度爆炸

神经网络后向传播的时候导致求导的值很大，累乘之后很大，导致最后的梯度很大。为防止梯度爆炸，一种方式是设置梯度剪切阈值 gradient_clipping_threshold，一旦梯度超过改值，直接置为该值。

神经网络损失函数？

输出具体的类别标签时：
使用 sigmod 激活函数时
1. 平方误差
2. 交叉熵
使用 softmax 激活函数时
用 log-likehood
在激活函数使用 sigmoid 的前提之下，相比于 quadratic cost function， cross entropy cost function 具有收敛速度快和更容易获得全局最优（至于为什么更容易获得全局最优，个人感觉有点类似于动量的物理意义，增加收敛的步长，加快收敛的速度，更容易跳过局部最优）的特点。
因为我们一般使用随机值来初始化权重，这就可能导致一部分期望值和预测值相差甚远。所以选择 sigmoid 作为激活函数的时候，推荐使用 cross entropy。如果激活函数不是 sigmoid，quadratic cost function 就不会存在收敛速度慢的问题。
对于分类问题，如果希望输出是类别的概率，那么激活函数选择使用 softmax，同时使用 log-likelihood 作为损失函数。

Dropout 怎么做，有什么用处，解释

可以通过阻止某些特征的协同作用来缓解。在每次训练的时候，每个神经元有百分之 50 的几率被移除，这样可以让一个神经元的出现不应该依赖于另外一个神经元。另外，我们可以把 dropout 理解为 模型平均。假设我们要实现一个图片分类任务，我们设计出了 100000 个网络，这 100000 个网络，我们可以设计得各不相同，然后我们对这 100000 个网络进行训练，训练完后我们采用平均的方法，进行预测，这样肯定可以提高网络的泛化能力，或者说可以防止过拟合，因为这 100000 个网络，它们各不相同，可以提高网络的稳定性。而所谓的 dropout 我们可以这么理解，这 n 个网络，它们权值共享，并且具有相同的网络层数(这样可以大大减小计算量)。我们每次 dropout 后，网络模型都可以看成是整个网络的子网络。(需要注意的是如果采用 dropout，训练时间大大延长，但是对测试阶段没影响)。
Dropout 说的简单一点就是我们让在前向传导的时候，让某个神经元的激活值以一定的概率 p，让其停止工作

Dropout 单元的数理？

Bathsize 的影响？

神经网络交叉验证？

神经网络归一化

数值问题
不归一化容易引起数值问题
求解需要
在训练前我们将数据归一化，说明数据归是为了更方便的求解。

CNN 参数调节

学习率
学习率是指在优化算法中更新网络权重的幅度大小。学习率可以是恒定的、逐渐降低的、基于动量的或者是自适应的，采用哪种学习率取决于所选择优化算法的类型，如 SGD、Adam、Adagrad、AdaDelta 或 RMSProp 等算法。优化策略这方面的内容可参阅量子位之前编译过的 “一文看懂各种神经网络优化算法：从梯度下降到 Adam 方法” 。

迭代次数
迭代次数是指整个训练集输入到神经网络进行训练的次数。当测试错误率和训练错误率相差

较小时，可认为当前的迭代次数是合适的，否则需继续增大迭代次数，或调整网络结构。

批次大小
在卷积神经网络的学习过程中，小批次会表现得更好，选取范围一般位于区间[16,128]内。还需要注意的是，CNN 网络对批次大小的调整十分敏感。

激活函数
激活函数具有非线性，理论上可以使模型拟合出任何函数。通常情况下，rectifier 函数在 CNN 网络中的效果较好。当然，可以根据实际任务，选择其他类型的激活函数，如 Sigmoid 和 Tanh 等等。

隐含层的数目和单元数
增加隐含层数目以加深网络深度，会在一定程度上改善网络性能，但是当测试错误率不再下降时，就需要寻求其他的改良方法。增加隐含层数目也带来一个问题，即提高了训练该网络的计算成本。

当网络的单元数设置过少时，可能会导致欠拟合，而单元数设置过多时，只要采取合适的正则化方式，就不会产生不良影响。

权重初始化
在网络中，通常会使用小随机数来初始化各网络层的权重，以防止产生不活跃的神经元，但是设置过小的随机数可能生成零梯度网络。一般来说，均匀分布方法效果较好。

Dropout 方法
作为一种常用的正则化方式，加入 Dropout 层可以减弱深层神经网络的过拟合效应。该方法会按照所设定的概率参数，在每次训练中随机地不激活一定比例的神经单元。该参数的默认值为 0.5。

手动调整超参数是十分费时也不切实际。接下来介绍两种搜索最优超参数的常用方法。

网格搜索和随机搜索
网格搜索是通过穷举法列出不同的参数组合，确定性能最优的结构。随机搜索是从具有特定分布的参数空间中抽取出一定数量的候选组合。

网格搜索方法也需要制定策略，在初始阶段最好先确定各超参数值的大概范围。可以先尝试在较小迭代次数或较小规模的训练集上进行大步幅的网格搜索。然后在下个阶段中，设置更大的迭代次数，或是使用整个训练集，实现小幅精确定位。

虽然在许多机器学习算法中，通常会使用网格搜索来确定超参数组合，但是随着参数量的增大，训练网络所需的计算量呈指数型增长，这种方法在深层神经网络的超参数调整时效果并不是很好。

