统计学习简介

统计推断基本步骤：选择模型和推导统计量，统计量评估，收集数据，根据统计量进行推断

统计学习基本步骤：选择模型和指定策略，收集数据，算法，模型评估

- 模型：统计推断模型就是概率分布，而统计学习模型的本质仍是概率分布，但是因为我们的目的并不再是求分布，而是预测（输出），所以统计学习中模型要不就是统计推断模型后面加上 $argmax$ ，要不就是全新模型（比如神经网络），使得模型本身成为输入输出的映射函数。

  注意：任何数据本质都是概率分布，所以任何形式的映射函数拟合所学习的内容都是在当前 x 下，哪一个 y 是最优值，但正因为存在概率，不同于预测值的 y' 是完全有可能出现的，这也是损失函数很少见值为0的原因

- 策略：对应推导统计量。统计推断中，推导统计量这一步为：我们基于一些假设（比如最大似然），构造函数，并且认为使得函数满足某些特征的参数值是真值，也就是 $min\ or\ max f(x)$ 得到 $\hat{\theta} = g(X)$ 。而在统计学习中，有两点不同：其一，假设不仅仅限于统计推断中的那些，也就是函数形式 $f(x)$ 会随着问题和我们寻找最优参数的思路而变化；其二，函数 $f(x)$ 形式可能会较为复杂，无法推导出参数关于数据的解析式 $\hat{\theta} = g(X)$ 。那么策略得到的结果就不再是一个参数解析式，而是一个优化目标函数。

- 算法：对应根据统计量进行推断。正因为在策略中，我们得到一个优化目标：θ 会满足 $min\ or\ max f(x)$，根据统计量进行推断这一步就变成了最优化（使用算法找极值点），著名的梯度下降也就是在这里出现。
- 模型评估：统计推断中，我们对统计量进行评估放在收集数据之前，这是因为模型不成问题，我们只需要在这基础上导出一个最能提取参数信息的统计量，而统计量作为一个随机变量是可以通过分布的性质进行评估的。

  而统计学习中，我们不对策略（构造函数的假设）提出质疑，认为只要模型合理，算出来的结果应该就是正确的，因此我们注意力全部放在模型挑选上面（模型简单了还是复杂了？当前问题选取这个这个模型对不对？）并且我们手里没有统计量，而只有优化目标函数，那么想法就是：将测试集数据带入优化目标函数，看我们找到参数在测试集上是否仍是最优化点或是其附近点。

  统计量评估和模型评估如此相像，这应该也是从点估计当中借鉴方差和偏差的原因

统计推断和统计学习的根本不同就是统计量变成了优化目标函数

需要注意的是，上面论述的是统计学习和统计推断不同的部分，但是它们仍存在相同的部分，在形式上，某些模型中能够导出统计量，那就不用最优化了

统计决策理论是统计推断中的一个小知识点，邵军和Trevor Hastie都没有很大篇幅的提及，可以认为统计推断中的统计决策理论的现代化，就是统计学习，上述的统计推断和统计学习的区别应该也就是统计推断和统计决策理论的区别

Chapter 1 模型分类

从不同的角度，可以给出统计学习的多种分类方法：

- 第一种：根据输入输出进行分类
  - 监督学习：数据遵循联合概率分布 $P(X,Y)$
    - 生成模型：学习联合概率 $P(X,Y)$，朴素贝叶斯，隐马尔可夫

        - 判别模型：学习条件概率 $P(Y|X)$ ，k近邻，感知机，决策树，逻辑斯蒂回归，最大熵，支持向量机，提升方法，条件随机场

    或

        - 分类
        - 回归
    - 无监督学习：学习 $P(X)$
    - 强化学习
    - 半监督学习和主动学习

- 第二种：根据模型内部结构进行分类，概率模型一定可以表示成联合概率分布的形式，而非概率模型则不一定
    - 概率模型：决策树，朴素贝叶斯，隐马尔可夫模型，条件随机场，概率潜在语义分析，潜在狄利克雷分配，高斯混合模型，逻辑斯蒂回归
    - 非概率模型：感知机，支持向量机，k近邻，AdaBoost，k均值，潜在语义分析，神经网络，逻辑斯蒂回归

- 第三种：根据模型内部结构进行分类
    - 线性模型：感知机，线性支持向量机，k近邻，k均值，潜在语义分析
    - 非线性模型：核函数支持向量机，AdaBoost，神经网络

- 第四种：根据模型内部结构进行分类，参数模型假定模型可以由有限维参数完全刻画，非参数模型假设模型参数维度不固定（随着训练数据增大而增大）
    - 参数模型：感知机，朴素贝叶斯，逻辑斯蒂回归，k均值，高斯混合模型
    - 非参数模型：决策树，支持向量机，AdaBoost，k近邻，潜在语义分析，概率潜在语义分析，潜在狄利克雷分配

- 第五种：根据使用技巧进行分类
    - 贝叶斯学习：朴素贝叶斯，潜在狄利克雷分配
    - 核方法

- 第六种：根据算法分类
    - 在线学习
    - 批量学习

面试问题

- 生成模型和判别模型都有哪些？区别是什么？ 前者学习联合概率分布，后者学习条件概率分布，属于前者的时朴素贝叶斯和隐马尔可夫，而其他大多都是后者

Chapter 2 监督学习

基本假设：联合概率分布 $P(X,Y)$ 存在

不管是生成模型还是判别模型，最终目的是得到给 x 算 y 的步骤，这个步骤可以通过 $\underset{c}{argmax}\ p(y=c|x,\theta)$

Chapter 3 无监督学习

基本假设：概率 $P(X)$ 存在

无监督学习学的是输入随机变量的概率分布，可是用到这样原始形式的场景几乎没有，那么我们还要根据具体目的对上述函数进行推导。

比如我想看数据有几个cluster，那么cluster个数随机变量 K 很明显是一个统计量，它的分布为 $p(k|\theta)$ ；在我们cluster数量选定后，再进一步看每一个点都在都在哪个cluster中，包含 x 的cluster的序号 Z 是 X 的条件分布，它的分布为 $p(z|x,k,\theta)$ 。这里只是对推导进行简单演示，无监督学习中有很复杂的统计量。

而在使用中，同样为

$$argmax\ p$$

Chapter 4 过拟合与欠拟合

统计推断的前提是模型确定，参数值未知，我们需要通过随机样本去推断参数值，Casella整本书都是基于这一点进行讨论的，但这是很理想的情况。现实生活中，对于一个问题，我们掌握的可能只有随机样本观测值，我们就需要去选择模型，而这一点就会带来过拟合和欠拟合

统计学习的根源虽然在统计推断，但它和计量经济学一样，是统计推断的实际应用，那么在推断之前，肯定会遇到模型选择的问题，甚至在Kaggle上面，这成了重中之重。

首先我们假定观测数据为有限值（不涉及渐进统计），并且产生数据的真实模型属于分布族A，参数为 θ_A ，统计量为 $T_A(X)$。如果我们挑选了一个分布族B，它比A要复杂（狭义理解为参数较多）并且形式上包含A（意思是去掉B中某些项，分布族形式能够变成A，而去不去掉可以由参数控制），参数为 θ_B ，统计量为 $T_B(X)$ 。这样设定是符合实际情况的，因为现实中虽然我们并不知道分布族的真实形式，但是我们可以从过往经验和当前问题的具体形式猜个大概。

正确情况就是 $T_A(X)$ 提取到了 θ_A 全部信息，但现在因为我选择了复杂模型B，导致 $T_B(X)$ 提取出的信息不仅包括（部分） θ_A 信息，还包括数据的波动信息，最终将他们一股脑全灌给 θ_B 。这就会导致训练出的模型出色拟合训练集数据，而在测试集上效果较差，这就是过拟合。

而欠拟合则是我们挑选的模型过于简单，那么在进行参数推断的时候，θ_A 的信息残缺，那么肯定训练集和测试集数据都拟合不好。

统计当中的分布族选取在统计学习中就是模型的选取，当然也就包括和模型有关的超参数的选取，比如神经网络的层数，每层的神经元的个数等

过拟合需要减少模型复杂度，统计学习做法是加正则化项（使参数有往0走的动力），我的理解是当 θ_B 中不属于 θ_A 的那部分趋近于零，那么 $\theta_B \approx \theta_A$ 。或是加大数据量。

欠拟合需要增加复杂度，一般是重新选择模型

过拟合对应高方差，低偏差；欠拟合对应低方差，高偏差。这里所说的方差与偏差和统计学点估计评价点估计量涉及到的偏差和方差应该不是一个概念。但是这两个名词所代表的意思是确定的，方差代表围绕均值的波动程度，偏差代表均值与真值的偏离程度

