前言：

找工作时（IT 行业），除了常见的软件开发以外，机器学习岗位也可以当作是一个选择，不少计算机方向的研究生都会接触这个，如果你的研究方向是机器学习/数据挖掘之类，且又对其非常感兴趣的话，可以考虑考虑该岗位，毕竟在机器智能没达到人类水平之前，机器学习可以作为一种重要手段，而随着科技的不断发展，相信这方面的人才需求也会越来越大。

纵观 IT 行业的招聘岗位，机器学习之类的岗位还是挺少的，国内大点的公司里百度，阿里，腾讯，网易，搜狐，华为（华为的岗位基本都是随机分配，机器学习等岗位基本面向的是博士）等会有相关职位，另外一些国内的中小型企业和外企也会招一小部分。当然了，其中大部分还是百度北京要人最多，上百人。阿里的算法岗位很大一部分也是搞机器学习相关的。另外本人有幸签约了网易杭州研究院的深度学习算法岗位，打算从事机器学习领域至少 5 年。非常感谢小易收留了我！

下面是本人在找机器学习岗位工作时，总结的常见机器学习算法（主要是一些常规分类器）大概流程和主要思想，希望对大家找机器学习岗位时有点帮助。实际上在面试过程中，懂这些算法的基本思想和大概流程是远远不够的，那些面试官往往问的都是一些公司内部业务中的课题，往往要求你不仅要懂得这些算法的理论过程，而且要非常熟悉怎样使用它，什么场合用它，算法的优缺点，以及调参经验等等。说白了，就是既要会点理论，也要会点应用，既要有点深度，也要有点广度，否则运气不好的话很容易就被刷掉，因为每个面试官爱好不同。

朴素贝叶斯：

有以下几个地方需要注意：

1. 如果给出的特征向量长度可能不同，这是需要归一化为通长度的向量（这里以文本分类为例），比如说是句子单词的话，则长度为整个词汇量的长度，对应位置是该单词出现的次数。

2. 计算公式如下：

$$p(c_i|w) = \frac{p(w|c_i)p(c_i)}{p(w)}$$

其中一项条件概率可以通过朴素贝叶斯条件独立展开。要注意一点就是 $p(w|c_i)$ 的计算方法，而由朴素贝叶斯的前提假设可知，$p(w_0, w_1, w_2..w_N|c_i) = p(w_0|c_i)p(w_1|c_i)p(w_2|c_i)...p(w_N|c_i)$，因此一般有两种，一种是在类别为 ci 的那些样本集中，找到 wj 出现次数的总和，然后除以该样本的总和；第二种

方法是类别为 ci 的那些样本集中，找到 wj 出现次数的总和，然后除以该样本中所有特征出现次数的总和。

3. 如果 $p(w|c_i)$ 中的某一项为 0，则其联合概率的乘积也可能为 0，即 2 中公式的分子为 0，为了避免这种现象出现，一般情况下会将这一项初始化为 1，当然为了保证概率相等，分母应对应初始化为 2（这里因为是 2 类，所以加 2，如果是 k 类就需要加 k，术语上叫做 laplace 光滑，分母加 k 的原因是使之满足全概率公式）。

朴素贝叶斯的优点：

对小规模的数据表现很好，适合多分类任务，适合增量式训练。

缺点：

对输入数据的表达形式很敏感。

决策树：

决策树中很重要的一点就是选择一个属性进行分枝，因此要注意一下信息增益的计算公式，并深入理解它。

信息熵的计算公式如下：

$$H = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

其中的 n 代表有 n 个分类类别（比如假设是 2 类问题，那么 n=2）。分别计算这 2 类样本在总样本中出现的概率 p1 和 p2，这样就可以计算出未选中属性分枝前的信息熵。

现在选中一个属性 xi 用来进行分枝，此时分枝规则是：如果 xi=vx 的话，将样本分到树的一个分支；如果不相等则进入另一个分支。很显然，分支中的样本很有可能包括 2 个类别，分别计算这 2 个分支的熵 H1 和 H2,计算出分枝后的总信息熵 H’=p1*H1+p2*H2.，则此时的信息增益 ΔH=H-H’。以信息增益为原则，把所有的属性都测试一边，选择一个使增益最大的属性作为本次分枝属性。