有研究指出，在深度神经网络的超参数调整中，随机搜索方法比网格搜索的效率更高，具体可参考文末中的“随机搜索在超参数优化中的应用”。

当然，可根据神经网络的理论经验，进行超参数的手动调整在一些场景下也是可行的。

可视化
我们可以通过可视化各个卷积层，来更好地了解 CNN 网络是如何学习输入图像的特征。可视化有两种直接方式，分别是可视化激活程度和可视化相关权重。在网络训练过程中，卷积层的激活情况通常会变得更为稀疏和具有局部特性。当不同输入图像的激活图都存在大片未激活的区域，那么可能是设置了过高的学习率使得卷积核不起作用，导致产生零激活图像。性能优良的神经网络通常含有多个明显而平滑的卷积器，且没有任何干扰特征。若在权重中观察到相关干扰特征，可能原因是网络未被充分训练，或是正则化强度较低导致了过拟合效

CNN 池化层的作用？

1. 不变性，旋转位移不变性，更关注是否存在某些特征而不是特征具体的位置。可以看作加了一个很强的先验，让学到的特征要能容忍一些的变化。
2. 减小下一层输入大小，减小计算量和参数个数。
3. 获得定长输出。（文本分类的时候输入是不定长的，可以通过池化获得定长输出）
4. 防止过拟合或有可能会带来欠拟合。保留主要的特征同时减少参数(降维，效果类似 PCA)和计算量，防止过拟合，提高模型泛化能力

神经网络层数的影响

有文章研究过层数的影响，He 在 2015 年的研究表明，如果神经网络的层数很深，那么可以减少卷积核心得数量，也是可以的。更深的层，抽象能力会更好，但是按照普通的来说，五个卷积层已经足够使用了，已经可以学习到很好的图像表达了，在卷积神经网络中，也分析了每一个卷积层表达的具体抽象。说明了 Alexnet 当中其实已经学习到了蛮不错的表达。

神经网络局部最优

局部最优其实不是神经网络的问题，在一个非常高维的空间中做梯度下降，这时的 local minimum 是很难形成的，因为局部最小值要求函数在所有维度上都是局部最小的。实际情况是，函数会落在一个 saddle-point 上。

在 saddle-point 上会有一大片很平坦的平原，让梯度几乎为 0，导致无法继续下降。

但是 saddle-point 并不是一个局部极小值点，因为它还是有可以下降的方向，只不过现在这些优化算法都很难去找到这个方向罢了。

神经网络为什么要激活

卷积神将网络中，反向传播在池化层和卷积乘如何实现？

要套用 DNN 的反向传播算法到 CNN，有几个问题需要解决：

1）池化层没有激活函数，这个问题倒比较好解决，我们可以令池化层的激活函数为 σ(z)=z，即激活后就是自己本身。这样池化层激活函数的导数为 1.

2）池化层在前向传播的时候，对输入进行了压缩，那么我们现在需要向前反向推导 δl−1，这个推导方法和 DNN 完全不同。

3) 卷积层是通过张量卷积，或者说若干个矩阵卷积求和而得的当前层的输出，这和 DNN 很不相同，DNN 的全连接层是直接进行矩阵乘法得到当前层的输出。这样在卷积层反向传播的时候，上一层的 δl−1 递推计算方法肯定有所不同。

4）对于卷积层，由于 W 使用的运算是卷积，那么从 δl 推导出该层的所有卷积核的 W,b 的方式也不同。

从上面可以看出，问题 1 比较好解决，但是问题 2,3,4 就需要好好的动一番脑筋了，而问题 2,3,4 也是解决 CNN 反向传播算法的关键所在。另外大家要注意到的是，DNN 中的 al,zl 都只是一个向量，而我们 CNN 中的 al,zl 都是一个张量，这个张量是三维的，即由若干个输入的子矩阵组成。

下面我们就针对问题 2,3,4 来一步步研究 CNN 的反向传播算法。

在研究过程中，需要注意的是，由于卷积层可以有多个卷积核，各个卷积核的处理方法是完全相同且独立的，为了简化算法公式的复杂度，我们下面提到卷积核都是卷积层中若干卷积核中的一个。

已知池化层的δl，推导上一隐藏层的 δl−1

已知卷积层的δl，推导上一隐藏层的 δl−1

已知卷积层的δl，推导该层的 W,b 的梯度

Tensorflow 如何看模型评估

可以使用 tensorboard 来看一系列的过程，模型的评估主要有几个指标：平均准确率、识别的时间、loss 下降变化等

直方图均衡化

采用累积分布函数对像素进行变换

$$s_k = \sum_{j=0}^{k} \frac{n_j}{n} \quad k = 0, 1, 2, \cdots, L-1$$

处理海量数据问题之六把密匙

1. 分而治之/hash 映射 + hash 统计 + 堆/快速/归并排序；
2. 双层桶划分 ---**第 k 大，中位数，不重复或重复的数字**
3. Bloom filter/Bitmap --实现数据字典，进行数据的判重，或者集合求交集
4. Trie 树/数据库/倒排索引
5. 外排序
6. 分布式处理之 Hadoop/Mapreduce

密匙一、分而治之/Hash 映射 + Hash_map 统计 + 堆/快速/归并排序

1、海量日志数据，提取出某日访问百度次数最多的那个 IP。

既然是海量数据处理，那么可想而知，给我们的数据那就一定是海量的。针对这个数据的海量，我们如何着手呢?对的，无非就是分而治之/hash 映射 + hash 统计 + 堆/快速/归并排序，说白了，就是先映射，而后统计，最后排序：

1. 分而治之/hash 映射：针对数据太大，内存受限，只能是：把大文件化成(取模映射)小文件，即 16 字方针：大而化小，各个击破，缩小规模，逐个解决

2. hash_map 统计：当大文件转化了小文件，那么我们便可以采用常规的 hash_map(ip，value)来进行频率统计。
3. 堆/快速排序：统计完了之后，便进行排序(可采取堆排序)，得到次数最多的 IP。