过拟合一般用正则化能够缓解，它分为L1和L2正则化，他们的普遍形式为

$$L_p = (\sum X_i^p)^{\frac{1}{p}}$$

当p=1时为L1范数

$$L_1 = \sum |X_i|$$

当p=2时为L2范数

$$L_2 = \sqrt{\sum X_i^2}$$

在统计学习中，L2范数的形式还可以是

$$L_2 = \sum X_i^2$$

暂时不知道为什么

把他们加到损失函数后面就可以构成L1正则化和L2正则化，因为他们作为惩罚项可以收缩参数

而他们的区别为（面试常考）

- L1会趋向于产生少量的特征，而其他的特征都是0。因为最优的参数值很大概率出现在坐标轴上，这样就会导致某一维的权重为0 ，产生稀疏权重

矩阵

- L2会选择更多的特征，这些特征都会接近于0。最优的参数值很小概率出现在坐标轴上，因此每一维的参数都不会是0。当最小化||w||时，就会使每一项趋近于0。

4.1. 面试问题

如何判断过拟合和欠拟合？看损失函数在训练集和测试集上的表现，如果在训练集上损失小而在测试集上损失大就表明模型过拟合；如果在训练集上损失大而在测试集上损失也大就表明模型欠拟合；如果在训练集上损失大而在测试集上损失也大，同时后者比前者的数值大很多，就代表模型既方差大又偏差大。

如何解决过拟合和欠拟合？看上述文字

处理过拟合的具体手段？l1，l2 ，神经网络dropout，增加数据量，bagging

策略

这里主要介绍我们针对一个模型，明确定义它的好坏衡量标准，最终的结果是一个目标优化函数，只要模型参数使得这个函数值尽可能地小，我们就认为这个模型是好的

损失函数：$L(Y, f(X))$ 衡量模型输出值和真实值之间的差别

风险函数：或称期望损失，$R_{exp}(f) = \int L(y, f(x))P(x, y)dxdy$

经验风险：经验损失，$R_{emp}(f) = \frac{1}{N}\sum L(y_i, f(x_i))$

- 经验风险最小化：$min\frac{1}{N}\sum L(y_i, f(x_i))$
- 结构风险最小化：$min\frac{1}{N}\sum L(y_i, f(x_i)) + \lambda J(f)$

上面所述是损失函数的基本框架，那么接下来就要寻找 L 的具体形式了，而 L 的具体形式是根据模型特性来的，比如线性回归用最小二乘法，逻辑回归用交叉熵，等等

从本质上来说，策略和模型评估本质上是一样的，不同的是在策略层面上，我们是得到模型的好坏量化值，并用这个值去改进参数，而模型评估层面，得到了一个好坏量化值之后，我们是用这个值去挑选模型（比如两个模型选一个）

所以策略针对一个模型，模型评估更高一层，针对的是多个模型

统计学习模型

Chapter 1 线性回归

$$Y = w^T\phi(x) + \epsilon$$
$$P(y|x,\theta) = N(w^T\phi(x), \sigma^2)$$

我们认为 x, y 有存在联合概率分布，但并不关心这个事实，因为我们想要的是观测到一个 x ，能准确预测一个 y ，也就是 y 关于 x 的条件概率，在这里我们认为条件概率满足正态分布，其中 x 控制均值。

X 确定，而 Y 仍是随机变量的这种情况是很普遍的，因为现实生活中，两个随机发生的事情并不是一对一的，而是许多因素互相交织，这些不确定的因素我们并不关心也是无法全面研究的，但是通常来说，他们的总体作用形式是正态分布，因此 Y 的随机性就来自这些我们没有研究的因素。

如何推断其中参数？用极大似然法

常规极大似然法认为我在实验中观测到了 x_i ，那么肯定是因为在分布中它的概率（密度）取值最大，那么我们就应调整参数，使得 x_i 对应的概率（密度）取值最大。

而这里采用的是极大似然法的变种：既然我在观测到 x_i 的情况下，观测到了 y_i ，那么肯定是 $P(y_i|x_i,\theta)$ 是最大的，最终能够推出：

$$l(\theta) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - w^Tx_i)^2 - \frac{N}{2}log(2\pi\sigma^2)$$

这个时候我们并不需要管 σ^2 ，因为线性回归的目的在于，训练模型使得我能在观测到一个 x 时，输出一个 y，而 $y = E(Y) = w^T\phi(x)$ ， 那么着重点为 w ，也就推出 $l(\theta)$ 最大代表 $\sum_{i=1}^{N}(y_i - w^Tx_i)^2$ 最小。因此若为目的导向，我们可以将统计形式化简成统计学习形式

模型为：

$$y = w^T\phi(x)$$

损失函数为：

$$loss(\theta) = \sum_{i=1}^{N}(y_i - w^Tx_i)^2$$

统计形式和统计学习形式本质是一样的，那么原形式（统计学中的似然函数求导）找点，和简化形式（损失函数梯度下降）找点，两者找到的点也就是一样的。

统计偏向理论，统计学习偏向实验，梯度下降虽然本质就是通过导数为零求极值，但是这种形式能够进行编程，找出解析解。

需要说明的是，条件分布中，两个随机变量之间的关系是很少能够直接用式子表示的，也就是 $Y = w^T\phi(x) + \epsilon$ 这种表示形式是很奢侈的，多数时候只能表示成 $P(y|x,\theta) = N(w^T\phi(x), \sigma^2)$ 形式

Chapter 2 逻辑回归

$$p(y|x,\theta) = Ber(sigm(w^Tx))$$

逻辑回归中 x 和 Y 的关系没有 $Y = w^T\phi(x) + \epsilon$ 类似形式，只能写成条件概率

它的损失函数就是随机样本联合分布的极大似然

它的拓展形式是softmax，也就是当前的 y 离散取值不只两个，而是多个，求解思路和上述二分类也是相同的

输入大多数是连续变量，输出为分类变量，在实际使用中（比如信用卡评级），经常先将特征进行分箱（离散化）。从统计角度如何理解？它的损失函数形式是交叉熵，这个形式可以从极大似然法推出，同时也可以通过KL散度推出。

2.1. 损失函数选择

面试常考的一个问题是：逻辑回归为什么使用交叉熵而不是MSE？首先要说明的是mse是万能的loss函数，任何监督学习都是能使用的，而在分类问题里交叉熵优于MSE的答案总共有四点

1. 分类问题为多项分布，回归问题为残差正态分布，两者的最大似然表达式分别对应交叉熵和MSE
2. 拟合目标是概率分布函数，所以用KL散度
3. 交叉熵训练更容易收敛
4. MSE损失函数非凸

第一点涉及到我们在解决一个问题前，对于这个问题的先验知识。在分类问题中，因为所有的点要不是0，要不是1

第二点，我们首先要介绍KL散度

$$D_{KL}(p||q) = \sum_{j=1}^{m} P(Y=y_j) \log \frac{P(Y=y_j)}{Q(Y=y_j)}$$

上述散度衡量的一条样本，这里的下标 j 不是一条样本，而是一条特定样本里的一个状态，那么 P 就是一个样本的的真实分布，而 Q 则是这条样本的预测分布。比如说我们现在研究第10个样本，这条样本的标签是 1 ，那么它的真实分布和预测分布为

$$P(Y=1)=1, \quad P(Y=0)=0$$
$$Q(Y=1)=p, \quad Q(Y=0)=1-p$$

$$Q(Y=1)=Q(Y=1|X=x)=p$$

我们要使得预测分布接近真实分布，也就是在输入 x 的情况下，算出的概率值 p 要尽量接近1，KL散度就会变小，如果两者相等，其值为零。但这个散度其实存在冗余信息

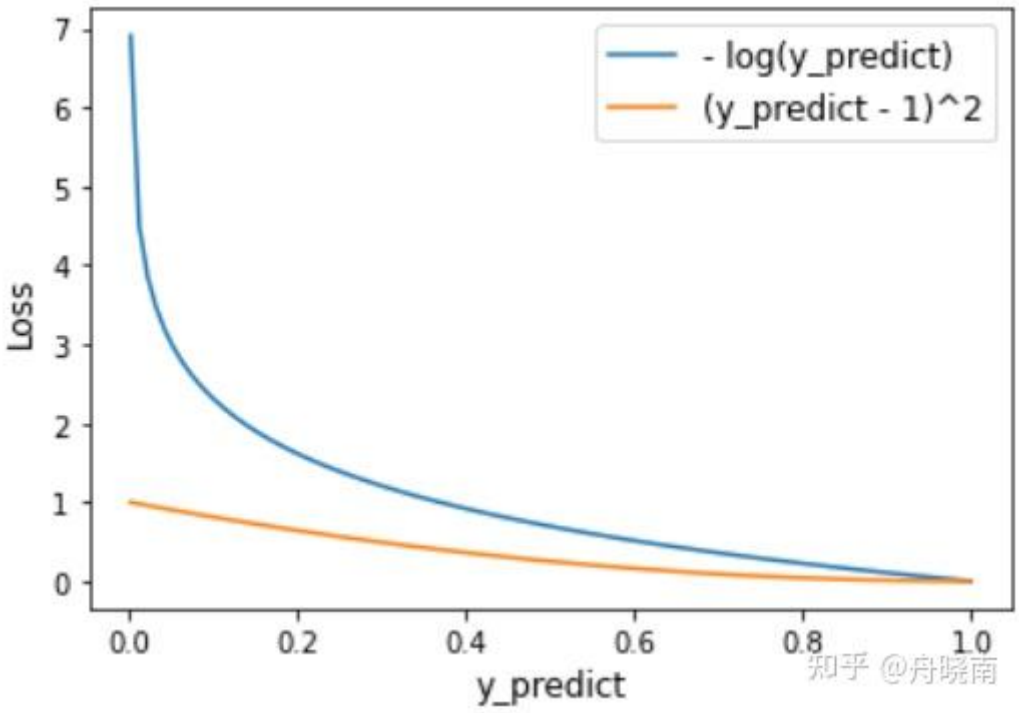
$$D_{KL}(p||q) = \sum_{j=1}^{m} P(Y=y_j)\log P(Y=y_j) - \sum_{j=1}^{m} P(Y=y_j)\log Q(Y=y_j)$$

$$D_{KL}(p||q) - \sum_{j=1}^{m} P(Y=y_j)\log P(Y=y_j) = -\sum_{j=1}^{m} P(Y=y_j)\log Q(Y=y_j)$$