决策树的优点：

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

缺点：

容易过拟合（后续出现了随机森林，减小了过拟合现象）；

Logistic 回归：

Logistic 是用来分类的，是一种线性分类器，需要注意的地方有：

1. logistic 函数表达式为：

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

where

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

其导数形式为：

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

2. logsitc 回归方法主要是用最大似然估计来学习的，所以单个样本的后验概率为：

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

到整个样本的后验概率：

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

其中：

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

通过对数进一步化简为：

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

3. 其实它的 loss function 为-l(θ)，因此我们需使 loss function 最小，可采用梯度下降法得到。梯度下降法公式为:

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

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

Logistic 回归优点：

1、实现简单；

2、分类时计算量非常小，速度很快，存储资源低；

缺点：

1、容易欠拟合，一般准确度不太高

2、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；

线性回归：

线性回归才是真正用于回归的，而不像 logistic 回归是用于分类，其基本思想是用梯度下降法对最小二乘法形式的误差函数进行优化，当然也可以用 normal equation 直接求得参数的解，结果为：

$$\hat{w} = (X^T X)^{-1} X^T y$$

而在 LWLR（局部加权线性回归）中，参数的计算表达式为:

$$\hat{w} = (X^T W X)^{-1} X^T W y$$

因为此时优化的是：

1. Fit θ to minimize $\sum_i w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$.
2. Output $\theta^T x$.

由此可见 LWLR 与 LR 不同，LWLR 是一个非参数模型，因为每次进行回归计算都要遍历训练样本至少一次。

线性回归优点：

实现简单，计算简单；

缺点：

不能拟合非线性数据；

KNN 算法：

KNN 即最近邻算法，其主要过程为：

1. 计算训练样本和测试样本中每个样本点的距离（常见的距离度量有欧式距离，马氏距离等）；

2. 对上面所有的距离值进行排序；

3. 选前 k 个最小距离的样本；

4. 根据这 k 个样本的标签进行投票，得到最后的分类类别；

如何选择一个最佳的 K 值，这取决于数据。一般情况下，在分类时较大的 K 值能够减小噪声的影响。但会使类别之间的界限变得模糊。一个较好的 K 值可通过各种启发式技术来获取，比如，交叉验证。另外噪声和非相关性特征向量的存在会使 K 近邻算法的准确性减小。

近邻算法具有较强的一致性结果。随着数据趋于无限，算法保证错误率不会超过贝叶斯算法错误率的两倍。对于一些好的 K 值，K 近邻保证错误率不会超过贝叶斯理论误差率。

注：马氏距离一定要先给出样本集的统计性质，比如均值向量，协方差矩阵等。关于马氏距离的介绍如下：

马氏距离是由印度统计学家马哈拉诺比斯（P. C. Mahalanobis）提出的，表示数据的协方差距离。它是一种有效的计
同的是它考虑到各种特性之间的联系（例如：一条关于身高的信息会带来一条关于体重的信息，因为两者是有关联的）
于测量尺度。 对于一个均值为$\mu = (\mu_1, \mu_2, \mu_3, \ldots, \mu_p)^T$，协方差矩阵为$\sum$的多变量向量$x = (x_1, x_2, x_3, \ldots,$

$$D_M(x) = \sqrt{(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

马氏距离也可以定义为两个服从同一分布并且其协方差矩阵为$\sum$的随机变量$\vec{x}$与$\vec{y}$的差异程度：

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x}-\vec{y})^T\Sigma^{-1}(\vec{x}-\vec{y})}$$

如果协方差矩阵为单位矩阵，马氏距离就简化为欧氏距离；如果协方差矩阵为对角阵，其也可称为*正规化的欧氏距离*。

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{p}\frac{(x_i-y_i)^2}{\sigma_i^2}}$$

其中σ_i是x_i的标准差。

KNN 算法的优点：

1. 思想简单，理论成熟，既可以用来做分类也可以用来做回归；
2. 可用于非线性分类；
3. 训练时间复杂度为 O(n)；
4. 准确度高，对数据没有假设，对 outlier 不敏感；