具体而论，则是：“首先是这一天，并且是访问百度的日志中的 IP 取出来，逐个写入到一个大文件中。注意到 IP 是 32 位的，最多有个 2^32 个 IP。同样可以采用映射的方法，比如%1000，把整个大文件映射为 1000 个小文件，再找出每个小文中出现频率最大的 IP（可以采用 hash_map 对那 1000 个文件中的所有 IP 进行频率统计，然后依次找出各个文件中频率最大的那个 IP）及相应的频率。然后再在这 1000 个最大的 IP 中，找出那个频率最大的 IP，即为所求。”--十道海量数据处理面试题与十个方法大总结。

关于本题，还有几个问题，如下：

1、Hash 取模是一种等价映射，不会存在同一个元素分散到不同小文件中的情况，即这里采用的是 mod1000 算法，那么相同的 IP 在 hash 取模后，只可能落在同一个文件中，不可能被分散的。因为如果两个 IP 相等，那么经过 Hash(IP)之后的哈希值是相同的，将此哈希值取模（如模 1000），必定仍然相等。

2、那到底什么是 hash 映射呢？简单来说，就是为了便于计算机在有限的内存中处理 big 数据，从而通过一种映射散列的方式让数据均匀分布在对应的内存位置(如大数据通过取余的方式映射成小树存放在内存中，或大文件映射成多个小文件)，而这个映射散列方式便是我们通常所说的 hash 函数，设计的好的 hash 函数能让数据均匀分布而减少冲突。尽管数据映射到了另外一些不同的位置，但数据还是原来的数据，只是代替和表示这些原始数据的形式发生了变化而已。

OK，有兴趣的，还可以再了解下一致性 hash 算法，见 blog 内此文第五部分：

http://blog.csdn.net/v_july_v/article/details/6879101。

2、寻找热门查询，300 万个查询字符串中统计最热门的 10 个查询

原题：搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门），请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

解答：由上面第 1 题，我们知道，数据大则划为小的，如如一亿个 Ip 求 Top 10，可先%1000 将 ip 分到 1000 个小文件中去，并保证一种 ip 只出现在一个文件中，再对每个小文件中的 ip 进行 hashmap 计数统计并按数量排序，最后归并或者最小堆依次处理每个小文件的 top10 以得到最后的结。

但如果数据规模比较小，能一次性装入内存呢?比如这第 2 题，虽然有一千万个 Query，但是由于重复度比较高，因此事实上只有 300 万的 Query，每个 Query255Byte，因此我们可以考虑把他们都放进内存中去（300 万个字符串假设没有重复，都是最大长度，那么最多占用内存 3M*1K/4=0.75G。所以可以将所有字符串都存放在内存中进行处理），而现在只是需要一个合适的数据结构，在这里，HashTable 绝对是我们优先的选择。

所以我们放弃分而治之/hash 映射的步骤，直接上 hash 统计，然后排序。So，针对此类典型的 TOP K 问题，采取的对策往往是：hashmap + 堆。如下所示：

1. **hash_map 统计**：先对这批海量数据预处理。具体方法是：维护一个 Key 为 Query 字串，Value 为该 Query 出现次数的 HashTable，即 hash_map(Query，Value)，每次读取一个 Query，如果该字串不在 Table 中，那么加入该字串，并且将 Value 值设为 1；如果该字串在 Table 中，那么将该字串的计数加一即可。最终我们在 O(N)的时间复杂度内用 Hash 表完成了统计；
2. **堆排序**：第二步、借助堆这个数据结构，找出 Top K，时间复杂度为 N‘logK。即借助堆结构，我们可以在 log 量级的时间内查找和调整/移动。因此，维护一个 K(该题目中是 10)大小的小根堆，然后遍历 300 万的 Query，分别和根元素进行对比。所以，我们最终的时间复杂度是：O（N） + N' * O（logK），（N 为 1000 万，N’为 300 万）。

别忘了这篇文章中所述的堆排序思路：“维护 k 个元素的最小堆，即用容量为 k 的最小堆存储最先遍历到的 k 个数，并假设它们即是最大的 k 个数，建堆费时 O（k），并调整堆(费时 O（logk）)后，有 k1>k2>...kmin（kmin 设为小顶堆中最小元素）。继续遍历数列，每次遍历一个元素 x，与堆顶元素比较，若 x>kmin，则更新堆（x 入堆，用时 logk），否则不更新堆。这样下来，总费时 O（k*logk+（n-k）*logk）=O（n*logk）。此方法得益于在堆中，查找等各项操作时间复杂度均为 logk。”--第三章续、Top K 算法问题的实现。

当然，你也可以采用 trie 树，关键字域存该查询串出现的次数，没有出现为 0。最后用 10 个元素的最小推来对出现频率进行排序。

3、有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 字节，内存限制大小是 1M。返回频数最高的 100 个词。

由上面那两个例题，分而治之 + hash 统计 + 堆/快速排序这个套路，我们已经开始有了屡试不爽的感觉。下面，再拿几道再多多验证下。请看此第 3 题：又是文件很大，又是内存受限，咋办?还能怎么办呢?无非还是：

1. **分而治之/hash 映射**：顺序读文件中，对于每个词 x，取 hash(x)%5000，然后按照该值存到 5000 个小文件（记为 x0,x1,...x4999）中。这样每个文件大概是 200k 左右。如果其中的有的文件超过了 1M 大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过 1M。
2. **hash_map 统计**：对每个小文件，采用 trie 树/hash_map 等统计每个文件中出现的词以及相应的频率。
3. **堆/归并排序**：取出出现频率最大的 100 个词（可以用含 100 个结点的最小堆）后，再把 100 个词及相应的频率存入文件，这样又得到了 5000 个文件。最后就是把这 5000 个文件进行归并（类似于归并排序）的过程了。