这样调整是因为拆出来的第一项是常数（信息熵），而我们调整 $Q(Y=1|X=x)$ 的函数结构只会变动第二项，所以把它单拎出来，这就是交叉熵。而如果存在多个样本，就要使得平均交叉熵最小，得到

$$loss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m} P(Y=y_j)\log Q(Y=y_j)$$

第三个原因是交叉熵训练更容易收敛，那么就是在sigmod函数的基础上，去比较两种损失函数的梯度下降谁快



假设 label 为 1，预测为 y_predict，那么预测错误的点的 Loss 如上图所示，显然 log-loss 比 MSE 要大，并且对于预测错误的样本，错得越离谱，惩罚越严重。再从公式上看

$$Loss = (\hat{y} - y)^2$$

$$\frac{\partial Loss}{\partial w} = 2(y_i - \hat{y}_i)\hat{y}_i(1-\hat{y}_i)x_i$$

MSE损失函数会导致梯度中包含 $\hat{y}_i(1-\hat{y}_i)$ ，也就是预测值逼近真实值时，梯度接近零，收敛很慢。

第四点是MSE损失函数非凸，凸的定义是二阶导恒大于等于0，那么对上述一阶导接着求导

$$\frac{\partial}{\partial w}(\frac{\partial Loss}{\partial w}) = \hat{y}_i(1-\hat{y}_i)x_i^2(-y_i + 2(1+y_i)\hat{y}_i - 3\hat{y}_i^2)$$

二阶导不恒大于零，所以非凸。

Chapter 3 感知机

- 感知机损失函数设计中不包含正确分类点，这是因为用超平面划分有无数种情况，对于一个点来说，只要超平面把它分正确了，超平面的参数它就不再关心了，所以只有在错误的时候，点才要去调整超平面参数。同时另一个角度来说，如果包含正确分类点，那么损失函数最终收敛的地方不一定是正确分类的地方，而不包含正确分类点，函数收敛、函数值为零、分类全部正确三者是等价的
- 损失函数的图形应该总体是凸函数，但是在最终正确的wb取值附近是一大块平地，因为感知机的参数不唯一，而这些参数对应的损失函数取值都为0
- 从编程角度来看，监督学习只需要两个内容：参数，以及损失函数关于它们的偏导
- 在初始给样本打标签的时候

```
y = np.array([1 if i == 1 else -1 for i in y])
```

  无论哪个标成1或者-1都没有关系，这是因为一个平面有两个方向的法向量，当我把第一类样本标成1得到的法向量其实是把第二类样本标成1得到的法向量转了180，而两者描述的都是同一个平面

Chapter 4 KNN

$$P(y=c|x,D,K)=\frac{1}{K}\sum_{i\in N_k(x,D)}I(y_i=c)$$

- KdTree算法的核心部分是一个中序遍历，对于当前节点，传入：
    - 目标点，以及此时找到的最近距离
    - 当前节点为空，则返回特定结果，如果不为空，执行下面
    - 先将目标点和此时找到的最近距离传入nearer_node子节点，认为它会返回在nearer_node子树中找到的最近点，最近距离，以及访问过的点数
    - 如果子树中返回的最近距离比当前的最近距离小，就更新最近点和最近距离为nearer_node中找到的
    - 判断超球是否被超平面切割，如果不切割直接返回，如果切割执行下面
    - 判断当前节点是否比nearer_node子树中找到的点更接近目标点，更接近就更新最近点和最近距离
    - 去往further_node，将结果进行比较，返回
- KdTree的优势在于将数据点分成二叉树，那么log时间复杂度就能找到最近点
- 算法中演示的都是找最近点，而不是最近k点，要推广到最近k点需要借助长度为k的最大堆（堆顶元素是是整个堆中最大元素），当堆没有满的时候只要有点就进堆（此处中序遍历，左中右都要走），而当堆满之后，需要将堆顶的点的距离和当前点的距离进行比较，后者小的话挤掉堆顶元素进堆（此处中序遍历就只用走最近邻的逻辑）

Chapter 5 朴素贝叶斯法

- 类别：生成模型
- 用法：主要用于多类分类，输入X为离散分布，输出Y是离散分布
- 应用：文档分类（输入X为一篇文章的词向量）

它就是单纯的贝叶斯公式的应用，而没有贝叶斯统计的意思

$$P(Y=y_i|X=x)=\frac{P(X=x|Y=y_i)P(Y=y_i)}{\sum P(X=x|Y=y_j)P(Y=y_j)}$$

数据是由 $P(X,Y)$ 产生，并且就把数据频率视作概率（此处频率当概率就是极大似然估计），那么只要出现数据，联合概率分布就已确定（训练阶段结束）

而预测就输入 x ，算概率，取产生最大概率的 y_i 作为预测值

朴素的原因是认为 X 的各个维度之间独立

- 在用贝叶斯公式转换输入和输出之后，最后一步是在训练集找到相应概率，书中说了两种方法，第一种是极大似然估计，第二种是贝叶斯估计，贝叶斯估计就是在极大似然估计的基础上，为了防止结果概率为零，加了一个常数
- 程序中，只要计算出p(Xi = xi | Y = ck)，就能算出结果，具体操作是做成一个键为ck，值为列表（离散则是各个取值的概率，连续则是函数参数）
- 李航贝叶斯估计法推概率和样本的关系搞不懂
- 在sklearn包中，朴素贝叶斯有多种

```
from sklearn.naive_bayes import BernoulliNB,
MultinomialNB, GaussianNB
# 伯努利模型、多项式模型、高斯模型
```

  他们的区别应该就是在Y=ck固定的时候，每一个X分量的分布是指定的这一种，用样本数据去得到这个分布的参数

Chapter 6 最大熵模型

- 最大熵原理：除去系统的已知信息，我们对于系统未知部分的猜测总是认为这一部分是平均的、等可能的
- 熵最大模型和极大似然法之间的不同之处在于：
  - 极大似然法：拿抛硬币举例，我对系统的已有认知只有这个系统只会产生两种结果，因此概率可以写成p和1-p；而我对系统的未知部分是借助数据去算似然函数，使得这个函数最大确定的，因此数据是用来确定我未知部分的辅助工具

    - 熵最大模型：数据和我对系统的已有认知构成约束条件，而熵最大原理不需要任何的辅助工具就能确定系统的未知部分；从感性的角度来说，这样做是因为熵最大原理本身不需要辅助工具，而确定系统又必须要用数据，所以数据就只能加入已有认知一起构成约束条件
- 具体操作上，因为数据和已有认知毕竟是两个东西，所以在两者结合构成约束条件的时候，数据产生的叫做经验概率（猜测）

Chapter 7 SVM

- 线性可分支持向量机：针对数据可以被超平面正确分类，通过硬间隔最大化实现
- 线性支持向量机：大多数数据可以被超平面正确分类，但是有一些数据越过间隔，甚至是越过超平面，通过软间隔最大化实现
- 非线性支持向量机：将原始数据空间映射成另一个空间，而这个空间中的数据点能够被线性支持向量机分类，通过核技巧与软间隔最大化

那么最终需要优化的目标就是加上软间隔和核函数后的一般形式：

$$\underbrace{min}_{\alpha}\frac{1}{2}\sum_{i=1,j=1}^{m}\alpha_i\alpha_j y_i y_j K(x_i,x_j)-\sum_{i=1}^{m}\alpha_i$$

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

$$0\le\alpha_i\le C$$

即，调整所有的 α 使得在满足条件的情况下，目标函数达到最小。

但问题是 α 的个数是样本的个数，因此数量极其庞大，不能同时调节这么多参数，所以采用启发式算法，将两个 α 视作变量进行调节，而将其他 α 固定，这就是线性支持向量机优化问题的具体实现——SMO算法

7.1. 线性可分支持向量机优化问题推导

首先我们要定义平面（拿二维举例）

$$w=(w_1,w_2)\quad x=(x_1,x_2)$$
$$wx+b=0$$

它的本质是两个向量的内积

$$w(x-A)=0$$
$$A=(-\frac{b}{w_1},0)\quad or\quad (0,-\frac{b}{w_2})$$

也就是我先给定一个向量 A，锚定了一个平面，然后给定一个法向量，并且规定从原点出发的 x 向量和 A 的差向量都必须和给定的 w 向量垂直，从而收集所有符合条件的 x 确定平面。那么如果一个点不在平面上，这个差向量和法向量不垂直，就会产生正负号，我们就用这个正还是负来判断样本点是哪一类。

