## Machine Learning 相关问题

1. 最小二乘与梯度下降的区别？

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

### 线性最小二乘的基本公式

编辑

考虑超定方程组（超定指未知数小于方程个数）：

$$\sum_{j=1}^{n} X_{ij}\beta_j = y_i, (i = 1, 2, 3...m)$$

其中m代表有m个等式，n代表有 n 个未知数 $\beta$，m>n；将其进行向量化后为：

$$X\beta = y$$

$$X = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} \\ X_{21} & X_{22} & \cdots & X_{2n} \\ \vdots & \vdots & & \vdots \\ X_{m1} & X_{m2} & \cdots & X_{mn} \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

显然该方程组一般而言没有解，所以为了选取最合适的 $\beta$ 让该等式"尽量成立"，引入残差平方和函数S

$$S(\beta) = \|X\beta - y\|^2$$

（在统计学中，残差平方和函数可以看成n倍的均方误差MSE）

当 $\beta = \hat{\beta}$ 时，$S(\beta)$ 取最小值，记作：

$$\hat{\beta} = argmin(S(\beta))$$

通过对 $S(\beta)$ 进行微分 [2] 求最值，可以得到：

$$X^T X\hat{\beta} = X^T y$$

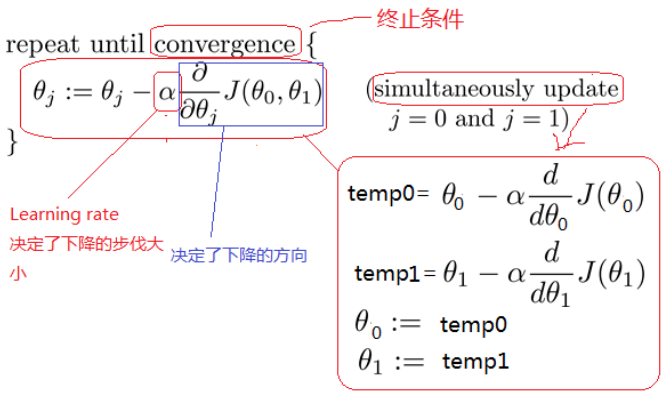
如果矩阵 $X^T X$ 非奇异则 $\beta$ 有唯一解 [3]：

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

**梯度下降法**：梯度下降法是一个最优化算法，是求解无约束优化问题最简单的方法之一。最速下降法是用负梯度方向为搜索方向的，最速下降法越接近目标值，步长越小，前进越慢。可以用于求解非线性方程组。

**方法**：
(1)先确定向下一步的步伐大小，我们称为Learning rate；
(2)任意给定一个初始值：$\theta_0$ $\theta_1$；
(3)确定一个向下的方向，并向下走预先规定的步伐，并更新 $\theta_0$ $\theta_1$；
(4)当下降的高度小于某个定义的值，则停止下降；

**算法**：




**特点**：
(1)初始点不同，获得的最小值也不同，因此梯度下降求得的只是局部最小值；
(2)越接近最小值时，下降速度越慢；

**高斯-牛顿法**：高斯—牛顿迭代法的基本思想是使用泰勒级数展开式去近似地代替非线性回归模型，然后通过多次迭代，多次修正回归系数，使回归系数不断逼近非线性回归模型的最佳回归系数，最后使原模型的残差平方和达到最小。

若用牛顿法求式3，则牛顿迭代公式为：

$$x^{(k+1)} = x^{(k)} - H^{-1}\nabla f$$

其中，H 为函数 f(x)的黑森矩阵，$\nabla f$ 为 f(x)的梯度，它们的数学表达式：

$$\nabla f = 2\sum_{i=1}^{m} r_i \frac{\delta r_i}{\delta x_j}$$

牛顿法迭代公式

$$H_{jk} = 2\sum_{i=1}^{m}\left(\frac{\delta r_i}{\delta x_j}\frac{\delta r_i}{\delta x_k} + r_i\frac{\partial^2 r_i}{\partial x_j \partial x_k}\right)$$

高斯牛顿法通过舍弃黑森矩阵的二阶偏导数实现，也就是：

$H_{jk} \approx 2\sum_{i=1}^{m} J_{ij}J_{ik}$，其中雅可比矩阵 $J_r$ 中的元素 $J_{ij} = \frac{\delta r_i}{\delta x_j}$

那么梯度和黑森矩阵可以写成如下简化形式：

$\nabla f = 2J_r^T r$，$H \approx 2J_r^T J_r$

高斯-牛顿法迭代公式

相应的迭代公式为：

$$x^{(k+1)} = x^{(k)} + \Delta,\quad \Delta = -\left(J_r^T J_r\right)^{-1} J_r^T r$$

看到这里大家都明白高斯牛顿和牛顿法的差异了吧，就在这迭代项上。经典高斯牛顿算法迭代步长λ为1.

那回过头来，高斯牛顿法里为啥要舍弃黑森矩阵的二阶偏导数呢？主要问题是因为牛顿法中Hessian矩阵中的二阶信息项通常难以计算或者花费的工作量很大，而利用整个H的割线近似也不可取，因为在计算梯度 $\nabla f$ 时已经得到J(x)，这样H中的一阶信息项$J^TJ$几乎是现成的。鉴于此，为了简化计算，获得有效算法，我们可用一阶导数信息逼近二阶信息项。注意这么干的前提是，残差r接近于零或者接近线性函数从而 $\nabla^2 r$ 接近与零时，二阶信息项才可以忽略。通常称为“小残量问题”，否则高斯牛顿法不收敛。

**各参数求解方法优缺点**：

解析法：需要用目标函数的到函数，

梯度法：又称最速下降法，是早期的解析法，收敛速度较慢

牛顿法：收敛速度快，但不稳定，计算也较困难。高斯牛顿法基于其改进，但目标作用不同

共轭梯度法：收敛较快，效果好

变尺度法：效率较高，常用 DFP 法(Davidon Fletcher Powell)

2. 最优化问题中，牛顿法为什么比梯度下降法求救需要迭代次数更少？

https://www.zhihu.com/question/19723347

（1）牛顿法是二阶收敛的，而梯度下降法是一阶收敛。二阶其实相当于考虑了梯度的梯度，所以相对更快。

（2）梯度下降法在非常靠近最优点时会有震荡，就是说明明离的很近了，却很难到达，因为线性的逼近非常容易一个方向过去就过了最优点(因为只能是负梯度方向)。但牛顿法因为是二次收敛就很容易到达了。

（3）牛顿迭代与梯度下降法的不同之处在于多了一项二阶导数。这个二阶导数对算法的影响有两个方面，方向和大小。方向，二阶导数的几何含义是曲率，也就是迭代的时候是考虑到梯度下降的方向的。大小，每次迭代的步长是和陡度成反比的，越陡步长越小，平坦的时候步长越大。

3. 最小二乘法、极大似然、梯度下降有何区别？

最小二乘和极大似然是目标函数，梯度下降是优化算法。

Linear Regression 的 cost function 是最小二乘，即

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

但是 Logistic Regression 的 cost function 却是

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

最小二乘法只能解决线性最小二乘问题，而 logistic 回归的损失函数不是线性最小二乘问题，因此 logistic 回归的优化算法梯度下降法；

通过最优化算法，计算上述损失函数，即通过$\min J(\theta)$求出对应$\theta$。这其中的最优化算法包括常用的梯度下降法(最速下降法)、牛顿法、拟牛顿法等。

4. 梯度下降 or 拟牛顿法

牛顿和拟牛顿法在凸优化情形下，如果迭代点离全局最优很近时，收敛速率快于梯度下降法。

然而：

1）**大数据带来的问题**：因为数据量大，从计算上来说不可能每一次迭代都使用全部样本计算优化算法所需的统计量（梯度，Hessian 等等），因此只能基于 batch 来计算，从而引入了噪声（sgd）。梯度的估计本身已经带了噪声，利用有噪声的梯度和历史梯度用近似公式逼近 Hessian（L-BFGS），则噪声很可能更大，而且微分本身时放大噪声的。所以即使多花费了计算量，拟牛顿的效果未必会更好。