缺点：

1. 计算量大；
2. 样本不平衡问题（即有些类别的样本数量很多，而其它样本的数量很少）；
3. 需要大量的内存；

SVM：

要学会如何使用 libsvm 以及一些参数的调节经验，另外需要理清楚 svm 算法的一些思路：

1. svm 中的最优分类面是对所有样本的几何裕量最大（为什么要选择最大间隔分类器，请从数学角度上说明？网易深度学习岗位面试过程中有被问到。答案就是几何间隔与样本的误分次数间存在关系：$误分次数 \leq \left(\frac{2R}{\delta}\right)^2$，其中的分母就是样本到分类间隔距离，分子中的 R 是所有样本中的最长向量值），即：

$$\begin{aligned} \max_{\gamma,w,b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \ldots, m \\ & ||w|| = 1. \end{aligned}$$

经过一系列推导可得为优化下面原始目标：

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

2．下面来看看拉格朗日理论：

$$\begin{aligned} \min_{w} \quad & f(w) \\ \text{s.t.} \quad & g_i(w) \leq 0, \quad i = 1, \ldots, k \\ & h_i(w) = 0, \quad i = 1, \ldots, l. \end{aligned}$$

To solve it, we start by defining the **generalized Lagrangian**

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w).$$

可以将 1 中的优化目标转换为拉格朗日的形式（通过各种对偶优化，KKD 条件），最后目标函数为：

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}(w^T x^{(i)} + b) - 1 \right].$$

我们只需要最小化上述目标函数，其中的 α 为原始优化问题中的不等式约束拉格朗日系数。

3．对 2 中最后的式子分别 w 和 b 求导可得：

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}.$$

$$\frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0.$$

由上面第 1 式子可以知道，如果我们优化出了 α，则直接可以求出 w 了，即模型的参数搞定。而上面第 2 个式子可以作为后续优化的一个约束条件。

4. 对 2 中最后一个目标函数用对偶优化理论可以转换为优化下面的目标函数：

$$\begin{aligned}\max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle. \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0,\end{aligned}$$

而这个函数可以用常用的优化方法求得 α，进而求得 w 和 b。

5. 按照道理，svm 简单理论应该到此结束。不过还是要补充一点，即在预测时有：

$$\begin{aligned} w^T x + b &= \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \right)^T x + b \\ &= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b. \end{aligned}$$

那个尖括号我们可以用核函数代替，这也是 svm 经常和核函数扯在一起的原因。

6. 最后是关于松弛变量的引入，因此原始的目标优化公式为：

$$\begin{aligned}\min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \\ & \xi_i \geq 0, \quad i = 1, \ldots, m.\end{aligned}$$

此时对应的对偶优化公式为：

$$\begin{aligned}\max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0,\end{aligned}$$

与前面的相比只是 α 多了个上界。

SVM 算法优点：

可用于线性/非线性分类，也可以用于回归；

低泛化误差；

容易解释；

计算复杂度较低；

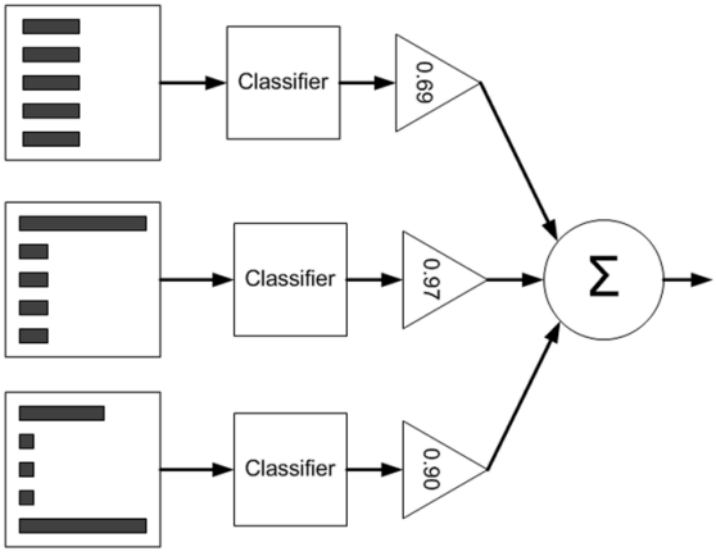
缺点：

对参数和核函数的选择比较敏感；

原始的 SVM 只比较擅长处理二分类问题；

Boosting：

主要以 Adaboost 为例，首先来看看 Adaboost 的流程图，如下：



从图中可以看到，在训练过程中我们需要训练出多个弱分类器（图中为 3 个），每个弱分类器是由不同权重的样本（图中为 5 个训练样本）训练得到（其中第一个弱分类器对应输入样本的权值是一样的），而每个弱分类器对最终分类结果的作用也不同，是通过加权平均输出的，权值见上图中三角形里面的数值。那么这些弱分类器和其对应的权值是怎样训练出来的呢？