再规定法向量永远指向正样本，那么就会有任意样本总有如下事实

$$\hat{\gamma}=y_i(wx_i+b)>0$$

符号搞定，接下来看大小，上式不仅和样本有关，还和法向量长度有关，所以归一化法向量，也就是将法向量长度固定为1

$$\gamma_i=y_i(\frac{w}{||w||}x_i+\frac{b}{||w||})$$

就是一个样本点到平面的绝对距离。我们要找的超平面需要满足两个要求

1. 能正确分割样本
2. 离所有样本要足够远

这两个限制条件就得到了如下优化问题，这个问题是具有几何意义的

$$\max_{w,b}\quad \gamma$$
$$s.t.\quad y_i(\frac{w}{||w||}x_i+\frac{b}{||w||})\geq\gamma,\ \ i=1,2,\dots,N$$

操控参数 w, b，也就是操控超平面，使得样本点到超平面距离的下界最大。但是有个问题，上述优化问题存在多解，本质是 (w, b) 和 $(\lambda w, \lambda b)$ 得出的平面是同一个，也就是 (w, b) 和超平面并不是一一对应的，当然这没有错，但平面和参数一一对应更好，所以我们需要缩减 (w, b) 参数空间自由度，使得其中的点与超平面一一对应。所以我们进行恒等变换，并把函数距离置为 $\hat{\gamma} = 1$

$$\max_{w,b} \quad \frac{\hat{\gamma}}{||w||}$$
$$s.t. \quad y_i(wx_i + b) \geq \hat{\gamma}, \ \ i = 1, 2, \dots, N$$

$$\max_{w,b} \quad \frac{1}{||w||}$$
$$s.t. \quad y_i(wx_i + b) \geq 1, \ \ i = 1, 2, \dots, N$$

为了理解上面的做法，我们再举个例子，如下问题求出的最大值是 1/3

$$max \ \frac{y}{x}$$

$$s.t. \ 3x \geq y$$

而下面一个问题求出的也是1/3

$$max \ \frac{1}{x}$$

$$s.t. \ 3x \geq 1$$

我们可以观察到，在第一个优化问题中，如果x和y成（正）比例变化，他们仍然同样满足约束条件，并且得到的最大值和原来的一样，换句话说，对于这个问题求到的最大值而言，有很多对x和y和其对应。而第二个优化问题则是将y固定成1，其实就是在这很多对x和y中，挑选出了一对。所以单从解的数量的角度上来看，两者并不是等价的，但是一般来说，我们求解一个优化问题都是想找到最优的一个解，我们并不在乎这个优化问题是否存在很多解，在这种目的下，上面两个优化问题可以视作等价。用这个思路来看前面，就是目标函数最大值对应一系列等比例的函数距离和 (ω, b)，我这里固定函数距离，那么就是挑选特定的 (ω, b)，这也能够解释为什么原始形式可以有多个解，而改进形式只有一个解。

再进一步优化，我们有

$$\max_{w,b} \quad \frac{1}{2}||w||^2$$

$$s.t. \quad y_i(wx_i+b)-1 \geq 0, \ \ i=1,2,\ldots,N$$

优化问题得到了，现在的任务就是解这个问题。

1. 写出最原始，具有几何意义的最优化问题

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

2. 写出包含函数距离的等价问题

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

3. 写出拉格朗日函数（这里会引入系数 α，并且有 α 大于零的限制条件)

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

4. 跳过直接拉格朗日函数，和求拉格朗日函数的极小极大，直接去求拉格朗日函数的极大极小（对偶问题），得到最终要优化的问题。进行优化，满足条件的 α 。这里要说明什么时候极大极小问题的解就是极小极大问题的解：满足KKT条件，两解相同

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(x_i \cdot x_j)-\sum_{i=1}^{N}\alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{N}\alpha_i y_i=0 \\ & \alpha_i \geqslant 0, \quad i=1,2,\cdots,N \end{aligned}$$

（面试会考）将原始问题转化成对偶问题进行求解的优点在李航书中有提及：

- 对偶问题往往更容易求解
- 自然引入核函数，进而推广到非线性分类问题。这句话在推导中的体现为，在求拉格朗日函数对于 ω 和 b 的极小的过程中，出现了 ω 关于 x_i 的表现形式，带入到拉格朗日函数中得到了 $x_i \cdot x_j$ 的形式，这个形式就是核函数要替换的结构。

5. 而 ω 和 b 可以写成样本点和 α 的函数，得解

所以我们需要做的就是在计算机中算最终的优化问题

7.2. 线性支持向量机优化问题推导

与上面大同小异，因为不再严格限制所有点必须在间隔以外并正确分类，最终得到的优化问题如下

$$\begin{aligned}
\min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{N}\alpha_i \\
\text{s.t.} \quad & \sum_{i=1}^{N}\alpha_i y_i = 0 \\
& 0 \leqslant \alpha_i \leqslant C, \quad i = 1, 2, \cdots, N
\end{aligned}$$

7.3. 非线性支持向量机优化问题推导

其实本质还是线性支持向量机的优化问题，只不过在进入真正的优化问题之前对原数据进行处理，使得它们变得线性可分

$$K(x, z) = \phi(x) \cdot \phi(z)$$

ϕ 的作用就是将输入空间的数据点，映射到特征空间，使得所有的数据点在特征空间中（准）线性可分。但是这个函数还不是核函数，看上面的优化问题，在将点映射完之后，在优化的时候，必须要做的是将特征空间中每两个点搭配进行内积，那为什么不将映射和搭配内积化成一步呢，只要将输入空间的两个数据点传进某个函数，这个函数就能自动将映射到特征空间和内积都做完，所以核函数就是这样定义的。

这样定义第一个好处当然是化简了步骤，第二个好处是我们能够不受映射函数形式的限制。因为通过 ϕ 计算 K 有时是很困难的，而我直接定义 K，之后的做法就是将两个数据点往这个函数里面灌就行。

但这里有个细节需要注意，核函数是要满足初始定义的，不能随便一个函数就能当核函数。在先算 ϕ 再算 K 的时候，函数形式自然而然就能满足初始定义；而要是直接写 K 又不想去看 ϕ，则需要要另外的判别条件，李航书中介绍了。

但用判别条件造核函数还是很烦，实际生产中就是用轮子

7.4. 优化知识点

当我在目标函数和约束条件中，等比例变换目标函数中出现过的元素时，如果目标函数取值不变，而约束条件同样又满足，那么这个优化问题会有很多套解。如果想要减少解的数量，就将目标函数中的某一个元素在目标函数和约束条件中都固定成一个可行解，转换成新的优化问题，这个新的优化问题在解的数量上会少于原先的问题，但是因为新优化问题的解为旧优化问题解的子集，可称两个优化问题等价。

还需说明一点，上述“需要等比例变换才能判断是否有多套解”这个说法不够准确，因为只要各个元素有变换对应关系就可以，比如

$$max\ \frac{y^2}{x}$$

$$s.t.\ 3x \geq y^2$$

x 变成 λx，而 y 变成 $\sqrt{\lambda}y$ 能够使得目标函数值不变，同时约束条件满足

7.5. SMO算法

在具体实现过程中，用的肯定就是线性支持向量机优化问题（最多带上核技巧），不过因为调节的参数 α 和样本数量一致，成千上万，不好调节，所以采用启发式算法，就是每次将其他视作不变，调两个参数，下一次再调两个。这就是SMO算法的核心思想

7.6. 面试问题

- SVM对数据缺失比较敏感，因为它涉及距离的度量（网上写的原因），另一种原因是说因为模型的形成完全靠一小部分点，而不是全部点，所以如果这小部分点中的某些缺失就会导致超平面产生较大变化。
- 基本策略是超平面的几何间隔最大化

- 函数间隔的起点和终点？我的理解是平面法向量和点到平面的向量，这两个向量的起点和终点，那么前一个没有起点和重点，而后一个计算上来说就是平面上一点为起点，数据点为终点
- SVM可以处理数据不平衡吗？我觉得是可以的，因为SVM只考虑边界上的点（不确定，要问别人）

函数定义辨析.md，拉格朗日乘数法.md，SVM.md

Chapter 8 决策树

有ID3，C4.5和CART三种，预防过拟合是在树生成之后（为什么决策树生成像贪心算法？）

- 输入为离散特征，输出离散
- ID3：输入离散，输出离散，选择特征基于信息增益最大
- C4.5：输入离散，输出离散，选择特征基于信息增益比最大
- CART：输入离散，输出离散，二叉树，选择特征基于基尼指数最小
- CART（回归树）：输入离散，输出连续，二叉树，选择特征基于平方差和最小

Chapter 9 集成学习

集成学习不是模型，而是思想，使用多个基本模型组合得到结果

基本的思想为两种：Boosting和Bagging

- Boosting：有多轮训练串行（每一轮都是在固定的样本上），在上一轮训练结束后，下一轮训练根据上一轮进行参数的调节，再训练，最终得到的结果是每一个训练模型的组合（Adaboost是线性组合），它可以做分类也可以做回归，参见分类GBDT
- Bagging：有多轮训练并行（每一轮的样本都是在同一个大样本抽取出来，每一轮的样本可以重复也可以不重复），每一轮独立训练。最后对于

分类问题，由投票表决产生的分类结果；对于回归问题，由k个模型预测结果的均值作为最后预测的结果（所有模型的重要性相同）

从偏差和方差的角度来理解Boosting和Bagging方法的差异。基分类器，有时又被称为弱分类器，因为基分类器的错误率要大于集成分类器。基分类器的错误，是偏差和方差两种错误之和。偏差主要是由于分类器的表达能力有限导致的系统性错误，表现在训练误差不收敛。方差是由于分类器对于样本分布过于敏感，导致在训练样本数较少时，产生过拟合。