4、海量数据分布在 100 台电脑中，想个办法高效统计出这批数据的 TOP10。

如果每个数据元素只出现一次，而且只出现在某一台机器中，那么可以采取以下步骤统计出现次数 TOP10 的数据元素：

1. 堆排序：在每台电脑上求出 TOP10，可以采用包含 10 个元素的堆完成（TOP10 小，用最大堆，TOP10 大，用最小堆，比如求 TOP10 大，我们首先取前 10 个元素调整成最小

堆，如果发现，然后扫描后面的数据，并与堆顶元素比较，如果比堆顶元素大，那么用该元素替换堆顶，然后再调整为最小堆。最后堆中的元素就是 TOP10 大）。

2. 求出每台电脑上的 TOP10 后，然后把这 100 台电脑上的 TOP10 组合起来，共 1000 个数据，再利用上面类似的方法求出 TOP10 就可以了。

但如果同一个元素重复出现在不同的电脑中呢，如下例子所述：

就拿2台机器求top2的情况来说：第一台：a(50),b(50),c(49),d(49),e(0),f(0);第二台：a(0),b(0),c(49),d(49),e(50),f(50)

这个时候，你可以有两种方法：

- 遍历一遍所有数据，重新 hash 取摸，如此使得同一个元素只出现在单独的一台电脑中，然后采用上面所说的方法，统计每台电脑中各个元素的出现次数找出 TOP10，继而组合 100 台电脑上的 TOP10，找出最终的 TOP10。
- 或者，暴力求解：直接统计统计每台电脑中各个元素的出现次数，然后把同一个元素在不同机器中的出现次数相加，最终从所有数据中找出 TOP10。

5、有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求你按照 query 的频度排序。

方案 1：直接上：

1. hash 映射：顺序读取 10 个文件，按照 hash(query)%10 的结果将 query 写入到另外 10 个文件（记为 a0,a1,..a9）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。
2. hash_map 统计：找一台内存在 2G 左右的机器，依次对用 hash_map(query, query_count)来统计每个 query 出现的次数。注：hash_map(query,query_count)是用来统计每个 query 的出现次数，不是存储他们的值，出现一次，则 count+1。
3. 堆/快速/归并排序：利用快速/堆/归并排序按照出现次数进行排序，将排序好的 query 和对应的 query_cout 输出到文件中，这样得到了 10 个排好序的文件（记为 $b_0, b_1, \cdots, b_{10}$）。最后，对这 10 个文件进行归并排序（内排序与外排序相结合）。

   根据此方案 1，这里有一份实现：

   https://github.com/ooooola/sortquery/blob/master/querysort.py。

除此之外，此题还有以下两个方法：

方案 2：一般 query 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 query，一次性就可以加入到内存了。这样，我们就可以采用 trie 树/hash_map 等直接来统计每个 query 出现的次数，然后按出现次数做快速/堆/归并排序就可以了。

方案 3：与方案 1 类似，但在做完 hash，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理（比如 MapReduce），最后再进行合并。

6、 给定 a、b 两个文件，各存放 50 亿个 url，每个 url 各占 64 字节，内存限制是 4G，

让你找出 a、b 文件共同的 url？

可以估计每个文件安的大小为 5G×64=320G，远远大于内存限制的 4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

1. 分而治之/hash 映射：遍历文件 a，对每个 url 求取$hash(url)\%1000$，然后根据所取得的值将 url 分别存储到 1000 个小文件（记为$a_0, a_2, \cdots, a_{999}$，这里漏写个了 a1）中。这样每个小文件的大约为 300M。遍历文件 b，采取和 a 相同的方式将 url 分别存储到 1000 小文件中（记为$b_0, b_1, \cdots, b_{999}$）。这样处理后，所有可能相同的 url 都在对应的小文件（a_0 vs b_0, a_1 vs b_1, ..., a_0 vs b_{999}）中，不对应的小文件不可能有相同的 url。然后我们只要求出 1000 对小文件中相同的 url 即可。
2. hash_set 统计：求每对小文件中相同的 url 时，可以把其中一个小文件的 url 存储到 hash_set 中。然后遍历另一个小文件的每个 url，看其是否在刚才构建的 hash_set 中，如果是，那么就是共同的 url，存到文件里面就可以了。

OK，此第一种方法：分而治之/hash 映射 + hash 统计 + 堆/快速/归并排序，再看最后 4 道题，如下：

7、怎么在海量数据中找出重复次数最多的一个？

方案：先做 hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

8、上千万或上亿数据（有重复），统计其中出现次数最多的前 N 个数据。

方案：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用 hash_map/搜索二叉树/红黑树等来进行统计次数。然后利用堆取出前 N 个出现次数最多的数据。

9、一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词，请给出思想，给出时间复杂度分析。

方案 1：如果文件比较大，无法一次性读入内存，可以采用 hash 取模的方法，将大文件分解为多个小文件，对于单个小文件利用 hash_map 统计出每个小文件中 10 个最常出现的词，然后再进行归并处理，找出最终的 10 个最常出现的词。

方案 2：通过 hash 取模将大文件分解为多个小文件后，除了可以用 hash_map 统计出每个小文件中 10 个最常出现的词，也可以用 trie 树统计每个词出现的次数，时间复杂度是 O(n*le)（le 表示单词的平准长度），最终同样找出出现最频繁的前 10 个词（可用堆来实现），