下面通过一个例子来简单说明。

书中（machine learning in action）假设的是 5 个训练样本，每个训练样本的维度为 2，在训练第一个分类器时 5 个样本的权重各为 0.2. 注意这里样本的权值和最终训练的弱分类器组对应的权值 α 是不同的，样本的权重只在训练过程中用到，而 α 在训练过程和测试过程都有用到。

现在假设弱分类器是带一个节点的简单决策树，该决策树会选择 2 个属性（假设只有 2 个属性）的一个，然后计算出这个属性中的最佳值用来分类。

Adaboost 的简单版本训练过程如下：

1. 训练第一个分类器，样本的权值 D 为相同的均值。通过一个弱分类器，得到这 5 个样本（请对应书中的例子来看，依旧是 machine learning in action）的分类预测标签。与给出的样本真实标签对比，就可能出现误差(即错误)。如果某个样本预测错误，则它对应的错误值为该样本的权重，如果分类正确，则错误值为 0. 最后累加 5 个样本的错误率之和，记为 ε。

2. 通过 ε 来计算该弱分类器的权重 α，公式如下：

$$\alpha = \frac{1}{2} ln\left(\frac{1-\varepsilon}{\varepsilon}\right)$$

3. 通过 α 来计算训练下一个弱分类器样本的权重 D，如果对应样本分类正确，则减小该样本的权重，公式为：

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\alpha}}{Sum(D)}$$

如果样本分类错误，则增加该样本的权重，公式为：

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{\alpha}}{Sum(D)}$$

4. 循环步骤 1,2,3 来继续训练多个分类器，只是其 D 值不同而已。

测试过程如下：

输入一个样本到训练好的每个弱分类中，则每个弱分类都对应一个输出标签，然后该标签乘以对应的 α，最后求和得到值的符号即为预测标签值。

Boosting 算法的优点：

低泛化误差；

容易实现，分类准确率较高，没有太多参数可以调；

缺点：

对 outlier 比较敏感；

聚类：

根据聚类思想划分：

1. 基于划分的聚类：

K-means, k-medoids(每一个类别中找一个样本点来代表),CLARANS.

k-means 是使下面的表达式值最小：

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2$$

k-means 算法的优点：

（1）k-means 算法是解决聚类问题的一种经典算法，算法简单、快速。

（2）对处理大数据集，该算法是相对可伸缩的和高效率的，因为它的复杂度大约是 O(nkt)，其中 n 是所有对象的数目，k 是簇的数目,t 是迭代的次数。通常 k<<n。这个算法通常局部收敛。

（3）算法尝试找出使平方误差函数值最小的 k 个划分。当簇是密集的、球状或团状的，且簇与簇之间区别明显时，聚类效果较好。

缺点：

（1）k-平均方法只有在簇的平均值被定义的情况下才能使用，且对有些分类属性的数据不适合。

（2）要求用户必须事先给出要生成的簇的数目 k。

（3）对初值敏感，对于不同的初始值，可能会导致不同的聚类结果。

（4）不适合于发现非凸面形状的簇，或者大小差别很大的簇。

（5）对于"噪声"和孤立点数据敏感，少量的该类数据能够对平均值产生极大影响。

2. 基于层次的聚类：

自底向上的凝聚方法，比如 AGNES。

自上向下的分裂方法，比如 DIANA。

3. 基于密度的聚类：

DBSACN,OPTICS,BIRCH(CF-Tree),CURE.

4. 基于网格的方法：

STING, WaveCluster.

5. 基于模型的聚类：

EM,SOM,COBWEB.