2）**高维带来的问题**：因为参数极多，所以 Hessian 阵也会非常巨大，无法显式计算和存储，因此牛顿法几乎不可行，只能尝试用 L-BFGS。

在文本应用中，第一层 embedding 的参数量占整体参数量的绝大多数，因此第一层参数的计算速度与空间消耗几乎决定了整体消耗。由于文本输入是高维稀疏的，因此在每一个 batch 下第一层的梯度也是稀疏的（只有该样本出现的字（词）所连接的那些权值的梯度是非零的），因此 sgd 每一步迭代不需要把第一层所有的参数的梯度进行通信，而只需要通信极小部分的非零的梯度，这样稀疏更新会大大加快计算速度（通信量大大减小，因此通信速度增加）。而牛顿法或者拟牛顿法$p = -H^{-1}g$，本质上是使用几乎稠密的矩阵$H^{-1}$（Hessian 或其近似），对梯度的各维度信息进行混合、整定，因此实际的更新几乎不稀疏，无法使用稀疏更新，会大大减慢计算速度。

从直观上讲，sgd 相当于样本里出现的每个字或词，其信息只利用在与该字或词直接相关的网络参数的更新。因此如果一个字或词始终没出现，那么其对应的网络就完全没有学习任何东西，其权值在最后还是等于初始化后的值。牛顿法或拟牛顿法的思路则是不同字词之间是存在关联的，一个字或词的信息，应该同时分享并贡献到其他字词的参数的更新。

3）**非凸带来的问题**：非凸情形下，牛顿或拟牛顿法会被鞍点吸引。或者更直接的说，由于非凸，$H$非正定，导致$-H^{-1}g$不再是下降方向，每一步迭代反倒可能不降反升。当然，可以通过对一些维度做修正（Hessian modification），但是在高维情形，鞍点数量极多，而且$H$本身是带噪声的估计，因此修正未必是可行之路。

5. 判断函数凸或非凸

凸函数定义：凸函数是一个定义在某个向量空间的凸子集 C（区间）上的实值函数 f，而且对于凸子集 C 中任意两个向量$x_1, x_2$，总有$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$。

由于一阶导数连续减小,所以凸函数的二阶导数小于 0。

Hessian 矩阵半正定也可证明是凸函数。

Ps. 半正定：$X^T MX \geq 0$

正定的充要条件：

判定定理 1：对称阵 A 为正定的充分必要条件是：A 的特征值全为正。

判定定理 2：对称阵 A 为正定的充分必要条件是：A 的各阶顺序主子式都为正。

判定定理 3：任意阵 A 为正定的充分必要条件是：A 合同于单位阵。

设 A,B 是两个 n 阶方阵，若存在可逆矩阵 C，使得$C^T AC = B$，则称方阵 A 与 B 合同，记作 A≃B。

6. 无监督和有监督算法的区别，什么是半监督？

https://www.zhihu.com/question/20446337

区别：训练样本和待分类的类别已知；

无监督学习：K-聚类、PCA 主成分分析

有监督学习：SVM 支持向量机、线性判别、KNN、线性回归、逻辑回归、决策树、随机森林、神经网络等

半监督学习：S3VM、S4VM、CS4VM、TSVM

半监督学习使用大量的未标记数据，以及同时使用标记数据，来进行模式识别工作。当使用半监督学习时，将会要求尽量少的人员来从事工作，同时，又能够带来比较高的准确性。

7. 判别模型和生成模型？

有监督机器学习方法可以分为生成方法和判别方法。

常见的生成方法有混合高斯模型、朴素贝叶斯法和隐形马尔科夫模型等；

常见的判别方法有 SVM、LR 等；

判别模型的结果是$P(\hat{c}|X)$，即输出ĉ（label）关于输入X（feature）的条件分布；

生成模型的结果是$P(\hat{c},X)$，即输入X（feature）和输出ĉ（label）的联合分布。

**生成模型：**

优点：

1）生成给出的是联合分布$P(\hat{c},X)$，不仅能够由联合分布计算条件分布$P(\hat{c}|X)$（反之则不行），还可以给出其他信息，比如可以使用$P(\hat{X}) = \sum_{i=1}^{k} P(X|\hat{c}_i) * P(\hat{c}_i|X)$来计算边缘分布$P(\hat{X})$。如果一个输入样本的边缘分布$P(\hat{X})$很小的话，那么可以认为学习出的这个模型可能不太适合对这个样本进行分类，分类效果可能会不好，这也是所谓的 outlier detection。

2）生成模型收敛速度比较快，即当样本数量较多时，生成模型能更快地收敛于真实模型。

3）生成模型能够应付存在隐变量的情况，比如混合高斯模型就是含有隐变量的生成方法。

缺点：

1）天下没有免费午餐，联合分布是能提供更多的信息，但也需要更多的样本和更多计算，尤其是为了更准确估计类别条件分布，需要增加样本的数目，而且类别条件概率的许多信息是我们做分类用不到，因而如果我们只需要做分类任务，就浪费了计算资源。

2）另外，实践中多数情况下判别模型效果更好。

**判别模型：**

优点：

1）与生成模型缺点对应，首先是节省计算资源，另外，需要的样本数量也少于生成模型。

2）准确率往往较生成模型高。

3）由于直接学习$P(\hat{c}|X)$，而不需要求解类别条件概率，所以允许我们对输入进行抽象（比如降维、构造等），从而能够简化学习问题。

缺点：

1）是没有生成模型的上述优点。

8. LR 的损失函数和含义，梯度下降简单推导、特性？

logistic函数求导

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

逻辑回归的对数似然损失函数(cost function)：

$$
cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if y=1} \\ -log(1-h_\theta(x)) & \text{if y=0} \end{cases}
$$

逻辑回归最终的损失函数最终表达式：

$$cost(h_\theta(x), y) = \sum_{i=1}^{m} -y_i log(h_\theta(x)) - (1 - y_i) log(1 - h_\theta(x))$$

逻辑回归损失函数定义为：

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

梯度下降算法中，自变量迭代过程：

$$\theta_j = \theta_j - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta_j}, (j = 0,1, \dots, n)$$

梯度下降推导过程：

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m}\left( y^{(i)} \frac{1}{h_\theta(x^{(i)})} \frac{\partial h_\theta(x^i)}{\partial \theta_j} - (1-y^{(i)}) \frac{1}{1-h_\theta(x^{(i)})} \frac{\partial h_\theta(x^i)}{\partial \theta_j} \right) \cdots\cdots (1)$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left( y^{(i)} \frac{1}{g(\theta^T x^{(i)})} - (1-y^{(i)}) \frac{1}{1-g(\theta^T x^{(i)})} \right) \cdot \frac{\partial g(\theta^T x^{(i)})}{\partial \theta_j} \cdots\cdots (2)$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left( y^{(i)} \frac{1}{g(\theta^T x^{(i)})} - (1-y^{(i)}) \frac{1}{1-g(\theta^T x^{(i)})} \right) \cdot g(\theta^T x^{(i)})(1-g(\theta^T x^{(i)}) x_j^{(i)} \cdots (3)$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left( y^{(i)}(1-g(\theta^T x^{(i)}) - (1-y^{(i)}) g(\theta^T x^{(i)}) \right) \cdot x_j^{(i)} \cdots\cdots (4)$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left( y^{(i)} - g(\theta^T x^{(i)}) \right) \cdot x_j^{(i)} \cdots\cdots (5)$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left( h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \cdots\cdots (6)$$

(1)--->(2)：使用 sigmoid 函数的形式 g(z)替换 hθ(x)、提出公因子，放在式子尾;

(2)--->(3)：这一步具体推导如下:

$$\frac{\partial g(\theta^T x^{(i)})}{\partial \theta_j} = \frac{dg(z)}{dz} \cdot \frac{\partial(\theta^T x^{(i)})}{\partial \theta_j} \cdots\cdots g(z) = \frac{1}{1+e^{-z}} : 令z=\theta^T x^{(i)}$$

$$= \frac{e^{-z}}{(1+e^{-z})^2} \cdot \frac{\partial(\theta_0 + \theta_1 x_1 + \cdots + \theta_j x_j + \cdots + \theta_m x_m)}{\partial \theta_j}$$

$$= \frac{e^{-z} - 1 + 1}{(1+e^{-z})^2} \cdot x_j^{(i)}$$

$$= \left[ \frac{1}{1+e^{-z}} - \left( \frac{1}{1+e^{-z}} \right)^2 \right] \cdot x_j^{(i)}$$

$$= \left( g(z) - g^2(z) \right) \cdot x_j^{(i)}$$

$$= g(\theta^T x^{(i)}) \left( 1 - g(\theta^T x^{(i)}) \right) \cdot x_j^{(i)}$$

迭代公式的最终形式：

$$\theta_j = \theta_j - \partial \cdot \frac{1}{m}\sum_{i=1}^{m}\left( h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