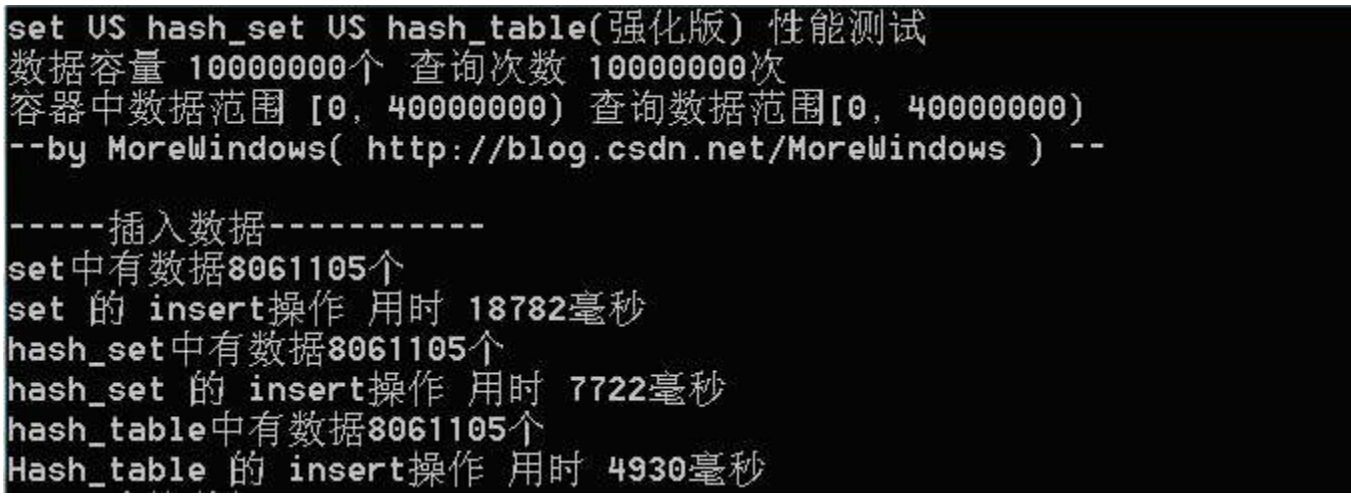
时间复杂度是 O(n*lg10)。

10. 1000 万字符串，其中有些是重复的，需要把重复的全部去掉，保留没有重复的字符串。请怎么设计和实现？

- 方案 1：这题用 trie 树比较合适，hash_map 也行。
- 方案 2：from xjbzju:，1000w 的数据规模插入操作完全不现实，以前试过在 stl 下 100w 元素插入 set 中已经慢得不能忍受，觉得基于 hash 的实现不会比红黑树好太多，使用 vector+sort+unique 都要可行许多，建议还是先 hash 成小文件分开处理再综合。

上述方案 2 中读者 xbzju 的方法让我想到了一些问题，即是 set/map，与 hash_set/hash_map 的性能比较?共计 3 个问题，如下：

- 1、hash_set 在千万级数据下，insert 操作优于 set？这位 blog：http://t.cn/zOibP7t 给的实践数据可靠不?
- 2、那 map 和 hash_map 的性能比较呢？谁做过相关实验?



- 3、那查询操作呢，如下段文字所述?

可以发现在hash_table中最长的链表也只有5个元素，**长度为1和长度为2的链表中的数据占全部数据的89%以上。因此绝大数查询将仅仅访问哈希表1次到2次。**这样的查询效率当然会比set（内部使用红黑树——类似于二叉平衡树）高的多。有了这个图示，无疑已经可以证明hash_set会比set快速高效了。但hash_set还可以动态的增加表的大小，因此我们再实现一个表大小可增加的hash_table。

或者小数据量时用 map，构造快，大数据量时用 hash_map?

rbtree PK hashtable

据朋友№邦卡猫№的做的红黑树和 hash table 的性能测试中发现：当数据量基本上 int 型 key 时，hash table 是 rbtree 的 3-4 倍，但 hash table 一般会浪费大概一半内存。

因为 hash table 所做的运算就是个%，而 rbtree 要比较很多，比如 rbtree 要看 value 的数据 ，每个节点要多出 3 个指针（或者偏移量） 如果需要其他功能，比如，统计某个范围内的 key 的数量，就需要加一个计数成员。

且 1s rbtree 能进行大概 50w+次插入，hash table 大概是差不多 200w 次。不过很多的时候，其速度可以忍了，例如倒排索引差不多也是这个速度，而且单线程，且倒排表的拉链长度不会太大。正因为基于树的实现其实不比 hashtable 慢到哪里去，所以数据库的索引一般都是用的 B/B+树，而且 B+树还对磁盘友好(B 树能有效降低它的高度，所以减少磁盘交互次数)。比如现在非常流行的 NoSQL 数据库，像 MongoDB 也是采用的 B 树索引。关于 B 树系列，请参考本 blog 内此篇文章：从 B 树、B+树、B*树谈到 R 树。更多请待后续实验论证。

11. 一个文本文件，找出前 10 个经常出现的词，但这次文件比较长，说是上亿行或十亿行，总之无法一次读入内存，问最优解。

方案 1：首先根据用 hash 并求模，将文件分解为多个小文件，对于单个文件利用上题的方法求出每个文件件中 10 个最常出现的词。然后再进行归并处理，找出最终的 10 个最常出现的词。

12. 100w 个数中找出最大的 100 个数。

方案 1：采用局部淘汰法。选取前 100 个元素，并排序，记为序列 L。然后一次扫描剩余的元素 x，与排好序的 100 个元素中最小的元素比，如果比这个最小的要大，那么把这个最小的元素删除，并把 x 利用插入排序的思想，插入到序列 L 中。依次循环，知道扫描了所有的元素。复杂度为 O(100w*100)。

方案 2：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的一部分在比 100 多的时候，采用传统排序算法排序，取前 100 个。复杂度为 O(100w*100)。