以上这些算法的简介可参考聚类（百度百科）。

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：https://www.coursera.org/course/ml

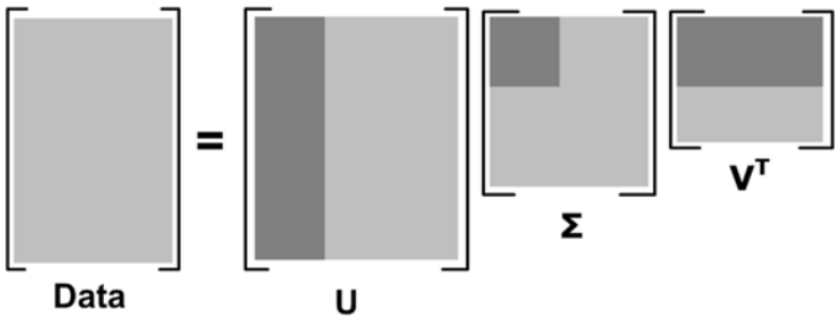
基于协同滤波的实现：

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：



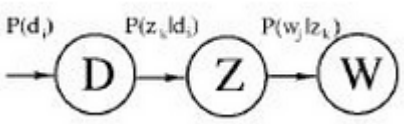
其中更深的颜色代表去掉小特征值重构时的三个矩阵。

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。

pLSA:

pLSA 由 LSA 发展过来，而早期 LSA 的实现主要是通过 SVD 分解。pLSA 的模型图如下：



公式中的意义如下：

(1) 以$P(d_i)$的概率选中文档d_i；

(2) 以$P(z_k|d_i)$的概率选中主题z_k；

(3) 以$P(w_j|z_k)$的概率产生一个单词。

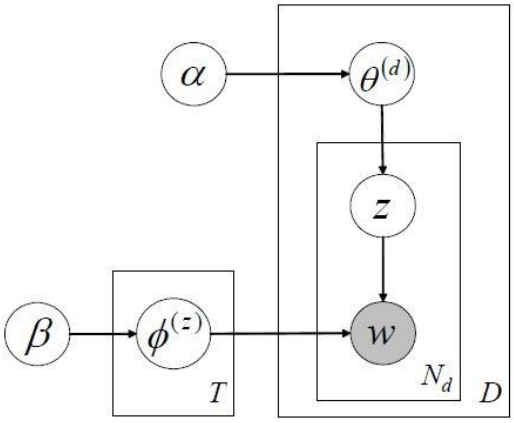
我们可以观察到的数据就是(d_i, w_j)对，而z_k是隐含变量。(d_i, w_j)的联合分布为

$$P(d_i, w_j) = P(d_i)P(w_j|d_i), \quad P(w_j|d_i) = \sum_{k=1}^{K} P(w_j|z_k)P(z_k|d_i).$$

具体可以参考 2010 龙星计划：机器学习中对应的主题模型那一讲

LDA：

主题模型，概率图如下：



和 pLSA 不同的是 LDA 中假设了很多先验分布，且一般参数的先验分布都假设为 Dirichlet 分布，其原因是共轭分布时先验概率和后验概率的形式相同。

GDBT：

GBDT(Gradient Boosting Decision Tree) 又叫 MART（Multiple Additive Regression Tree)，好像在阿里内部用得比较多（所以阿里算法岗位面试时可能会问到），它是一种迭代的决策树算法，该算法由多棵决策树组成，所有树的输出结果累加起来就是最终答案。它在被提出之初就和 SVM 一起被认为是泛化能力

（generalization)较强的算法。近些年更因为被用于搜索排序的机器学习模型而引起大家关注。

GBDT 是回归树，不是分类树。其核心就在于，每一棵树是从之前所有树的残差中来学习的。为了防止过拟合，和 Adaboosting 一样，也加入了 boosting 这一项。

关于 GDBT 的介绍可以可以参考：GBDT（MART） 迭代决策树入门教程 | 简介。

Regularization:

作用是（网易电话面试时有问到）：

1. 数值上更容易求解；

2. 特征数目太大时更稳定；

3. 控制模型的复杂度，光滑性。复杂性越小且越光滑的目标函数泛化能力越强。而加入规则项能使目标函数复杂度减小，且更光滑。

4. 减小参数空间；参数空间越小，复杂度越低。

5. 系数越小，模型越简单，而模型越简单则泛化能力越强（Ng 宏观上给出的解释）。

6. 可以看成是权值的高斯先验。

异常检测：

可以估计样本的密度函数，对于新样本直接计算其密度，如果密度值小于某一阈值，则表示该样本异常。而密度函数一般采用多维的高斯分布。如果样本有 n 维，则每一维的特征都可以看作是符合高斯分布的，即使这些特征可视化出来不太符合高斯分布，也可以对该特征进行数学转换让其看起来像高斯分布，比如说 x=log(x+c), x=x^(1/c)等。异常检测的算法流程如下：