9. 什么是准确率、召回率、F 值、ROC 曲线、AUC？

假设：定好了一个阈值，超过此阈值定义为坏用户（1），低于此阈值定义为好用户（0）

混淆矩阵（Confusion matrix）：

| 真实情况 | 预测结果 | |
|---|---|---|
| | 正例 | 反例 |
| 正例 | **TP**（真正例） | **FN**（假反例） |
| 反例 | **FP**（假正例） | **TN**（真反例） |

**精确率（Precision）**为在预测为坏人的人中，预测正确（实际为坏人）的人占比:

$$P = \frac{TP}{TP + FP}$$

**召回率（Recall）**即为在实际为坏人的人中，预测正确（预测为坏人）的人占比：

$$R = \frac{TP}{TP + FN}$$

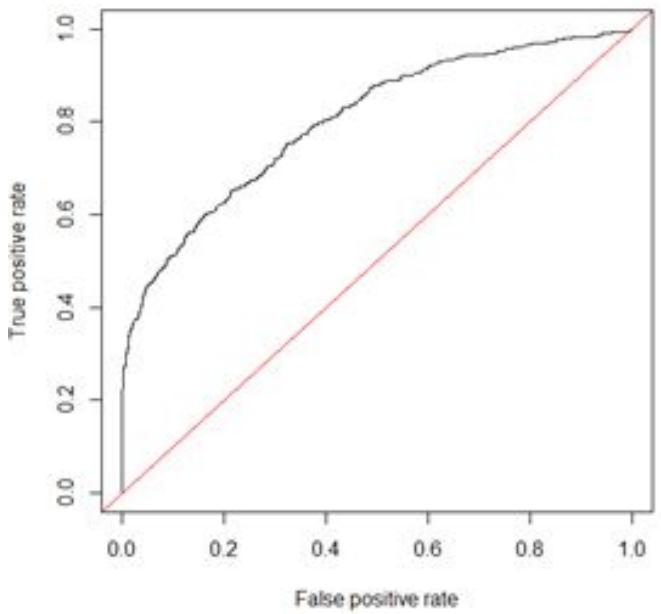
**F1 值**是精确率和召回率的调和均值，，相当于精确率和召回率的综合评价指标：

$$F1 = \frac{2PR}{P + R}$$

另外还有 Fα 值，为 F1 值的变体，利用α给 P 和 R 赋予不同的权重：

$$F\alpha = \frac{(\alpha^2 + 1)PR}{\alpha^2 P + R}$$

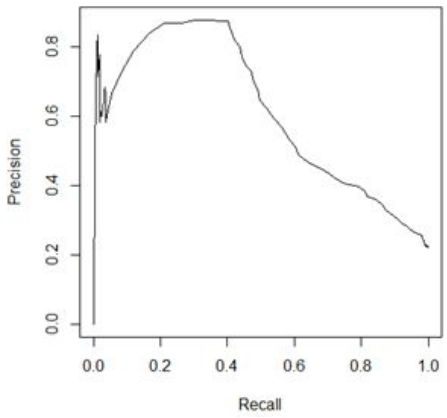
**ROC 曲线**（Receiver operating characteristic curve）：ROC 曲线其实是多个混淆矩阵的结果组合，如果在模型中我们没有定好阈值，而是将模型预测结果从高到低排序，将每个概率值依次作为阈值，那么就有多个混淆矩阵。对于每个混淆矩阵，我们计算两个指标**真正率 TPR**（True positive rate）和**假正率 FPR**（False positive rate），**TPR=TP/(TP+FN)=Recall**，TPR 就是召回率。**FPR=FP/(FP+TN)**，FPR 即为实际为好人的人中，预测为坏人的人占比。我们以 FPR 为 x 轴，TPR 为 y 轴画图，就得到了 ROC 曲线。ROC 曲线经过（0,1）点。




**AUC**（Area Under Curve）的值为 ROC 曲线下面的面积，若如上所述模型十分准确，则 AUC 为 1。但现实生活中尤其是工业界不会有如此完美的模型，一般 AUC 均在 0.5 到 1 之间，AUC 越高，模型的区分能力越好，上图 AUC 为 0.81。若 AUC=0.5，即与上图中红线重合，表示模型的区分能力与随机猜测没有差别。若 AUC 真的小于 0.5，请检查一下是不是好坏标签标反了，或者是模型真的很差。

**PR 曲线**（Precision-Recall curve）和 ROC 曲线类似，ROC 曲线是 FPR 和 TPR 的点连成的线，PR 曲线是准确率

和召回率的点连成的线，如下图所示：




10. 决策树的原理？

**信息熵(information entropy)：**

$$Ent(D) = -\sum_{k=1}^{|y|} p_k \cdot log_2 p_k$$

$Ent(D)$值越小，则 D 的纯度越高。

**信息增益(information gain)：**

$$Gain(D,a) = Ent(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} Ent(D^v)$$

信息增益准则对可取数目较多的属性有所偏好，为减少这种偏好带来的不利影响，C4.5 决策树算法不直接使用信息增益，而是使用「增益率（gain ratio)」来选择优先划分的属性。

**增益率(gain ratio):**

$$Gain\_ratio(D,a) = \frac{Gain(D,a)}{IV(a)}$$

其中，

$$IV(a) = -\sum_{v=1}^{V} \frac{|D^v|}{|D|} \log_2 \frac{|\mathrm{D}^2|}{|\mathrm{D}|}$$

增益率准则对可取数目较少的属性有所偏好，因此 C4.5 并不是直接选择增益率最大的候选划分属性，而是使用一个启发式：先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益率最高的属性。

**基尼系数（Gini index）：**

$$Gini(D) = \sum_{k=1}^{|y|} \sum_{k' \neq k}^{|y|} p_k p_k' = 1 - \sum_{k=1}^{|y|} p_k^2$$

**基尼指数：**

$$Gini\_index(D,a) = \sum_{v=1}^{V} \frac{|D^v|}{|D|} Gini(D^v)$$

在候选属性集合 A 中，选择那个使得划分后基尼指数最小的属性作为最优划分属性，即：

$$a_* = argmin(Gini\_index(D,a))$$

11. SVM 用的什么损失函数，特性？

折叶损失（hinge loss）

SVM 的损失函数定义如下：

$$L_i = \sum_{j \neq y_i} max\ (0, s_j - s_{y_i} + \Delta)$$

其中Δ代表间隔，上面的公式是将所有不正确分类$y \neq y_i$ 加起来。

在神经网路中，$f(x_i, W) = W * x_i$，因此

$$L_i = \sum_{j \neq y_i} \max\left(0, W_j^T x_i - W_{y_i}^T x_i + \Delta\right)$$

如果考虑整个训练集合上的平均损失，包括正则项，则公式如下：

$$L_i = \frac{1}{N} \sum_i \sum_{j \neq y_i} \max\left(0, W_j^T x_i - W_{y_i}^T x_i + \Delta\right) + \lambda \sum_k \sum_l W_{k,l}^2$$

Hinge loss 是一个 凸函数，所以很多常用的凸优化技术都可以使用。不过它是不可微的。Hinge Loss 的零区域对应的正是非支持向量的普通样本，从而所有的普通样本都不参与最终超平面的决定，这才是支持向量机最大的优势所在，对训练样本数目的依赖大大减少，而且提高了训练效率。

12. SVM 的 kernel，什么时候用什么 kernel

http://blog.csdn.net/batuwuhanpei/article/details/52354822

核函数目的是希望通过将输入空间内线性不可分的数据映射到一个高纬的特征空间内使得数据在特征空间内是可分的，我们定义这种映射为 ϕ(x)，那么我们就可以把求解约束最优化问题变为：

$$\begin{aligned} min_\alpha \quad & \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (\phi_i \cdot \phi_j) - \sum_{i=1}^N \alpha_i \\ s.t. \quad & \sum_{i=1}^N \alpha_i y_i = 0 \\ \alpha_i \geq 0, \quad & i = 1, 2, \ldots, N \end{aligned}$$

但是由于从输入空间到特征空间的这种映射会使得维度发生爆炸式的增长，因此上述约束问题中内积$\phi_i \cdot \phi_j$的运算会非常的大以至于无法承受，因此通常我们会构造一个核函数：

$$k(x, x_i) = \phi(x) \cdot \phi(x_i)$$

从而避免了在特征空间内的运算，只需要在输入空间内就可以进行特征空间的内积运算。通过上面的描述我们知道要想构造核函数$k$，我们首先要确定输入空间到特征空间的映射，但是如果想要知道输入空间到映射空间的映射，我们需要明确输入空间内数据的分布情况，但大多数情况下，我们并不知道自己所处理的数据的具体分布，故一般很难构造出完全符合输入空间的核函数，因此我们常用如下几种常用的核函数来代替自己构造核函数：