方案 3：在前面的题中，我们已经提到了，用一个含 100 个元素的最小堆完成。复杂度为 O(100w*lg100)。

接下来，咱们来看第二种方法，双层捅划分。

密匙二、多层划分

多层划分----其实本质上还是分而治之的思想，重在“分”的技巧上！

适用范围：**第 k 大，中位数，不重复或重复的数字**

基本原理及要点：因为元素范围很大，不能利用直接寻址表，所以通过多次划分，逐步确定范围，然后最后在一个可以接受的范围内进行。

问题实例：

13、2.5 亿个整数中找出不重复的整数的个数，内存空间不足以容纳这 2.5 亿个整数。

有点像鸽巢原理，整数个数为 2^32,也就是，我们可以将这 2^32 个数，划分为 2^8 个区域(比如用单个文件代表一个区域)，然后将数据分离到不同的区域，然后不同的区域在利用 bitmap 就可以直接解决了。也就是说只要有足够的磁盘空间，就可以很方便的解决。

14、5 亿个 int 找它们的中位数。

1. 思路一：这个例子比上面那个更明显。首先我们将 int 划分为 2^16 个区域，然后读取数据统计落到各个区域里的数的个数，之后我们根据统计结果就可以判断中位数落到那个区域，同时知道这个区域中的第几大数刚好是中位数。然后第二次扫描我们只统计落在这个区域中的那些数就可以了。

   实际上，如果不是 int 是 int64，我们可以经过 3 次这样的划分即可降低到可以接受的程度。即可以先将 int64 分成 2^24 个区域，然后确定区域的第几大数，在将该区域分成 2^20 个子区域，然后确定是子区域的第几大数，然后子区域里的数的个数只有 2^20，就可以直接利用 direct addr table 进行统计了。
2. 思路二@绿色夹克衫：同样需要做两遍统计，如果数据存在硬盘上，就需要读取 2 次。

   方法同基数排序有些像，开一个大小为 65536 的 Int 数组，第一遍读取，统计 Int32 的高 16 位的情况，也就是 0-65535，都算作 0,65536 - 131071 都算作 1。就相当于用该数除以 65536。Int32 除以 65536 的结果不会超过 65536 种情况，因此开一个长度为 65536 的数组计数就可以。每读取一个数，数组中对应的计数+1，考虑有负数的情况，需要将结果加 32768 后，记录在相应的数组内。

   第一遍统计之后，遍历数组，逐个累加统计，看中位数处于哪个区间，比如处于区间 k，那么 0- k-1 的区间里数字的数量 sum 应该<n/2（2.5 亿）。而 k+1 - 65535 的计数和也<n/2，第二遍统计同上面的方法类似，但这次只统计处于区间 k 的情况，也就是说(x / 65536) + 32768 = k。统计只统计低 16 位的情况。并且利用刚才统计的 sum，比如 sum = 2.49 亿，那么现在就是要在低 16 位里面找 100 万个数(2.5 亿-2.49 亿)。这次计数之后，再统计一下，看中位数所处的区间，最后将高位和低位组合一下就是结果了。

3).现在有一个 0-30000 的随机数生成器。请根据这个随机数生成器，设计一个抽奖范围是 0-350000 彩票中奖号码列表，其中要包含 20000 个中奖号码。

这个题刚好和上面两个思想相反，一个 0 到 3 万的随机数生成器要生成一个 0 到 35 万的随机数。那么我们完全可以将 0-35 万的区间分成 35/3=12 个区间，然后每个区间的长度都小于等于 3 万，这样我们就可以用题目给的随机数生成器来生成了，然后再加上该区间的基数。那么要每个区间生成多少个随机数呢？计算公式就是：区间长度*随机数密度，在本题目中就是 30000*（20000/350000）。最后要注意一点，该题目是有隐含条件的：彩票，这意味着你生成的随机数里面不能有重复，这也是我为什么用双层桶划分思想的另外一个原因

密匙三：Bloom filter/Bitmap

Bloom filter

关于什么是 **Bloom filter**，请参看 blog 内此文：

- 海量数据处理之 Bloom Filter 详解

适用范围：可以用来实现数据字典，进行数据的判重，或者集合求交集

基本原理及要点：

对于原理来说很简单，位数组+k 个独立 hash 函数。将 hash 函数对应的值的位数组置 1，查找时如果发现所有 hash 函数对应位都是 1 说明存在，很明显这个过程并不保证查找的结果是 100%正确的。同时也不支持删除一个已经插入的关键字，因为该关键字对应的位会牵动到其他的关键字。所以一个简单的改进就是 counting Bloom filter，用一个 counter 数组代替位数组，就可以支持删除了。

还有一个比较重要的问题，如何根据输入元素个数 n，确定位数组 m 的大小及 hash 函数个数。当 hash 函数个数 k=(ln2)*(m/n)时错误率最小。在错误率不大于 E 的情况下，m 至少要等于 n*lg(1/E)才能表示任意 n 个元素的集合。但 m 还应该更大些，因为还要保证 bit 数组里至少一半为 0，则 m 应该>=nlg(1/E)*lge 大概就是 nlg(1/E)1.44 倍(lg 表示以 2 为底的对数)。

举个例子我们假设错误率为 0.01，则此时 m 应大概是 n 的 13 倍。这样 k 大概是 8 个。

注意这里 m 与 n 的单位不同，m 是 bit 为单位，而 n 则是以元素个数为单位(准确的说是不同元素的个数)。通常单个元素的长度都是有很多 bit 的。所以使用 bloom filter 内存上通常都是节省的。

扩展：

Bloom filter 将集合中的元素映射到位数组中，用 k（k 为哈希函数个数）个映射位是否全 1 表示元素在不在这个集合中。Counting bloom filter（CBF）将位数组中的每一位扩展为一个 counter，从而支持了元素的删除操作。Spectral Bloom Filter（SBF）将其与集合元素的出现次数关联。SBF 采用 counter 中的最小值来近似表示元素的出现频率。