Anomaly detection algorithm

1. Choose features x_i that you think might be indicative of anomalous examples.
2. Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

5. Given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $p(x) < \varepsilon$

其中的 ε 也是通过交叉验证得到的，也就是说在进行异常检测时，前面的 p(x)的学习是用的无监督，后面的参数 ε 学习是用的有监督。那么为什么不全部使用普通有监督的方法来学习呢（即把它看做是一个普通的二分类问题）？主要是因为在异常检测中，异常的样本数量非常少而正常样本数量非常多，因此不足以学习到好的异常行为模型的参数，因为后面新来的异常样本可能完全是与训练样本中的模式不同。

另外，上面是将特征的每一维看成是相互独立的高斯分布，其实这样的近似并不是最好的，但是它的计算量较小，因此也常被使用。更好的方法应该是将特征拟合成多维高斯分布，这时有特征之间的相关性，但随之计算量会变复杂，且样本的协方差矩阵还可能出现不可逆的情况（主要在样本数比特征数小，或者样本特征维数之间有线性关系时）。

上面的内容可以参考 Ng 的 https://www.coursera.org/course/ml

EM 算法：

有时候因为样本的产生和隐含变量有关（隐含变量是不能观察的），而求模型的参数时一般采用最大似然估计，由于含有了隐含变量，所以对似然函数参数求导是求不出来的，这时可以采用 EM 算法来求模型的参数的（对应模型参数个数可能有多个），EM 算法一般分为 2 步：

E 步：选取一组参数，求出在该参数下隐含变量的条件概率值；

M 步：结合 E 步求出的隐含变量条件概率，求出似然函数下界函数（本质上是某个期望函数）的最大值。

重复上面 2 步直至收敛。

公式如下所示：

(E-step) For each i, set

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

(M-step) Set

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

M 步公式中下界函数的推导过程：

$$\begin{aligned}
\sum_i \log p(x^{(i)};\theta) &= \sum_i \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)};\theta) && (1)\\
&= \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})} && (2)\\
&\geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})} && (3)
\end{aligned}$$

EM 算法一个常见的例子就是 GMM 模型，每个样本都有可能由 k 个高斯产生，只不过由每个高斯产生的概率不同而已，因此每个样本都有对应的高斯分布（k 个中的某一个），此时的隐含变量就是每个样本对应的某个高斯分布。

GMM 的 E 步公式如下（计算每个样本对应每个高斯的概率）：

(E-step) For each i, j, set

$$w_j^{(i)} := p(z^{(i)} = j|x^{(i)};\phi,\mu,\Sigma)$$

更具体的计算公式为：

$$p(z^{(i)} = j|x^{(i)};\phi,\mu,\Sigma) = \frac{p(x^{(i)}|z^{(i)} = j;\mu,\Sigma)p(z^{(i)} = j;\phi)}{\sum_{l=1}^{k} p(x^{(i)}|z^{(i)} = l;\mu,\Sigma)p(z^{(i)} = l;\phi)}$$

M 步公式如下（计算每个高斯的比重，均值，方差这 3 个参数）：

(M-step) Update the parameters:

$$\begin{aligned}
\phi_j &:= \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)},\\
\mu_j &:= \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}},\\
\Sigma_j &:= \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}
\end{aligned}$$

关于 EM 算法可以参考 Ng 的 cs229 课程资料 或者网易公开课：斯坦福大学公开课 ：机器学习课程。

Apriori:

Apriori 是关联分析中比较早的一种方法，主要用来挖掘那些频繁项集合。其思想是：

1. 如果一个项目集合不是频繁集合，那么任何包含它的项目集合也一定不是频繁集合；

2. 如果一个项目集合是频繁集合，那么它的任何非空子集也是频繁集合；

Aprioir 需要扫描项目表多遍，从一个项目开始扫描，舍去掉那些不是频繁的项目，得到的集合称为 L，然后对 L 中的每个元素进行自组合，生成比上次扫描多一个项目的集合，该集合称为 C，接着又扫描去掉那些非频繁的项目，重复...