**线性核函数：**

$$k(x, x_i) = x \cdot x_i$$

线性核，主要用于线性可分的情况，我们可以看到特征空间到输入空间的维度是一样的，其参数少速度快，对于线性可分数据，其分类效果很理想，因此我们通常首先尝试用线性核函数来做分类，看看效果如何，如果不行再换别的。

**多项式核函数：**

$$k(x, x_i) = \left((x \cdot x_i) + 1\right)^d$$

多项式核函数可以实现将低维的输入空间映射到高纬的特征空间，但是多项式核函数的参数多，当多项式的阶数比较高的时候，核矩阵的元素值将趋于无穷大或者无穷小，计算复杂度会大到无法计算。

**高斯（RBF）核函数：**

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

高斯径向基函数是一种局部性强的核函数，其可以将一个样本映射到一个更高维的空间内，该核函数是应用最广的一个，无论大样本还是小样本都有比较好的性能，而且其相对于多项式核函数参数要少，因此大多数情况下在不知道用什么核函数的时候，优先使用高斯核函数。

**Sigmoid 核函数：**

$$k(x, x_i) = \tanh(x \cdot x_i + c)$$

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。

因此，在选用核函数的时候，如果我们对我们的数据有一定的先验知识，就利用先验来选择符合数据分布的核函数；如果不知道的话，通常使用交叉验证的方法，来试用不同的核函数，误差最下的即为效果最好的核函数，或者也可以将多个核函数结合起来，形成混合核函数。

在吴恩达的课上，也曾经给出过一系列的选择核函数的方法：

如果特征的数量大到和样本数量差不多，则选用 LR 或者线性核的 SVM；

如果特征的数量小，样本的数量正常，则选用 SVM+高斯核函数；

如果特征的数量小，而样本的数量很大，则需要手工添加一些特征从而变成第一种情况。

13. SVM、LR、决策树的对比

https://www.zhihu.com/question/21704547

https://www.zhihu.com/question/34735588

https://www.zhihu.com/question/26726794

**SVM 与 LR 的区别：**

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。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。两者的根本目的都是一样的。

但是逻辑回归相对来说模型更简单，好理解，实现起来，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。但是 SVM 的理论基础更加牢固，有一套结构化风险最小化的理论基础。还有很重要的一点，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量。

**决策树 (Decision tree)**

决策树的特点是它总是在沿着特征做切分。随着层层递进，这个划分会越来越细。虽然生成的树不容易给用户看，但是数据分析的时候，通过观察树的上层结构，能够对分类器的核心思路有一个直观的感受。

同时它也是相对容易被攻击的分类器[3]。这里的攻击是指人为的改变一些特征，使得分类器判断错误。常见于垃圾邮件躲避检测中。因为决策树最终在底层判断是基于单个条件的，攻击者往往只需要改变很少的特征就可以逃过监测。受限于它的简单性，决策树更大的用处是作为一些更有用的算法的基石。

适用场景：因为它能够生成清晰的基于特征(feature)选择不同预测结果的树状结构，数据分析师希望更好的理解手上的数据的时候往往可以使用决策树。

**随机森林 (Random forest)**

提到决策树就不得不提随机森林。顾名思义，森林就是很多树。严格来说，随机森林其实算是一种集成算法。它首先随机选取不同的特征(feature)和训练样本(training sample)，生成大量的决策树，然后综合这些决策树的结果来进行最终的分类。随机森林在现实分析中被大量使用，它相对于决策树，在准确性上有了很大的提升，同时一定程度上改善了决策树容易被攻击的特点。

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。

**SVM (Support vector machine)**

SVM 的核心思想就是找到不同类别之间的分界面，使得两类样本尽量落在面的两边，而且离分界面尽量远。

最早的 SVM 是平面的，局限很大。但是利用核函数(kernel function)，我们可以把平面投射(mapping)成曲面，进而大大提高 SVM 的适用范围。

适用场景：SVM 在很多数据集上都有优秀的表现。相对来说，SVM 尽量保持与样本间距离的性质导致它抗攻击的能力更强。和随机森林一样，这也是一个拿到数据就可以先尝试一下的算法。

**逻辑斯蒂回归 (Logistic regression)**

回归方法的核心就是为函数找到最合适的参数，使得函数的值和样本的值最接近。例如线性回归(Linear regression)就是对于函数$f(x)=ax+b$，找到最合适的$a,b$。LR 拟合的就不是线性函数了，它拟合的是一个概率学中的函数，$f(x)$的值这时候就反映了样本属于这个类的概率。

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。

**其他模型方法**

**近邻 (Nearest Neighbor)**

典型的例子是 KNN，它的思路就是——对于待判断的点，找到离它最近的几个数据点，根据它们的类型决定待判断点的类型。

适用场景：需要一个特别容易解释的模型的时候。比如需要向用户解释原因的推荐算法。

**贝叶斯 (Bayesian)**

典型的例子是 Naive Bayes，核心思路是根据条件概率计算待判断点的类型。是相对容易理解的一个模型，至今依然被垃圾邮件过滤器使用。

适用场景：需要一个比较容易解释，而且不同维度之间相关性较小的模型的时候。可以高效处理高维数据，虽然结果可能不尽如人意。

**判别分析 (Discriminant analysis)**

判别分析的典型例子是线性判别分析(Linear discriminant analysis)，简称 LDA。LDA 的核心思想是把高维的样本投射(project)到低维上，如果要分成两类，就投射到一维。要分三类就投射到二维平面上。这样的投射当然有很多种不同的方式，LDA 投射的标准就是让同类的样本尽量靠近，而不同类的尽量分开。对于未来要预测的样本，用同样的方式投射之后就可以轻易地分辨类别了。

适用场景：判别分析适用于高维数据需要降维的情况，自带降维功能使得我们能方便地观察样本分布。它的正确性有数学公式可以证明，所以同样是经得住推敲的方式。但是它的分类准确率往往不是很高，基本仅当做降维工具使用。

14. GBDT 和随机森林的区别

**随机森林特点：**

随机森林的随机性主要体现在两个方面：

数据集的随机选取：从原始的数据集中采取有放回的抽样（bagging），构造子数据集，子数据集的数据量是和原始数据集相同的。不同子数据集的元素可以重复，同一个子数据集中的元素也可以重复。

待选特征的随机选取：与数据集的随机选取类似，随机森林中的子树的每一个分裂过程并未用到所有的待选特征，而是从所有的待选特征中随机选取一定的特征，之后再在随机选取的特征中选取最优的特征。

以上两个随机性能够使得随机森林中的决策树都能够彼此不同，提升系统的多样性，从而提升分类性能。

随机森林的优点：

1) 实现简单，训练速度快，泛化能力强，可以并行实现，因为训练时树与树之间是相互独立的；

2) 相比单一决策树，能学习到特征之间的相互影响，且不容易过拟合；

3) 能处理高维数据（即特征很多），并且不用做特征选择，因为特征子集是随机选取的；

4) 对于不平衡的数据集，可以平衡误差；

5) 相比 SVM，不是很怕特征缺失，因为待选特征也是随机选取；

6) 训练完成后可以给出哪些特征比较重要。

随机森林的缺点：

1) 在噪声过大的分类和回归问题还是容易过拟合；

2) 相比于单一决策树，它的随机性让我们难以对模型进行解释。

**GBDT （Gradient Boost Decision Tree 梯度提升决策树）特点：**

GBDT 是以决策树为基学习器的迭代算法，注意 GBDT 里的决策树都是回归树而不是分类树。GBDT 的核心就在于：每一棵树学的是之前所有树结论和的残差，这个残差就是一个加预测值后能得真实值的累加量。

**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。

15. 随机森林有什么优点

见 14 题答案。

1) 对于数据集大时表现好，精确度比较高；
2) 不容易过拟合；
3) 可以得到变量的重要性排序；
4) 既能处理离散数据，也能处理连续性数据，且不需要归一化处理；
5) 能够很好的处理缺失数据；
6) 容易并行化。