可以看下上文中的第 6 题：

“6、给你 A,B 两个文件，各存放 50 亿条 URL，每条 URL 占用 64 字节，内存限制是 4G，让你找出 A,B 文件共同的 URL。如果是三个乃至 n 个文件呢？

根据这个问题我们来计算下内存的占用，4G=2^32 大概是 40 亿*8 大概是 340 亿，n=50 亿，如果按出错率 0.01 算需要的大概是 650 亿个 bit。现在可用的是 340 亿，相差并不多，这样可能会使出错率上升些。另外如果这些 urlip 是一一对应的，就可以转换成 ip，则大大简单了。

同时，上文的第 5 题：给定 a、b 两个文件，各存放 50 亿个 url，每个 url 各占 64 字

节，内存限制是 4G，让你找出 a、b 文件共同的 url？如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。”

Bitmap

- 关于什么是 Bitmap，请看 blog 内此文第二部分：

  http://blog.csdn.net/v_july_v/article/details/6685962。

下面关于 Bitmap 的应用，可以看下上文中的第 13 题，以及另外一道新题：

“13、在 2.5 亿个整数中找出不重复的整数，注，内存不足以容纳这 2.5 亿个整数。

方案 1：采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进行，共需内存 2^32 * 2 bit=1 GB 内存，还可以接受。然后扫描这 2.5 亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01 变 10，10 保持不变。所描完事后，查看 bitmap，把对应位是 01 的整数输出即可。

方案 2：也可采用与第 1 题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。**”**

15、给 40 亿个不重复的 unsigned int 的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那 40 亿个数当中？

方案 1：frome oo，用位图/Bitmap 的方法，申请 512M 的内存，一个 bit 位代表一个 unsigned int 值。读入 40 亿个数，设置相应的 bit 位，读入要查询的数，查看相应 bit 位是否为 1，为 1 表示存在，为 0 表示不存在。

密匙四、Trie 树/数据库/倒排索引

Trie 树

适用范围：数据量大，重复多，但是数据种类小可以放入内存
基本原理及要点：实现方式，节点孩子的表示方式
扩展：压缩实现。
问题实例：

1. 上面的**第 2 题**：寻找热门查询：查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个，每个不超过 255 字节。

2. 上面的**第 5 题**：有 10 个文件，每个文件 1G，每个文件的每一行都存放的是用户的 query，每个文件的 query 都可能重复。要你按照 query 的频度排序。
3. 1000 万字符串，其中有些是相同的(重复),需要把重复的全部去掉，保留没有重复的字符串。请问怎么设计和实现？
4. 上面的**第 8** 题：一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词。其解决方法是：用 trie 树统计每个词出现的次数，时间复杂度是 O(n*le)（le 表示单词的平准长度），然后是找出出现最频繁的前 10 个词。

更多有关 Trie 树的介绍，请参见此文：从 Trie 树（字典树）谈到后缀树。

数据库索引

适用范围：大数据量的增删改查

基本原理及要点：利用数据的设计实现方法，对海量数据的增删改查进行处理。

- 关于数据库索引及其优化，更多可参见此文：http://www.cnblogs.com/pkuoliver/archive/2011/08/17/mass-data-topic-7-index-and-optimize.html；
- 关于 MySQL 索引背后的数据结构及算法原理，这里还有一篇很好的文章：http://blog.codinglabs.org/articles/theory-of-mysql-index.html；
- 关于 B 树、B+ 树、B* 树及 R 树，本 blog 内有篇绝佳文章：http://blog.csdn.net/v_JULY_v/article/details/6530142。

倒排索引(Inverted index)

适用范围：搜索引擎，关键字查询

基本原理及要点：为何叫倒排索引？一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。

以英文为例，下面是要被索引的文本：

T0 = "it is what it is"

T1 = "what is it"

T2 = "it is a banana"

我们就能得到下面的反向文件索引：

"a": {2}

"banana": {2}

"is": {0, 1, 2}

"it": {0, 1, 2}

"what": {0, 1}

检索的条件"what","is"和"it"将对应集合的交集。

正向索引开发出来用来存储每个文档的单词的列表。正向索引的查询往往满足每个文档有序频繁的全文查询和每个单词在校验文档中的验证这样的查询。在正向索引中，文档占据了中心的位置，每个文档指向了一个它所包含的索引项的序列。也就是说文档指向了

它包含的那些单词，而反向索引则是单词指向了包含它的文档，很容易看到这个反向的关系。

扩展：

问题实例：文档检索系统，查询那些文件包含了某单词，比如常见的学术论文的关键字搜索。

关于倒排索引的应用，更多请参见：

- 第二十三、四章：杨氏矩阵查找，倒排索引关键词 Hash 不重复编码实践，
- 第二十六章：基于给定的文档生成倒排索引的编码与实践。

密匙五、外排序

适用范围：大数据的排序，去重

基本原理及要点：外排序的归并方法，置换选择败者树原理，最优归并树

问题实例：

1).有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 个字节，内存限制大小是 1M。返回频数最高的 100 个词。

这个数据具有很明显的特点，词的大小为 16 个字节，但是内存只有 1M 做 hash 明显不够，所以可以用来排序。内存可以当输入缓冲区使用。

关于多路归并算法及外排序的具体应用场景，请参见 blog 内此文：

- 第十章、如何给 10^7 个数据量的磁盘文件排序

密匙六、分布式处理之 Mapreduce

MapReduce 是一种计算模型，简单的说就是将大批量的工作（数据）分解（MAP）执行，然后再将结果合并成最终结果（REDUCE）。这样做的好处是可以在任务被分解后，可以通过大量机器进行并行计算，减少整个操作的时间。但如果你要我再通俗点介绍，那么，说白了，Mapreduce 的原理就是一个归并排序。