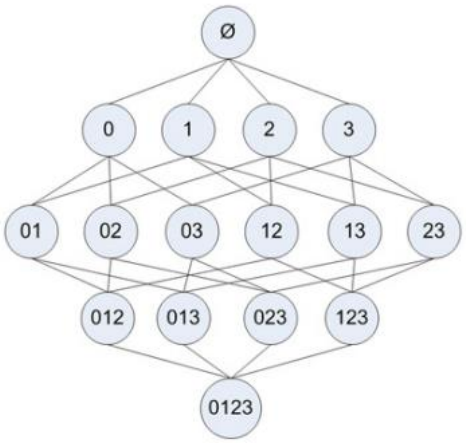
看下面这个例子：

元素项目表格：

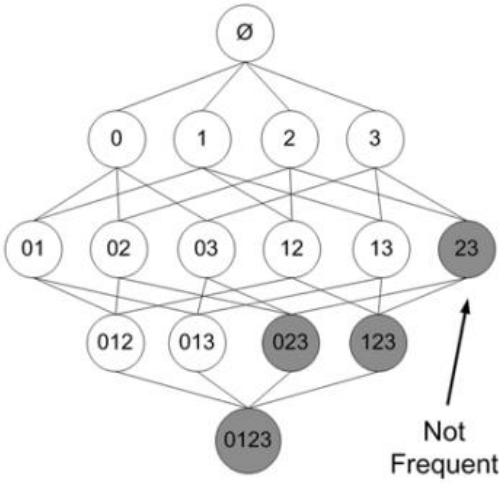
Transaction number	Items
0	soy milk, lettuce
1	lettuce, diapers, wine, chard
2	soy milk, diapers, wine, orange juice
3	lettuce, soy milk, diapers, wine
4	lettuce, soy milk, diapers, orange juice

Figure 11.1 A simple list of transactions from a natural foods grocery store called Hole Foods

如果每个步骤不去掉非频繁项目集，则其扫描过程的树形结构如下：



在其中某个过程中，可能出现非频繁的项目集，将其去掉（用阴影表示）为：



上面的内容主要参考的是 machine learning in action 这本书。

FP Growth:

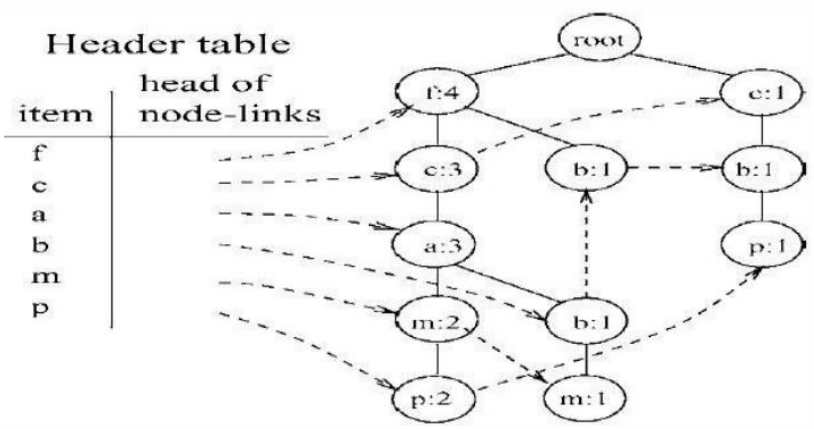
FP Growth 是一种比 Apriori 更高效的频繁项挖掘方法，它只需要扫描项目表 2 次。其中第 1 次扫描获得当个项目的频率，去掉不符合支持度要求的项，并对剩下的项排序。第 2 遍扫描是建立一颗 FP-Tree(frequent-patten tree)。

接下来的工作就是在 FP-Tree 上进行挖掘。

比如说有下表：

TID	Items bought	(Ordered) frequent items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

它所对应的 FP_Tree 如下：



然后从频率最小的单项 P 开始，找出 P 的条件模式基，用构造 FP_Tree 同样的方法来构造 P 的条件模式基的 FP_Tree，在这棵树上找出包含 P 的频繁项集。

依次从 m,b,a,c,f 的条件模式基上挖掘频繁项集，有些项需要递归的去挖掘，比较麻烦，比如 m 节点，具体的过程可以参考博客：Frequent Pattern 挖掘之二(FP Growth 算法)，里面讲得很详细。