16. 多分类怎么处理

- 1 vs 1
- 1 vs rest
- softmax

http://blog.csdn.net/supercally/article/details/54234115

假设我们有一个数组$V$，$V_i$表示$V$中的第$i$个元素，softmax 在指数域做了一个归一化(以保证之和为 1)，将输入向量映射为概率，这个元素的 Softmax 值就是

$$S_i = \frac{e^i}{\sum_j e^j}$$

定义交叉熵损失函数为：

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^j}\right)$$

对损失函数求偏导：

$$\begin{aligned}
&\frac{\partial L_i}{\partial f_{y_i}} = -\ln\left(\frac{e^{f_{y_i}}}{\sum_j e^j}\right)' \\
&= -1 * \frac{\sum_j e^j}{e^{f_{y_i}}} * \left(\frac{e^{f_{y_i}}}{\sum_j e^j}\right)' = -1 * \frac{\sum_j e^j}{e^{f_{y_i}}} * \left(1 - \frac{\sum_{j \neq f_{y_i}} e^j}{\sum_j e^j}\right)' \\
&= -1 * \frac{\sum_j e^j}{e^{f_{y_i}}} * (-1) * \sum_{j \neq f_{y_i}} e^j * (-1) * \frac{1}{\left(\sum_j e^j\right)^2} * \left(\sum_j e^j\right)' \\
&= -1 * \frac{\sum_j e^j}{e^{f_{y_i}}} * (-1) * \sum_{j \neq f_{y_i}} e^j * (-1) * \frac{1}{\left(\sum_j e^j\right)^2} * e^{f_{y_i}} \\
&= -\frac{\sum_{j \neq f_{y_i}} e^j}{\sum_j e^j} \\
&= -(1 - P_{f_{y_i}}) = P_{f_{y_i}} - 1
\end{aligned}$$

**SVM 与 Softmax 的区别：**

1) 原始像素数据映射得到的得分之后的处理方式不同，定义不同的损失函数，有不同的优化方法。Ps. SVM-Hinge Loss; Softmax-交叉熵；
2) SVM 下，我们能完成类别的判定，但是实际上我们得到的类别得分，大小顺序表示着所属类别的排序，但是得分的绝对值大小并没有特别明显的物理含义。Softmax 分类器中，结果的绝对值大小表征属于该类别的

概率。

17. 样本处理？

https://www.zhihu.com/question/30492527

在两类比例非常不均衡的情况下，就不能再用「分类正确率」(accuracy) 来衡量模型性能，而要用少数类的「准确率」(precision) 和「召回率」(recall)，或者二者的综合（F1, equal error rate, area under curve 等等)。在训练时，如果能够把这些评价指标作为目标函数，那是最好的。如果你的模型只支持用分类正确率作为目标函数，那么可以有下面几种对策：

1) 调整两类训练样本的权重，使得两类的总权重相等。这是最省事的办法。
2) 如果你的模型不支持类加权或样本加权，那么可以把少数类的数据复制几份，使得两类数据量基本相同。这与加权重是等效的，但是会浪费空间和时间。
3) 把少数类的数据复制几份，并适当加噪声。这可以增强模型的鲁棒性。
4) 加噪声可以拓展到一般的 data augmentation —— 根据已有数据生成新的数据，但保证类别相同。data augmentation 的方法有很多，不同性质的数据，augmentation 的方法也不一样。例如，图像数据可以平移、放缩、旋转、翻转等等。SMOTE
5) 如果多数类的数据量太大，也可以从中随机地取一小部分。当然，此方法一般要与上面 1~4 结合使用。

18. 正则化？

正则化的目的：防止过拟合！

正则化的本质：约束（限制）要优化的参数。

过多的变量（特征)，同时只有非常少的训练数据，会导致出现过度拟合的问题。因此为了解决过度拟合，有以下两个办法：

1) 尽量减少选取变量的数量：人工检查每一项变量，并以此来确定哪些变量更为重要，然后，保留那些更为重要的特征变量。
2) 正则化：正则化中我们将保留所有的特征变量，但是会减小特征变量的数量级（参数数值的大小$\theta_j$)

19. K-means 优缺点，K 的取值、改进

**K-means 概述**：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，直到质心不再改变，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。

**K-means 算法流程**：

- 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

K-means 优点：

1) 计算时间段，速度快
2) 容易解释
3) 聚类效果还不错

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，对异常值敏感。
2) 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-means 改进:

为了解决上述存在缺点，在基本 k-means 的基础上发展而来二分 (bisecting) k-means，其主要思想：一个大

cluster 进行分裂后可以得到两个小的 cluster；为了得到 k 个 cluster，可进行 k-1 次分裂。算法流程如下：

初始只有一个 cluster 包含所有样本点；
repeat:
从待分裂的 clusters 中选择一个进行二元分裂，所选的 cluster 应使得 SSE 最小；
until 有 k 个 cluster

20. Hadoop 怎么实现 K-Means？
http://www.open-open.com/doc/view/d4657e719c6f45e98e5ffd79aed3a613

21. 机器学习算法中的距离度量有哪些，一般在什么场景下用？
http://blog.csdn.net/jbfsdzpp/article/details/48497347

**欧氏距离：**

$$Distance(O_i, O_j) = \sqrt{\sum_{k=1}^{n}(O_{ik} - O_{jk})^2}$$

欧氏距离可以简单的描述为多维空间的点点之间的几何距离，通常采用的是原始数据，而并非规划化后的数据，比如有一属性在 1-100 内取值，那么便可以直接使用，而并非一定要将其归一到[0,1]区间使用。这样的话，欧式距离的原本意义便被消除了，正是因为这样，所以其优势在于新增对象不会影响到任意两个对象之间的距离。然而，如果对象属性的度量标准不一样，如在度量分数时采取十分制和百分制，对结果影响较大。

**曼哈顿距离：**

$$Distance(P_i, P_j) = \frac{1}{n}\sum_{k=1}^{n}|P_{ik} - P_{jk}|$$

曼哈顿距离是计算从一个对象到另一个对象所经过的折线距离，有时也可以进一步的描述为多维空间中对象在各维的平均差。需要注意的是，曼哈顿距离取消了欧式距离的平方，因此使得离群点的影响减弱。

**切比雪夫距离：**

$$Distance(Q_i, Q_j) = Max_{k=1}^{n}(\mathrm{Q_{ik}} - \mathrm{Q_{jk}})$$

切比雪夫距离主要表现为在多维空间中，对象从某个位置转移到另外一个对象所消耗的最少距离。因此可以简单的描述为用一维属性决定某对象属于哪个簇，这就好比我们去辨别一项罕见的现象一样，如果两个对象都存在这一罕见现象，那么这两个对象应该属于同一个簇。

**幂距离：**

$$Distance(R_i, R_j) = r\sqrt{\sum_{k=1}^{n}(|R_{ik} - R_{jk}|)^p}$$

幂距离可以简单的描述为针对不同的属性给予不同的权重值，决定其属于那个簇。$r, p$为自定义的参数，根据实际情况选择，其中，$p$是用来控制各维的渐进权重，r 控制对象间较大差值的渐进权重。当$r = p$时，即为闵可夫斯基距离，当$p = r = 1$时为曼哈顿距离，当$p = r = 2$时为欧式距离，当$p = r$并趋于无穷时即为切比雪夫距离(可以用极限理论证明).因此，这几种距离统称为闵氏距离，闵氏距离的不足在于：从横向(各维)看，它等同的看待了不同不同的分量，这种缺陷从切比雪夫距离中可以明显看出，忽略了不同维的差异。从纵向(单维)看，它忽略了不同维的各对象的分布差异，这种差异在统计学中可以用期望，方差，标准差等度量。

**余弦相似度：**

$$Distance(S_i, S_j) = Cos(\overrightarrow{S_i}, \overrightarrow{S_j}) = \frac{\sum_{k=1}^{n} S_{ik}S_{jk}}{\sum_{k=1}^{n} S_{ik}^2 \sum_{k=1}^{n} S_{jk}^2}$$

余弦相似度可以简单的描述为空间中两个对象的属性所构成的向量之间的夹角大小。即当两个向量方向完全相

同时，相似度为 1，即完全相似，当两个向量方向相反时，则为-1，即完全不相似。

**皮尔逊相似度：**

$$sim(a,b)=\frac{\sum_{c\in a\cap b}(R_{a,c}-\overline{R_a})\sum_{c\in a\cap b}(R_{b,c}-\overline{R_b})}{\sqrt{\sum_{c\in a\cap b}\left(R_{a,c}-\overline{R_a}\right)^2}\sqrt{\sum_{c\in a\cap b}\left(R_{b,c}-\overline{R_b}\right)^2}}$$

其中 c 为共有属性。

皮尔逊相似度可以描述为不同对象偏离拟合的中心线程度，可以进一步的理解为，许多对象的属性拟合成一条直线或者曲线，计算每个对象相对于这条线的各属性偏离程度。

**Jaccard 相似度：**

$$sim(a,b)=\frac{|a\cap b|}{|a\cup b|}=\frac{|a\cap b|}{|a\cap\bar{b}|+|\bar{a}\cap b|+|a\cap b|}$$