适用范围：数据量大，但是数据种类小可以放入内存

基本原理及要点：将数据交给不同的机器去处理，数据划分，结果归约。

问题实例：

1. The canonical example application of MapReduce is a process to count the appearances of each different word in a set of documents:
2. 海量数据分布在 100 台电脑中，想个办法高效统计出这批数据的 TOP10。
3. 一共有 N 个机器，每个机器上有 N 个数。每个机器最多存 O(N)个数并对它们操作。如何找到 N^2 个数的中数(median)？

更多具体阐述请参见 blog 内：

- 从 Hadhoop 框架与 MapReduce 模式中谈海量数据处理，
- 及 MapReduce 技术的初步了解与学习。

其它模式/方法论，结合操作系统知识

至此，六种处理海量数据问题的模式/方法已经阐述完毕。据观察，这方面的面试题无外乎以上一种或其变形，然题目为何取为是：秒杀 99%的海量数据处理面试题，而不是 100%呢。OK，给读者看最后一道题，如下：

非常大的文件，装不进内存。每行一个 int 类型数据，现在要你随机取 100 个数。

我们发现上述这道题，无论是以上任何一种模式/方法都不好做，那有什么好的别的方法呢？我们可以看看：操作系统内存分页系统设计(说白了，就是映射+建索引)。

Windows 2000 使用基于分页机制的虚拟内存。每个进程有 4GB 的虚拟地址空间。基于分页机制，这 4GB 地址空间的一些部分被映射了物理内存，一些部分映射硬盘上的交换文 件，一些部分什么也没有映射。程序中使用的都是 4GB 地址空间中的虚拟地址。而访问物理内存，需要使用物理地址。 关于什么是物理地址和虚拟地址，请看：

- 物理地址 (physical address): 放在寻址总线上的地址。放在寻址总线上，如果是读，电路根据这个地址每位的值就将相应地址的物理内存中的数据放到数据总线中传输。如果是写，电路根据这个 地址每位的值就将相应地址的物理内存中放入数据总线上的内容。物理内存是以字节(8 位)为单位编址的。
- 虚拟地址 (virtual address): 4G 虚拟地址空间中的地址，程序中使用的都是虚拟地址。 使用了分页机制之后，4G 的地址空间被分成了固定大小的页，每一页或者被映射到物理内存，或者被映射到硬盘上的交换文件中，或者没有映射任何东西。对于一 般程序来说，4G 的地址空间，只有一小部分映射了物理内存，大片大片的部分是没有映射任何东西。物理内存也被分页，来映射地址空间。对于 32bit 的 Win2k，页的大小是 4K 字节。CPU 用来把虚拟地址转换成物理地址的信息存放在叫做页目录和页表的结构里。

物理内存分页，一个物理页的大小为 4K 字节，第 0 个物理页从物理地址 0x00000000 处开始。由于页的大小为 4KB，就是 0x1000 字节，所以第 1 页从物理地址 0x00001000 处开始。第 2 页从物理地址 0x00002000 处开始。可以看到由于页的大小是 4KB，所以只需要 32bit 的地址中高 20bit 来寻址物理页。

返回上面我们的题目：非常大的文件，装不进内存。每行一个 int 类型数据，现在要你随机取 100 个数。针对此题，我们可以借鉴上述操作系统中内存分页的设计方法，做出如下解决方案：

操作系统中的方法，先生成 4G 的地址表，在把这个表划分为小的 4M 的小文件做个索引，二级索引。30 位前十位表示第几个 4M 文件，后 20 位表示在这个 4M 文件的第几

个，等等，基于 key value 来设计存储，用 key 来建索引。

但如果现在只有 10000 个数，然后怎么去随机从这一万个数里面随机取 100 个数？请读者思考。更多海里数据处理面试题，请参见此文第一部分：

http://blog.csdn.net/v_july_v/article/details/6685962。

参考文献

1. 十道海量数据处理面试题与十个方法大总结；
2. 海量数据处理面试题集锦与 Bit-map 详解；
3. 十一、从头到尾彻底解析 Hash 表算法；
4. 海量数据处理之 Bloom Filter 详解；
5. 从 Trie 树（字典树）谈到后缀树；
6. 第三章续、Top K 算法问题的实现；
7. 第十章、如何给 10^7 个数据量的磁盘文件排序；
8. 从 B 树、B+树、B*树谈到 R 树；
9. 第二十三、四章：杨氏矩阵查找，倒排索引关键词 Hash 不重复编码实践；
10. 第二十六章：基于给定的文档生成倒排索引的编码与实践；
11. 从 Hadhoop 框架与 MapReduce 模式中谈海量数据处理；
12. 第十六~第二十章：全排列，跳台阶，奇偶排序，第一个只出现一次等问题；
13. http://blog.csdn.net/v_JULY_v/article/category/774945；
14. STL 源码剖析第五章，侯捷著；
15. 2012 百度实习生招聘笔试题：

    http://blog.csdn.net/hackbuteer1/article/details/7542774。

后记

经过上面这么多海量数据处理面试题的轰炸，我们依然可以看出这类问题是有一定的解决方案/模式的，所以，不必将其神化。然这类面试题所包含的问题还是比较简单的，若您在这方面有更多实践经验，欢迎随时来信与我不吝分享：zhoulei0907@yahoo.cn。当然，自会注明分享者及来源。

不过，相信你也早就意识到，若单纯论海量数据处理面试题，本 blog 内的有关海量数据处理面试题的文章已涵盖了你能在网上所找到的 70~80%。但有点，必须负责任的敬告大家：无论是这些海量数据处理面试题也好，还是算法也好，面试时，70~80%的人不是倒在这两方面，而是倒在基础之上(诸如语言，数据库，操作系统，网络协议等等)，所以，无论

任何时候，基础最重要，没了基础，便什么都不是。