Boosting方法是通过逐步聚焦于基分类器分错的样本， 减小集成分类器的偏差。Bagging 方法则是采取分而治之的策略，通过对训练本多次采样，并分别训练出多个不同模型， 然后做综合，来减小集成分类器的方差。假设所有基分类器出错的概率是独立的，在某个测试样本上， 用简单多数投票方法来集成结果，超过半数基分类器出错的概率会随着基分类器的数量增加而下降。

- Boosting减少偏差
- Bagging减少方差

而Boosting根据思路也可以分成两种：

1. AdaBoost：根据当前的loss来改变样本权重，比如这个样本在学习中误差比较大，则获得一个大的权重，反之获得更小的权重，从而控制后续子模型的产生
2. Gradient Boosting：直接修改样本label，新的样本的label将变成原来的label和已知形成的模型预测值之间的残差

选择一种基础模型，再选择一种集成学习思想，就可以构成一种新模型

- Bagging+决策树=随机森林
- Boosting+比大小函数=Adaboost
- Boosting+Cart树+Gradient=GBDT（Gradient Boosting Decision Tree），GBDT是统计学习模型，XGBoost和LightGBM是该模型的工程实现。

Chapter 10 GBDT

全名Gradient Boosting Decision Tree，它的基学习器是Cart树，Cart树有两种，回归和分类，分类的分裂原则是基尼指数最小，回归的分裂原则是平方差和最小。

因为我们知道Boosting的思想是后一个基学习器拟合前面所有的残差，那么不管GBDT整体被用来分类还是回归，基学习器都只能是回归树。在整体是分类时，基学习器的回归拟合的是概率值

10.1. 二分类GBDT

Cart回归树预测的是当前数据为阳性的概率，概率作为连续值就能很好的融合在模型当中

现在基学习器已经定好，那么下一步就是要找每一棵树的拟合目标，而根据回归GBDT的情况我们可以看出，只要找到损失函数的形式，我们求梯度就能得到拟合目标，因为现在拟合的是分类概率，损失函数应该是交叉熵

逻辑回归中，损失函数为

$$L(\theta) = \sum_i -y_i log(\hat{y}_i) - (1-y_i) log(1-\hat{y}_i)$$

模型为

$$\hat{y} = h_\theta(x) = \frac{1}{1+e^{\theta^T x}}$$

令学习器形式为

$$F(x) = \sum_m h_m(x)$$

我的想法是直接将 $F(x)$ 换掉损失函数中的 $\hat{y}$，但是这里是换掉 $\theta^T x$，也就是换掉逻辑回归的部分，得到的形式为

$$L(F(x)) = \sum_i log(1+e^{-F(x_i)}) - (1-y_i)F(x_i)$$

然后对每一个样本的 $F(x_i)$ 求导取负梯度

$$r_{m,i}=-\left[\frac{\partial L}{\partial F(x_i)}\right]_{F(x_i)=F_{m-1}(x_i)}=y_i-\frac{1}{1+e^{-F(x_i)}}$$

我们可以看到当前Cart树的拟合目标是前面所有累加模型算出的概率到真实概率的差距，前面要是用 $F(x)$ 换掉损失函数中的 $\hat{y}$ 得不到这么简单的形式。

这个形式是拟合目标，也就是Cart树在每一步分裂过程中都要使用的 $y_{m,i}$ 值，有了这个值Cart树就能生成，但是最后的预测值是多少呢？也就是在一个样本分到叶子节点后，这个叶子节点预测值是多少？Cart回归树的取值是所有分到这个叶子节点的 $y_{m,i}$ 的平均值，但是这里不是，叶子节点的预测值为

$$c_{m,j}=\underset{c}{arg\,min}\sum_{x_i\in R_{m,j}}L(y_i,F_{m-1}(x_i)+c)$$

这个式子的意思是在拟合目标 $r_{m,i}$ 和分裂原则指导Cart树生成之后，叶子节点的值（也就是之后任何数据点来到这个叶子节点的预测值）是训练集中来到这个叶子节点的数据点都取同一个值，然后把这个值带到损失函数当中，调整这个值，使得损失函数极小。

这也是为什么基学习器结果的累加是直接加

$$F(x)=\sum_m h_m(x)$$

而不需要像回归GBDT一样在基学习器前面加上调整系数的原因

$$F(x)=\sum_m \alpha_m h_m(x)$$

因为梯度下降步伐不合适，需要调整的问题已经在决定叶子节点的预测值时解决

但是为什么这里采用这种方式，而回归GBDT是在整棵树完全长好之后，在损失函数中新引入一个参数进行调整，好像回归GBDT和分类GBDT得到预测值的方式对调直觉上也没问题，这个问题待解决

那么接下来的任务就是求

$$c_{m,j}=\underset{c}{arg\,min}\sum_{x_i\in R_{m,j}}L(y_i,F_{m-1}(x_i)+c)$$

把生成当前Cart树的最后一步完成，但是问题是这个式子没有解析解（也叫closed form solution，闭式解），只能得到近似解

$$c_{m,j} = \frac{\sum\limits_{x_i \in R_{m,j}} r_{m,i}}{\sum\limits_{x_i \in R_{m,j}} (y_i - r_{m,i})(1 - y_i + r_{m,i})}$$

其实我们看这个形式也是对处于当前叶子节点的所有点的值进行的一个函数操作，和取平均类似，但是我们要记住它的来源是损失函数，而取平均不是

最后的概率预测值是

$$P(x) = \frac{1}{1 + e^{-F_m(x)}}$$

至此，单棵树的拟合目标 $r_{m,i}$ ，分裂规则，最终叶子节点取值 $c_{m,j}$ 都已具备，树生成的算法就阐述完成

10.2. 多分类GBDT

正如二分类GBDT来源于逻辑回归，多分类来源于Softmax。它的思路是二分类中所有的拟合目标值和预测值都是一个数，它代表属于当前类别的概率。而多分类的拟合目标向量，比如 $[0, 1, 0]$ ，我们就针对每一类进行二分类GBDT，得到其属于这一类的概率，也就是这里每一步要训练三个基学习器

那么当前样本属于第一类的概率是

$$P_1(x) = \frac{e^{F_1(x)}}{\sum\limits_i e^{F_i(x)}}$$

在当前步，每一个基学习器对于一个样本，要学习的输出是

$$-\frac{\partial loss}{\partial F_i} = y_i - \frac{e^{F_i(x)}}{\sum\limits_j e^{F_j(x)}} = y_i - p(y_i|x)$$

10.3. 总结

为什么GBDT（广义来说，Boosting）做分类问题需要借助形式上类似于回归的逻辑回归（本质是分类）加上Cart回归树来做，是因为Boosting本身的定义就是后一个基学习器去补充前一个基学习器的不足，而集成Cart分类树，“补充”无法定义。

并且我们可以从这里推广，Boosting做分类问题的普遍思路是：任选回归基学习器，套上LR或Softmax

优点：

- 预测阶段的计算速度快，树与树之间可并行化计算
- 在分布稠密的数据集上， 泛化能力和表达能力都很好，这使GBDT在众多Kaggle竞赛中， 经常名列榜首
- 采用决策树作为弱分类器使得GBDT模型具有较好的解释性和鲁棒性， 能够自动发现特征间的高阶关系， 并且也不需要对数据进行特殊的预处理如归一化等

缺点：

- GBDT在高维稀疏的数据集上， 表现不如支持向量机或者神经网络。
- GBDT在处理文本分类特征问题上，相对其他模型的优势不如它在处理数值特征时明显。
- 训练过程需要串行训练， 只能在决策树内部采用一些局部并行的手段提高训练速度。

算法复杂度：

$$n \log n * d * m$$

$n \log n$ 是将一个特征排序的时间（逐点搜索的时间是 n ），d 是特征个数，m 是数的深度

Chapter 11 XGBoost

11.1. 核心设计思想

11.1.1. Boosting方法中损失函数参数代表的意义

损失函数的作用是将模型的输出值与数据的真实值进行对比，将两者的差别（不一定是做差）返回，作为损失函数的值，所以最基础的定义应该是如下

$$\mathcal{F}_\star = \underset{\mathcal{F}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, \mathcal{F}(x_i))$$

（用地形图来理解这个函数）

其中的 F 就代表模型，而当我就写 F，而不把模型具体的参数带进去时（比如线性回归中用 $ax+b$ 替代 F ），这个损失函数的最低点应该是0，因为这个函数在理想情况下，应该是每一个数据点都完美拟合（损失函数不管模型的内部结构，而理论上应该是一种模型能够将数据完美拟合）

而如果我们将用 $ax+b$ 替代 F，用 a 和 b 作为参数，那么模型具体的信息就暴露给了损失函数（也可以说是模型的局限性就暴露给了损失函数），那么在以这两者作为变量的损失函数中，最低点是满足模型局限性的最低点。换句话说，将模型的局限性暴露给损失函数后，无论 a 和 b 怎么变化，F 永远走不到上面说的完美拟合模型上去。

而对于Boosting来说，它的损失函数如下

$$\begin{aligned}\mathcal{F}_\star &= \underset{\mathcal{F}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, \mathcal{F}(x_i)) \\ &= \underset{\alpha_m, h_m}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, \sum_{m=1}^{M} \alpha_m h_m(x_i)), h_m \in \mathcal{H}\end{aligned}$$