Jaccard 相似度常用于二值型数据的相似度计算。，在数据挖掘中，经常将属性值二值化，通过计算 Jaccard 相似度，可以简单快速的得到两个对象的相似程度。当然，对于不同的属性，这种二值型程度是不一样的，比如，在中国，熟悉汉语的两个人相似度与熟悉英语的两个人相似度是不同的，因为发生的概率不同。所以通常会对 Jaccard 计算变换，变换的目的主要是为了拉开或者聚合某些属性的距离。

**汉明距离：**

可以描述为将同等长度的字符串由其中一个变换到另一个的最小替换次数。如将 a(11100)变换为 b(00010)，则其距离为 4，汉明距离主要是为了解决在通信中数据传输时，改变的二进制位数，也称为信号距离。

**相关距离：**

$$\rho_{xy}=\frac{Cov(X,Y)}{\sqrt{D(X)}\sqrt{D(Y)}}=\frac{E((X-EX)(Y-EY))}{\sqrt{D(X)}\sqrt{D(Y)}}$$

相关距离是用来衡量随机变量 X,Y 之间的相关程度的一种方法，取值为[-1,1]，且系数越大，相关度越高。当 X,Y 线性相关时，若为正线性相关时则为 1，相关距离=1-相关系数。

22. 解释贝叶斯公式和朴素贝叶斯分类

http://blog.csdn.net/han_xiaoyang/article/details/50616559

http://blog.csdn.net/han_xiaoyang/article/details/50629587

http://blog.csdn.net/han_xiaoyang/article/details/50629608

贝叶斯公式：

$$P(Y|X)=\frac{P(X|Y)P(Y)}{P(X)}$$

贝叶斯公式 + 条件独立假设 = 朴素贝叶斯方法

**朴素贝叶斯优点：**

1) 对待预测样本进行预测，过程简单速度快(想想邮件分类的问题，预测就是分词后进行概率乘积，在 log 域直接做加法更快)。
2) 对于多分类问题也同样很有效，复杂度也不会有大程度上升。
3) 在分布独立这个假设成立的情况下，贝叶斯分类器效果奇好，会略胜于逻辑回归，同时我们需要的样本量也更少一点。
4) 对于类别类的输入特征变量，效果非常好。对于数值型变量特征，我们是默认它符合正态分布的。

**朴素贝叶斯缺点：**

1) 对于测试集中的一个类别变量特征，如果在训练集里没见过，直接算的话概率就是 0 了，预测功能就失效了。当然，我们前面的文章提过我们有一种技术叫做『平滑』操作，可以缓解这个问题，最常见的平滑技术是拉普拉斯估测。
2) 朴素贝叶斯算出的概率结果，比较大小还凑合，实际物理含义…恩，别太当真。

3) 朴素贝叶斯有分布独立的假设前提，而现实生活中这些 predictor 很难是完全独立的。

**最常见应用场景**

- **文本分类/垃圾文本过滤/情感判别：**这大概会朴素贝叶斯应用做多的地方了，即使在现在这种分类器层出不穷的年代，在文本分类场景中，朴素贝叶斯依旧坚挺地占据着一席之地。原因嘛，大家知道的，因为多分类很简单，同时在文本数据中，分布独立这个假设基本是成立的。而垃圾文本过滤(比如垃圾邮件识别)和情感分析(微博上的褒贬情绪)用朴素贝叶斯也通常能取得很好的效果。
- **多分类实时预测：这**个是不是不能叫做场景？对于文本相关的多分类实时预测，它因为上面提到的优点，被广泛应用，简单又高效。
- **推荐系统：**是的，你没听错，是用在推荐系统里！！朴素贝叶斯和协同过滤(Collaborative Filtering)是一对好搭档，协同过滤是强相关性，但是泛化能力略弱，朴素贝叶斯和协同过滤一起，能增强推荐的覆盖度和效果。

朴素贝叶斯注意点：

- 大家也知道，很多特征是连续数值型的，但是它们不一定服从正态分布，一定要想办法把它们变换调整成满足正态分布！！
- 对测试数据中的 0 频次项，一定要记得平滑，简单一点可以用『拉普拉斯平滑』。
- 先处理处理特征，把相关特征去掉，因为高相关度的 2 个特征在模型中相当于发挥了 2 次作用。
- 朴素贝叶斯分类器一般可调参数比较少，比如 scikit-learn 中的朴素贝叶斯只有拉普拉斯平滑因子 alpha，类别先验概率 class_prior 和预算数据类别先验 fit_prior。模型端可做的事情不如其他模型多，因此我们还是集中精力进行数据的预处理，以及特征的选择吧。
- 那个，一般其他的模型(像 logistic regression，SVM 等)做完之后，我们都可以尝试一下 bagging 和 boosting 等融合增强方法。咳咳，很可惜，对朴素贝叶斯里这些方法都没啥用。原因？原因是这些融合方法本质上是减少过拟合，减少 variance 的。朴素贝叶斯是没有 variance 可以减小。

23. 如何进行特征选择？

http://blog.csdn.net/datoutong_/article/details/78813233

当数据预处理完成后，需要选择有意义的特征输入机器学习的算法和模型进行训练。通常来说，从两个方面考虑来选择特征：

- **特征是否发散：**如果一个特征不发散，例如方差接近于 0，也就是说样本在这个特征上基本上没有差异，这个特征对于样本的区分并没有什么用。
- **特征与目标的相关性：**这点比较显见，与目标相关性高的特征，应当优选选择

根据特征选择的形式又可以将特征选择方法分为 3 种：

- **Filter :过滤法**，按照发散性或者相关性对各个特征进行评分，设定阈值或者待选择阈值的个数，选择特征。
- **Wrapper：包装法**，根据目标函数（通常是预测效果评分），每次选择若干特征，或者排除若干特征。
- **Embedded：嵌入法**，先使用某些机器学习的算法和模型进行训练，得到各个特征的权值系数，根据系数从大到小选择特征。类似于 Filter 方法，但是通过训练来确定特征的优劣。

特征选择主要有两个目的：

- 减少特征数量、降维，使模型泛化能力更强，减少过拟合；
- 增强对特征和特征值之间的理解。

**Filter：过滤法:**

- 评估单个特征和结果值之间的相关程度，排序留下 Top 相关的特征部分。
- Pearson 相关系数，互信息，距离相关度
- 缺点：没有考虑到特征之间的关联作用，可能把有用的关联特征误踢掉。

**Wrapper：包装法**

- 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。
- 典型的包裹型算法为“递归特征删除算法”(recursivefeatureeliminationalgorithm)
- 比如用逻辑回归，怎么做这个事情呢？
  ① 用全量特征跑一个模型
  ② 根据线性模型的系数(体现相关性)，删掉 5-10%的弱特征，观察准确率/auc 的变化
  ③ 逐步进行，直至准确率/auc 出现大的下滑停止

**Embedded：嵌入法**

使用 SelectFromModel 选择特征 (Feature selection using SelectFromModel)

单变量特征选择方法独立的衡量每个特征与响应变量之间的关系，另一种主流的特征选择方法是基于机器学习模型的方法。有些机器学习方法本身就具有对特征进行打分的机制，或者很容易将其运用到特征选择任务中，例如回归模型，SVM，决策树，随机森林等等。其实 Pearson 相关系数等价于线性回归里的标准化回归系数。

- 根据模型来分析特征的重要性（有别于上面的方式，是从生产的模型权重等）。
- 最常见的方式为用正则化方式来做特征选择。
- 举个例子，最早在电商用 LR 做 CTR 预估，在 3-5 亿维的系数特征上用 L1 正则化的 LR 模型。剩余 2-3 千万的 feature，意味着其他的 feature 重要度不够。

24. 为什么会产生过拟合，有哪些方法可以缓解过拟合？

**过拟合产生的原因：**

出现这种现象的主要原因是训练数据中存在噪音或者训练数据太少。

**预防或克服措施：**

1、 增大数据量
2、 减少 feature 个数（人工定义留多少个 feature 或者算法选取这些 feature）
3、 正则化（留下所有的 feature，但对于部分 feature 定义其 parameter 非常小）
4、 交叉验证