它仍没有将模型具体信息暴露给损失函数，而是利用梯度下降的思想，将模型表示成一系列的模型的累加（梯度跨步），那么这个损失函数的最低点仍是0

11.1.2. F函数的变化规则

既然 h_m 就是第 m 次我走的方向，那能不能像常规损失函数一样，求出函数在当前点 F 的梯度，直接往这个方向上走呢？这是不行的，这是因为这里的损失函数中没有包含模型内部结构信息，对它求梯度，每一步永远是往完美拟合模型点上面走，而我的目的就是要找到模型，那么我一定要考虑模型的内部结构。这就会导致我每次走的方向（满足具体模型限制条件），和损失函数建议的方向（梯度下降方向）大概率不是同一个方向

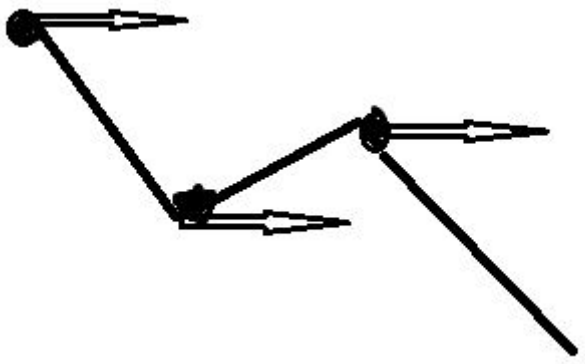
而在之前（比如线性回归模型中），我已经将模型内部结构暴露给了损失函数，那么无论损失函数怎么造，最终得到的结果都会满足线性回归的局限性。所以 a 和 b 梯度（建议方向）和实际行走方向是可以一样的

11.1.3. 一阶泰勒展开处理

虽然我不能往当前点的梯度方向上走，但我就是要走，同时又要考虑具体的模型局限性。

那么做法就是去调整 h_m 的内部结构，尽量让其指向当前 F 的梯度方向。最后得到的结果 h_m 乘上一个常数再添加到 F 上即可（目的是让跨步不走出当前点的线性拟合去，在我找到的文档中，这个常数不是像机器学习模型中是人为指定的，而是要用循环算）。

直观来看（箭头是当前点的梯度方向，实线则是真正走的方向）



从梯度下降角度，那就是往下降最快的方向走，而从泰勒展开的角度，我的损失函数变成如下：（右边第二个式子的第二项前面应该有常数）

$$\sum_{i=1}^{n}\mathcal{L}(y_i,\mathcal{F}_{m-1}(x_i)+h_m(x_i))\approx\sum_{i=1}^{n}\left[\mathcal{L}(y_i,\mathcal{F}_{m-1}(x_i))+\frac{\partial\mathcal{L}(y_i,\mathcal{F}_{m-1}(x_i))}{\partial\mathcal{F}_{m-1}(x_i)}\cdot h_m(x_i)\right]$$
$$\approx\sum_{i=1}^{n}\left[\mathcal{L}(y_i,\mathcal{F}_{m-1}(x_i))-\left[\frac{\partial\mathcal{L}(y_i,\mathcal{F}_{m-1}(x_i))}{\partial\mathcal{F}_{m-1}(x_i)}\right]^2\right]$$

那就是造出一个平方项，使得总的损失函数减小

那么第 m 个模型的任务就是拟合损失函数的负梯度

11.1.4. 二阶泰勒展开处理

常规机器学习算法也就在一阶打住了，那么在Boosting中使用一阶进行处理也是理所当然。一阶泰勒的特点是，展开形式是线性的，换句话说，只有在展开点周围线性拟合范围内，我的泰勒展开才能有效拟合原函数，这个范围当然是非常小的，这也是为什么在上面用 $h_m(x_i)$ 拟合 F' 之后还要乘一个常数（使我的变化量不超过有效的线性拟合范围，但是这个范围我们算不出来，只能用计算机一个个去试）

$$\sum_{i=1}^{n}\mathcal{L}(y_i,\mathcal{F}_{m-1}(x_i)+h_m(x_i))\approx\sum_{i=1}^{n}\left[\mathcal{L}(y_i,\mathcal{F}_{m-1}(x_i))+g_i\cdot h_m(x_i)+\frac{1}{2}\cdot h_i\cdot h_m(x_i)^2\right]$$
$$g_i=\frac{\partial\mathcal{L}(y_i,\mathcal{F}_{m-1}(x_i))}{\partial\mathcal{F}_{m-1}(x_i)},h_i=\frac{\partial\mathcal{L}^2(y_i,\mathcal{F}_{m-1}(x_i))}{\partial\mathcal{F}_{m-1}(x_i)^2}$$

而损失函数二阶展开之后，泰勒展开能够拟合的有效范围必然扩大了，从线性变化范围扩展到二次函数范围，在这个有效范围之内，能够使我损失函数变得最小的 h_m 不再在拟合边界上，而是由二次函数性质可以直接算出来

而二阶泰勒展开的拟合有效范围变大，换句话说，朝向损失函数最小值步子可以迈得更大，是**XGBoost**快于**GBDT**的根本原因

这里也没有常数了，因为二次函数极值点算出来是多少就是多少，不像一阶泰勒时要用常数将步子拉回拟合有效范围

$$\frac{\partial\mathcal{L}(y_i,\mathcal{F}_{m-1}(x_i)+h_m(x_i))}{\partial h_m(x_i)}\approx g_i+h_i\cdot h_m(x_i)=0$$

$$h_m(x_i) = -\frac{g_i}{h_i}$$

那么第 m 个模型（h_m）的任务就是尽量拟合上式。

而这里还有一点，在一阶泰勒展开中，梯度能够告诉我们的只有往哪个方向上走可能会找到最小点，而不能告诉我们附近的最小点到底在哪里，所以我们能够做的就是依据建议的方向（梯度）去走。而二阶泰勒展开中，展开形式本身就包含了附近最小点的信息，换句话说，模型取到这个点时，拟合范围内，损失函数下降的幅度：

$$\sum_{i=1}^{n}\left[g_i \cdot h_m(x_i) + \frac{1}{2} \cdot h_i \cdot h_m(x_i)^2\right]$$

是最大的。那不妨就用这个形式当做第 m 个模型的拟合效果衡量工具，也就是第 m 个模型的损失函数

11.1.5. GBDT和XGBoost跟泰勒展开的联系

直到这里，我们还没有引入任何具体的模型，而只阐述了一阶和二阶的区别，其实任何的基学习器都可以采用这两种思想进行Boosting。
两者的联系与区别在于：

- GBDT：基学习器为CART树，一阶泰勒展开思想，当前树拟合一阶导数
- XGBoost：基学习器为CART树（默认），二阶泰勒展开思想，当前树拟合泰勒展开有效拟合范围内的算出来的极值点

11.1.6. GBDT和XGBoost对于树生成的指导原则

- 对于一个树的构造，必要的是设置损失函数（也就是分裂原则）
- GBDT：只利用了总损失函数的一阶泰勒展开，获得的只有拟合的建议方向
- XGBoost：利用了总损失函数的二阶泰勒展开，获得了具体在当前点的附近哪个点海拔最低。也正因为展开形式中存在对于海拔的衡量，我直接就用这个形式当做当前模型的损失函数。而不用去找另外的损失函数来衡量我对海拔最低点的拟合

11.2. 整体框架

损失函数为

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

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

我们的目标就是最小化损失函数，第一项的 i 指代的是一条样本，而第二项的 k 指代的是一个基学习器，它算的是基学习器 f_k 带来的模型复杂度，也就是我们不仅要减少预测样本带来的损失，还要减少模型复杂度。这种损失函数的形式设置，可以起到防止过拟合的效果。在 t 时刻训练一个新的基学习器，我们有

$$\mathcal{L}^{(t)} = \sum_i l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{k=1}^{t-1} \Omega(f_k) + \Omega(f_t)$$

$$\mathcal{L}^{(t)} = \sum_i l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

按照道理，这时的损失函数应该是上面那种形式，但是在 t 时刻，前面的基学习器情况都固定了，关于它们的正则化项都是常数，所以可以直接去掉。再二阶泰勒展开

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

$$g_i = \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i}|_{\hat{y}_i = \hat{y}_i^{(t-1)}}$$

$$h_i = \frac{\partial l^2(y_i, \hat{y}_i)}{\partial \hat{y}_i^2}|_{\hat{y}_i = \hat{y}_i^{(t-1)}}$$

最前面哪一项又是常数，所以扔掉

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

$$= \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$$

$f_t(x_i)$ 是当前学习器对于一个样本的预测输出值，ω_j 是一个叶子节点的输出值，这两者本质是一个东西，我们按照“一个样本落到一片叶子中，就进到这个叶子这一堆”的原则将上式整理

$$\tilde{\mathcal{L}}^{(t)} = \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)\omega_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)\omega_j^2] + \gamma T$$

到这里，我们干的是将损失函数中关于当前基学习器 $f_t(x)$ 的部分剥离出来，接下来使这个损失函数最小。我们有树结构和叶子输出值两个变量，这里采用的是二元函数 $f(x,y)$ 求极值的思想，先固定 x 求关于 y 的极值，再求关于 x 的极值。

当前的 $f_t(x)$ 固定，也就是树的结构 q 固定，我们调整每片叶子上的输出值，使上面的二次函数为极小值

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

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T$$

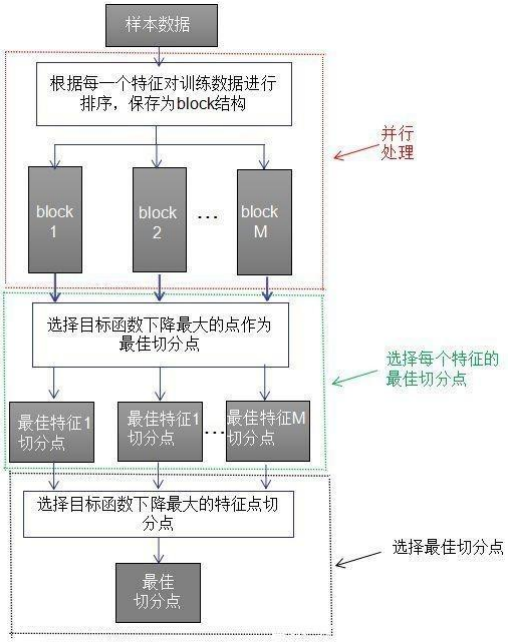
注意，这是在树的结构已经确定的情况下我们得到的最小化损失函数形式，现在要求树的结构，我们采用贪心算法，计算一个叶子节点分裂前损失函数减去分裂后损失函数，得到分裂原则

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda}\right] + \gamma$$

我们可以看到，样本对于训练最重要的东西是损失函数对它的一阶导数 g_i 和二阶导数 h_i

11.3. 特征预排序

在生成一棵Cart树时，分裂每一个节点需要对来到当前节点的样本的各个特征进行排序，再排查选出最优分裂点，如果每分裂一个节点就排序一次肯定是特别浪费时间，那么很自然的一个想法就是在生成当前树之前将数据排好序，更进一步，因为每一棵树所用的数据都是一样的，那么在训练开始之前就把特征排好序岂不是更好，这就叫预排序。



特征预排序就是图中的红框，因为能够并行处理，所以对算法的运行是一个优化

算法原文中写道 “Importantly, the column block structure also supports column subsampling, as it is easy to select a subset of columns in a block.” 也就是在父节点的数据是排序的，我们能够从中拿到子节点的排序数据，而不用重新排。

11.4. 切分点选择

按照常规的逻辑，在面对一个节点中已经排序好的数据，就是扫描切分点找到最优点。但是逐点扫描必然很慢，所以需要优化，下面从简单到复杂，介绍切分点算法

11.4.1. 贪婪算法

根据特征对样本数据进行排序，然后特征从小到大进行切分，比较每次切分后的目标函数大小，选择下降最大的节点作为该特征的最优切分点。最后比较不同block块结构最优切分点的目标函数下降值，选择下降最大的特征作为最优切分点，流程图如下：

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

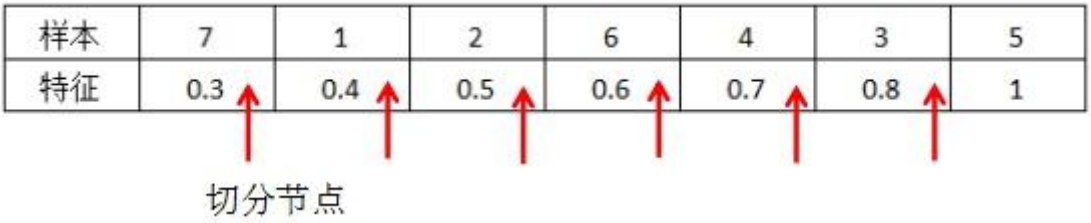
下表表示样本的某一列特征数值：

样本	1	2	3	4	5	6	7
特征	0.4	0.5	0.8	0.7	1	0.6	0.3

根据特征大小对样本重新排序：

样本	7	1	2	6	4	3	5
特征	0.3	0.4	0.5	0.6	0.7	0.8	1

贪婪算法切分节点：



红箭头表示每一次的切分节点，选择目标函数下降最大的点作为切分节点，看到确实是逐点扫描，很浪费时间。

11.4.2. 分位点算法

若特征是连续值，按照上述的贪婪算法，运算量极大 。当样本量足够大的时候，使用特征分位点来切分特征。流程图如下：

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ ***to*** m **do**
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

下表表示样本的某一列特征数值，用三分位作为切分节点

样本	1	2	3	4	5	6	7
特征	0.4	0.5	0.8	0.7	1	0.6	0.3

根据特征大小进行样本排序：

样本	7	1	2	6	4	3	5
特征	0.3	0.4	0.5	0.6	0.7	0.8	1

用特征的三分位点作切分节点：

样本	7	1	2	6	4	3	5
特征	0.3	0.4	0.5	0.6	0.7	0.8	1

特征分位数1/3 作为切分节点　　特征分位数2/3 作为切分节点

红箭头表示每一次的切分节点，选择目标函数下降最大的点作为切分节点。这其实有 LightGBM 的直方图加速的意思了，都是将数据分成一堆，扫描的是堆的分割，而不是点的分割。

11.4.3. 权重分位点算法

这个算法细节在 XGBoost 原文的附录中，应该是其使用的算法

分位点算法假设样本权重相等，也就是每一个样本同样重要，这是不对的，我们构建一棵树的目的是尽量拟合目标，减小损失函数值，那么对于前面树预测误差大的样本应给予更大的关注度，也就是需要给这些样本更大的权重。为了达成这个目标，我们回头再看损失函数

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

因为只有 f_t 相关的是变量，而其他都是常量，所以对其配方

$$\begin{aligned}\mathcal{L}^{(t)} &= \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) + constant \\ &= \sum_{i=1}^{n}\frac{1}{2}h_i[f_t^2(x_i) + 2f_t(x_i)\frac{g_i}{h_i} + \frac{g_i^2}{h_i^2}] + \Omega(f_t) + constant \\ &= \sum_{i=1}^{n}\frac{1}{2}h_i\left[f_t(x_i) + \frac{g_i}{h_i}\right]^2 + \Omega(f_t) + constant\end{aligned}$$

化成这种形式，我们可以认为损失函数是以 $-\frac{g_i}{h_i}$ 作为拟合目标，$\frac{1}{2}h_i$ 作为样本权重的的均方误差函数，所以样本权重就是 h_i 。下面举例如何进行分类点扫描，首先我们有一组数据

特征的值	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
h	0.1	0.1	0.1	0.2	0.3	0.3	0.4	0.5

我们定义一个函数

$$r_k(z) = \frac{\sum_{(x,h)\in\mathcal{D}_k, x<z} h}{\sum_{(x,h)\in\mathcal{D}_k} h}$$

$$r_k(0.3) = \frac{0}{0.1+0.1+0.1+0.2+0.3+0.3+0.4+0.5} = \frac{0}{2} = 0$$

$$r_k(0.4) = \frac{0.1}{2} = 0.05 \qquad r_k(0.5) = \frac{0.1}{2} = 0.1$$

$$r_k(0.6) = \frac{0.1}{2} = 0.15 \qquad r_k(0.7) = \frac{0.1}{2} = 0.25$$

$$r_k(0.8) = \frac{0.1}{2} = 0.4 \qquad r_k(0.9) = \frac{0.1}{2} = 0.55$$

$$r_k(1) = \frac{0.1}{2} = 0.75 \qquad r_k(2) = \frac{0.1}{2} = 1$$

它的含义就是所有特征值小于 z 的样本的权重，占了整体权重的多少，这个数值必然是在0-1之间的

features	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	2
$r_k(z)$	0	0.05	0.1	0.15	0.25	0.4	0.55	0.75	1

我们仍用三分位点来选的话，就是图上的箭头所指的两个分裂点。

11.5. 缺失值的处理

训练阶段

在训练过程中，如果特征 f_0 出现了缺失值，处理步骤如下：

1. 首先对于 f_0 非缺失的数据，计算出 L_{split} 并比较大小，选出最大的 L_{split}，确定其为分裂节点（即选取某个特征的某个阈值）
2. 然后对于 f_0 缺失的数据，将缺失值分别划分到左子树和右子树，分别计算出左子树和右子树的 L_{split}，选出更大的L_{split}，将该方向作为缺失值的分裂方向（记录下来，预测阶段将会使用）

预测阶段

在预测阶段，如果特征 f_0 出现了缺失值，则可以分为以下两种情况：

1、如果训练过程中，f_0 出现过缺失值，则按照训练过程中缺失值划分的方向（left or right），进行划分

2、如果训练过程中，f_0 没有出现过缺失值，将缺失值的划分到默认方向（左子树）

11.6. 面试问题

XGBoost对于缺失值不敏感

Chapter 12 LigtGBM

12.1. 概要

在算法上，其最主要的创新之处在于提出了：

1. 单边梯度采样法（Gradient-based One-Side Sampling, GOSS），样本采样只采样梯度大的样本和少部分小梯度样本。
2. 互斥特征捆绑 （Exclusive Feature Bundling, EFB），特征降维，合并不同时为空的特征，在尽可能小的特征损失下，降低要用的特征数量。
3. 采用了直方图算法（Histogram algorithm）和直方图作差的计算方法，将占内存空间大的连续特征，以分桶的形式，转为离散特征，直方图的叶子节点，有一个可以通过作差的方式得出。