25. 你用过哪些机器学习、数据挖掘工具或者框架？

Numpy, scipy, pandas, sklearn, xgboost, keras/Tensorflow

26. 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？

EM 算法也称期望最大化（Expectation-Maximum,简称 EM）算法。

EM 算法是一种迭代算法，用于含有隐变量的概率模型参数的极大似然估计。两个步骤交替计算：

E 步：利用当前估计的参数值，求出在该参数下隐含变量的条件概率值（计算对数似然的期望值）；

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的最大值（寻找能使 E 步产生的似然期望最大化的参数值。）然后，新得到的参数值重新被用于 E 步.....直到收敛到局部最优解。（note：每次迭代实际在求 Q 函数及其极大，即每次迭代使似然函数增大或达到局部极值。）

**优点**：简单性和普适性，可看作是一种非梯度优化方法（解决梯度下降等优化方法的缺陷：求和的项数将随着隐变量的数目以指数级上升，会给梯度计算带来麻烦）

**缺点**：对初始值敏感，不同的初值可能得到不同的参数估计值；不能保证找到全局最优值。

采用 EM 算法的模型有一般有混合高斯、协同过滤、k-means。

算法一定会收敛，但是可能会收敛到局部最优。求和的项数会随着隐变量的数目指数上升，会给梯度计算带来麻烦。EM 算法是一种非梯度优化算法。

EM 算法在高斯混合模型中的应用：

（1）明确隐变量，写出完全数据的对数似然函数。

（2）EM 算法的 E 步：确定 Q 函数（即：完全数据的对数似然函数关于在给定观测数据和参数的条件下对隐变量的条件概率的期望）：

（3）M 步：求 Q 函数对 theta 的极大值，即求新一轮迭代的模型参数。

EM 算法推导解释 K-means：

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。k-means 算法与 EM 算法的关系是这样的：

- k-means 是两个步骤交替进行:确定中心点，对每个样本选择最近中心点--> E 步和 M 步。
- E 步中将每个点选择最近的类优化目标函数，分给中心距它最近的类(硬分配)，可以看成是 EM 算法中 E 步(软分配)的近似。
- M 步中更新每个类的中心点，可以认为是在「各类分布均为单位方差的高斯分布」的假设下，最大化似然值；

27. 主题模型里 LDA 的原理和推导（腾讯）

**1CxuX5cAZeTtGPU9EZUQr3trzymkAT6Qk5**

**1CxuX5cAZeTtGPU9EZUQr3trzymkAT6Qk5**

28. 做广告点击预测，用哪些数据什么算法（BAT）？

LR、GBDT、FM、FFM、NN

**http://blog.csdn.net/xiewenbo/article/details/52038493**

**FM**

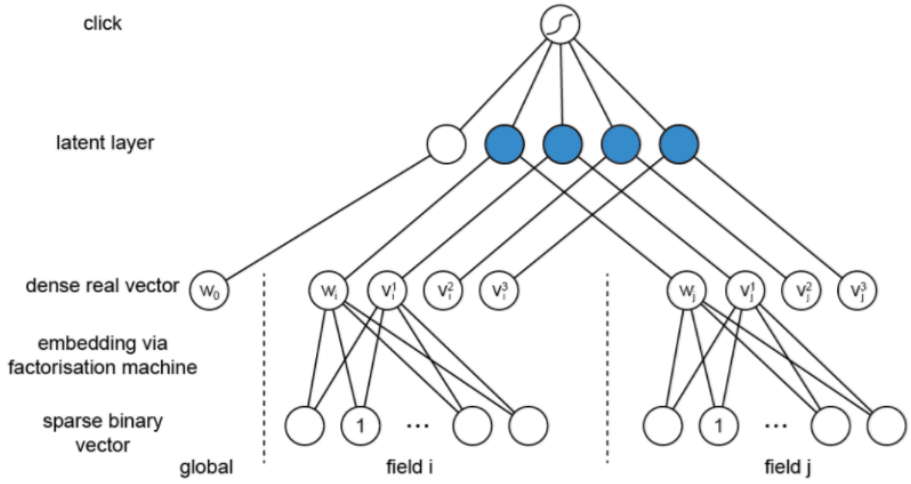
因为上述原因，我们需要将非常大的特征向量嵌入到低维向量空间中来减小模型复杂度，而FM（Factorisation machine）无疑是被业内公认为最有效的embedding model：

$$y_{\mathrm{FM}}(\boldsymbol{x}) := \mathrm{sigmoid}\Big(\underbrace{w_0 + \sum_{i=1}^{N} w_i x_i}_{\text{Logistic Regression}} + \underbrace{\sum_{i=1}^{N}\sum_{j=i+1}^{N} \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle x_i x_j}_{\text{Feature Interactions}}\Big)$$

第一部分仍然为Logistic Regression，第二部分是通过两两向量之间的点积来判断特征向量之间和目标变量之间的关系。比如上述的迪斯尼广告，occupation=Student和City=Shanghai这两个向量之间的角度应该小于90，它们之间的点积应该大于0，说明和迪斯尼广告的点击率是正相关的。这种算法在推荐系统领域应用比较广泛。

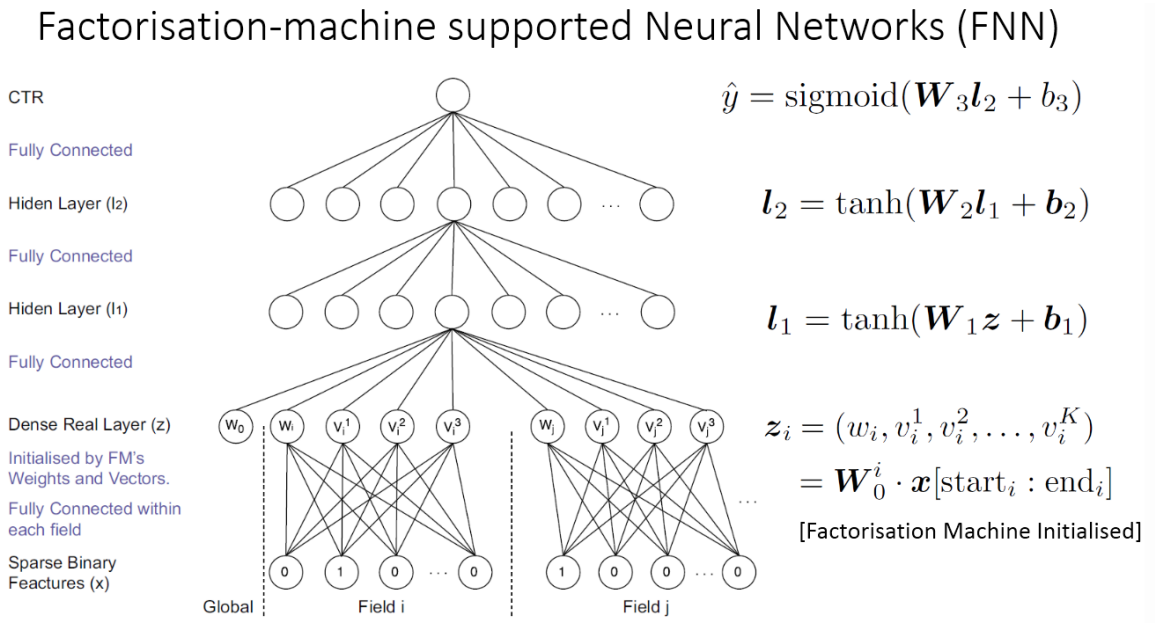
**FNN**

那我们就基于这个模型来考虑神经网络模型，其实这个模型本质上就是一个三层网络：



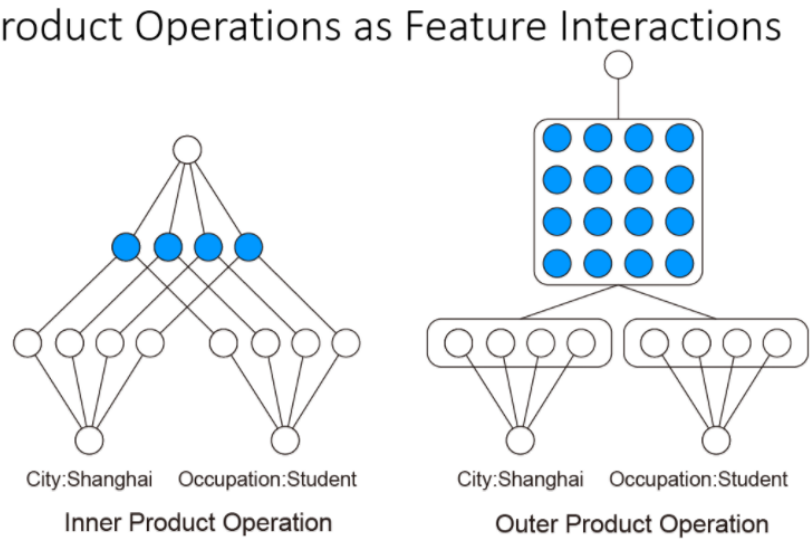

它在第二层对向量做了乘积处理（比如上图蓝色节点直接为两个向量乘积，其连接边上没有参数需要学习），每个field都只会被映射到一个 low-dimensional vector，field和field之间没有相互影响，那么第一层就被大量降维，之后就可以在此基础上应用神经网络模型。

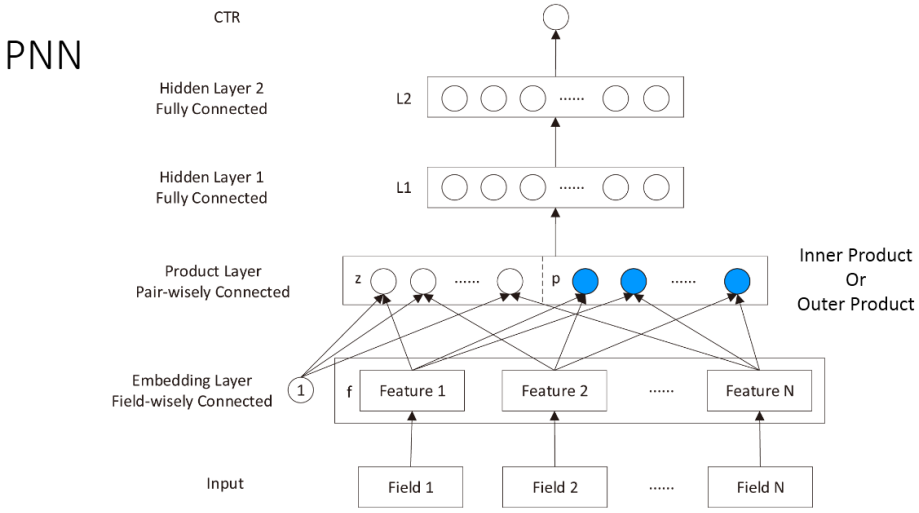
我们用FM算法对底层field进行embeddding，在此基础上面建模就是FNN(Factorisation-machine supported Neural Networks)模型：




PNN

对两个向量做内积和外积的乘法操作




29. 推荐系统的算法中最近邻和矩阵分解各自使用场景（AT）

30. 用户流失率预测怎么做（游戏公司、外卖公司等）

31. 线性分类器与非线性分类器的区别及优劣

32. 特征比数据量还大时，选择什么样的分类器？

33. 对于维度很高的特征，你选择线性还是非线性分类器？

34. 对于维度极低的特征，你选择线性还是非线性分类器？

35. L1 和 L2 正则的区别，如何选择 L1 和 L2 正则？

36. 随机森林的学习过程？

37. 随机森林中的每一颗树是如何学习的?

38. 随机森林算法中的 CART 树的基尼指数是什么？

补充：

39. xgboost 中各主要参数含有？

Xgboost 是一个开源软件库，提供了 C++、JAVA、Python 等接口，具有伸缩性、可移动性、并发性等特点。且由于其训练速度快，效果好的特性，在许多数据挖掘比赛中崭露头角。

相较于其他基于 bagging 策略的决策树而言，基于 boosting 策略的 Xgboost 的准确率和训练时间有显著的提高。Xgboost 实现的是一种通用的 Tree boosting 策略，此算法中有个经典的子算法梯度提升决策树（GBDT），其原理为首先采用训练集和样本真值训练一棵树，然后采用这棵树预测训练集，由于预测值和真实值存在偏差，所以二者相减可以得到残差，接下来训练第二颗树，此时不再使用真值，而使用其残差作为标签，两棵树训练完成后，可以进一步得到新的残差，然后训练第三棵树，使残差一步步缩小。树的总量可以人为指定，也可以通过监控某些指标来停止训练。

而相较于 GBDT 而言，Xgboost 有许多改进的地方。**一是将其损失函数从平方损失推广到二阶可导的损失**，GBDT 使用平方损失函数可以使残差一步步逼近于零，但是如果换用其它损失函数，则使用残差将不再能够保证。GBDT 使用的残差是泰勒展式到一阶的结果，而 Xgboost 是展开到二阶的结果。**二是加入了正则化项**，可以控制模型的复杂程度，模型越复杂，对于训练集的误差可以很低，但是对于测试集的误差却很高，越容易发生过拟合(overfitring)，Xgboost 加入了 L2 正则项，可以有效控制模型的复杂度。**三是剪枝策略**，Xgboost 会一直分裂到最大深度，然后进行剪枝，如果发现某个节点之后不再有正损失，就去除这个分裂，这是一种全局最优的策略。

**参数调优介绍**。

A. 通用参数介绍：
   1. booster[default: gbtree]：选择每次迭代的模型，gbtree 梳妆模型，gbliner 线性模型。
   2. silent[default:0]：是否开启静默模式。
   3. nthread[default: max]：选择线程数。

B. booster 参数介绍：
   1. eta[default: 0.3]：步长，减少每一步的权重，提高模型鲁棒性。
   2. min_child_weight[default: 1]：决定最小叶子节点样本权重和，避免过拟合。
   3. max_depth[default: 6]：树的最大深度，防止过拟合。
   4. max_leaf_nodes：树的最大节点或叶子数量（此参数会掩盖 max_depth）。
   5. gamma[default: 0]：指定了节点分裂所需要最小函数下降值。
   6. max_delta_step[default: 0]：限制每棵树权重改变的最大步长。
   7. subsample[default: 1]：对于每棵树随机采样的比例，避免过拟合。
   8. colsample_bytree[default: 1]：控制每棵树随机采样列数的占比。
   9. colsample_bylevel[default: 1]：控制树的每一级的每一次分裂对列数的采样占比。
   10. lambda[default: 1]：权重的 L2 正则化项。
   11. alpha[default: 1]：权重的 L1 正则化项。
   12. scale_pos_weight[default: 1]：当样本十分不平衡时，设定一个正值使其更快收敛。

40. 决策树、随机森林是什么样的模型？

41. 逻辑回归与线性回归的区别是什么？

http://blog.csdn.net/u010692239/article/details/52345754

https://www.cnblogs.com/tbcaaa8/p/4415429.html

线性回归的样本的输出，都是连续值，$y\in(+\infty,-\infty)$而，逻辑回归中 $y\in\{0,1\}$，只能取 0 和 1。

**线性回归：**

$$f(x) = \theta^T X = \theta_1 \mathrm{x}_1 + \theta_2 \mathrm{x}_2 + \cdots + \theta_n \mathrm{x}_\mathrm{n}$$

定义线性回归的损失函数为：

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

要使似然函数最大，只需使损失函数最小。使损失函数最小，使用损失函数的极小值代替最小值，只需对每一个求偏导数：

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

最后，使用梯度下降迭代求解：

$$\theta_j^{(k+1)} = \theta_j^{(k)} - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

**逻辑回归：**

$$f(x) = p(y = 1|x;\theta) = g(\theta^T X)$$

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

通过回归结果可以看出，线性回归的拟合函数，的确是对 f(x)的输出变量 y 的拟合，而逻辑回归的拟合函数是对为 1 类的样本的概率的拟合。

逻辑回归函数的对数似然函数为：

$$lnL(y|x;\theta) = \sum_{i=1}^{m} lnP(y^{(i)}|x^{(i)};\theta)$$

化简得：

$$lnL(y|x;\theta) = \sum_{i=1}^{m}(y^{(i)}\ln\left(h_\theta\left(x^{(i)}\right)\right) + (1-y^{(i)})\ln\left(1-h_\theta(x^{(i)})\right))$$

定义损失函数为：

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\ln\left(h_\theta\left(x^{(i)}\right)\right) + (1-y^{(i)})\ln\left(1-h_\theta(x^{(i)})\right))$$

要使似然函数最大，只需使损失函数最小。我们使用损失函数的极小值代替最小值，只需对每一个$\theta_j$求偏导数：

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

最后，使用梯度下降迭代法求解：

$$\theta_j^{\left((k+1)\right)} = \theta_j^{(k)} - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

补充问题：

3、数据挖掘算法。

（1）逻辑回归和线性回归区别，逻辑回归应用场景，其中的核函数有什么作用

（2）什么是梯度下降，梯度下降优化，你知道哪些

（3）神经网络中，你觉得 NN 最大的优势在哪里，其中的更新函数是什么

关于二分查找的非递归实现，然后就是九宫格算法

聚类算法的优缺点，缺点怎么优化？怎么选取初始点？