上述三点，尤其是前两点创新 是LightGBM变快、变轻的主要原因。此外，除上述优化外，LightGBM还有其它两个层面的算法创新：

1. 决策树生长策略由Level-wise变为Leaf-wise，减少许多无效的节点切分。
2. 支持直接处理类别，调包侠福音

而在工程上，LightGBM的优化主要由如下几点：

1. 特征并行
2. 数据并行
3. 投票并行

上述工程优化助力的大数据计算，极大的减少了并行计算的通讯成本。

12.2. Histogram

XGBoost使用的Cart树，这种树在每个节点寻找如何进行分裂时，是通过遍历特征以及特征中的取值进行的，比如我有如下特征及取值

$$fearture\ A = [0.1, 0.5, 0.3, 1.2]$$

首先尝试特征 A，那么需要将取值排序 $[0.1, 0.3, 0.5, 1.2]$ ，紧接着扫描切分点取值，有五个可能（也就是0.1左边，0.1和0.3中间...），这里每一个gap只找一个值就够了，因为两个同一个gap里的不同取值得到的分割结果还是一样，我们可以看到这种方式，需要扫描的分割点和样本数成正比，并且还需要预排序，很占计算时间和内存。

而Histogram的意思和信用卡的分箱类似，也就是将上述的样本值离散化，比如 $[[0.1, 0.3], [0.5, 1.2]]$ 成为 $k = 2$ 堆，原有的值不用保存，只用保存特征离散化之后的值，我们只用扫描三个取值就行了，这个堆数是我们人为能够控制的，通过直方图的方式，虽然分割的精度变差了，但是对最后的结果影响不大，一方面能够提升计算效率，另一方面这种较粗的分割点可以起到一种正则化的效果。另一个好处是，分的堆数 k 通常在整个过程中是固定的，细化到一次分裂，一个父节点histogram和两个子节点的histogram堆数一样，那么父节点histogram减去一个子节点histogram就能得到另一个子节点histogram，减少了运算。

分裂点的评判标准还是和XGBoost一样，使用

$$\Delta loss = \frac{(s_L)^2}{n_L} + \frac{(s_R)^2}{n_R} - \frac{(s_P)^2}{n_P}$$

12.3. GOSS

样本的梯度越小，则样本的训练误差越小，表示样本已经训练的很好了。最直接的做法就是丢掉这部分样本，然而直接扔掉会影响数据的分布，因此lightGBM采用了one-side 采样的方式来适配

GOSS首先根据数据的梯度绝对值排序，选取top a个样本，然后在剩余的数据中随机采样b个实例。这样算法就会更关注训练不足的实例，而不会过多改变原数据集的分布。GOSS保留了所有的大梯度样本，对小梯度样本进行随机采样，同时为了保证分布的一致性，在计算信息增益的时候，将采样的小梯度样本的一阶导（梯度）和二阶导乘以一个常量：(1-a)/b，a表示大梯度样本比例值，b表示小梯度样本的采样比值，这样做因为小样本是在 $(1-a)$ 的整体中抽出了 b，这些 b 样本应该用来代表 $(1-a)$ 样本，所以需要将其梯度放大。具体训练步骤为

1. 根据样本点的梯度的绝对值对它们进行降序排序
2. 对排序后的结果选取前 a 的样本生成一个大梯度样本点的子集
3. 对剩下的样本集合 $(1-a)$ 的样本，随机的选取 $b/(1-a)$ 个样本点，生成一个小梯度样本点的集合
4. 将大梯度样本和采样的小梯度样本合并
5. 在计算信息增益的时候，将小梯度样本的一阶导和二阶导乘上一个权重系数 $(1-a)/b$
6. 使用上述的采样的样本，学习一个新的弱学习器
7. 不断地重复 1-6 步骤直到达到规定的迭代次数或者收敛为止。

i	1	2	3	4	5	6	7	8
x_i	0.1	2.1	2.5	3.0	3.0	4.0	4.5	5.0
g_i	0.01	0.03	0.06	0.05	0.04	0.7	0.6	0.07
h_i	0.2	0.04	0.05	0.02	0.08	0.02	0.03	0.03

bini	bin1	bin2	bin3
N_i	3=0+1*3	4=1+1*3	1=1+0*3
G_i	0.09	0.85	0.6
H_i	0.12	0.08	0.03

梯度绝对值的阈值是0.1，大于0.1的数据有2，小于0.1的有6个。大梯度的数据全部保留，大梯度的采样率a=2/8。小梯度的采样个数为2个，采样率为b=2/8。则小梯度的数据都乘 (1-a)/b=(1-2/8)/(2/8)=3。

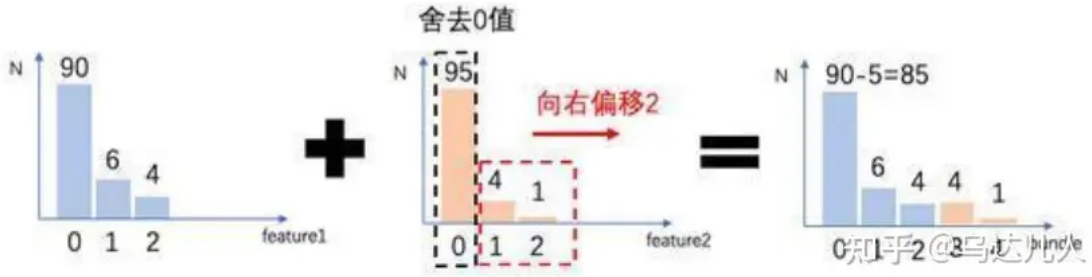
通过上面的算法可以在不改变数据分布的前提下不损失学习器精度的同时大大的减少模型学习的速率。 从上面的描述可知，当a=0时，GOSS算法退化为随机采样算法；当a=1时，GOSS算法变为采取整个样本的算法。在许多情况下，GOSS算法训练出的模型精确度要高于随机采样算法。另一方面，采样也将会增加若学习器的多样性，从而潜在的提升了训练出的模型泛化能力。

12.4. EFB互斥特征捆绑算法

EFB算法全称是Exclusive Feature Bundling，即互斥特征绑定算法。（将多个特征捆绑到一起）

EFB是通过特征捆绑的方式减少特征维度（其实是降维技术）的方式，来提升计算效率。通常被捆绑的特征都是互斥的（一个特征值为零,一个特征值不为零），这样两个特征捆绑起来才不会丢失信息。如果两个特征并不是完全互斥（部分情况下两个特征都是非零值），可以用一个指标对特征不互斥程度进行衡量，称之为冲突比率，当这个值较小时，我们可以选择把不完全互斥的两个特征捆绑，而不影响最后的精度。EFB的算法步骤如下：

- 将特征按照非零值的个数进行排序
- 计算不同特征之间的冲突比率
- 遍历每个特征并尝试合并特征，使冲突比率最小化



将特征1和特征2进行捆绑，形成新的特征，只把值都为0的特征合并

12.5. 代码说明（这一节还需要细细研究）

官方提供的LightGBM包是这样使用的

```
params = {
    'learning_rate': 0.018,
    'max_depth': 9,
    'n_estimators': 600,
    'num_leaves': 440,
    'objective': 'mae',
    'random_state': 42,
    'reg_alpha': 0.01,
    'reg_lambda': 0.01
    # 'device': 'gpu'
}
m = lgb.train(params, train_data, valid_sets=[train_data,
valid_data])
```

相关的公式如下

$$L = \sum_{i=1}^{n} loss(y_{res}, h(x)) + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + \alpha \sum_{j=1}^{T} |w_j|$$

$$w_j = learning\_rate \times \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

1. learning_rate：在XGBoost中我们分析过，一棵树的拟合点是二次函数的最低点，并不需要像梯度下降一样乘一个系数，但是这里还是乘了，learning_rate取值范围一般是0.001到1之间，但也取决于数据集的大小和质量。可以先试用较大的学习率调整其他参数，最后再用较小的学习率获得更优的性能。
2. max_depth：树的最大深度，防止过拟合
3. n_estimators：基学习器的数量
4. num_leaves：叶节点的数量，因为 LightGBM 使用的是 leaf-wise 的算法，因此在调节树的复杂程度时，使用的是 num_leaves 而不是 max_depth。大致换算关系：num_leaves = 2^(max_depth)，但是它的值的设置应该小于 2^(max_depth)，否则可能会导致过拟合。我们也可以同时调节这两个参数，对于这两个参数调优，我们先粗调，再细调。
5. objective：整体损失函数的选择
6. random_state：
7. reg_alpha：正则化参数，该参数用于控制模型复杂度，防止过拟合。它在目标函数中加入L2正则化惩罚项，使得大的权重被缩小，从而控制模型的

复杂度。默认值为0

8. reg_lambda：正则化参数，该参数用于控制模型复杂度，防止过拟合。它在目标函数中加入L2正则化惩罚项，使得大的权重被缩小，从而控制模型的复杂度。一般情况下，reg_lambda的取值范围是[0,∞)，默认值为0，表示正则化强度的程度。如果该参数为0，则不会有任何正则化影响。如果该参数设置得太大，则可能会导致在训练集上出现欠拟合。

12.6. XGBoost缺点

在论述LightGBM之前，我们必须要论述为什么需要LightGBM，也就是XGBoost的缺点有哪些

XGBoost核心是每一个基学习器拟合目标（创新点）是通过整体损失函数的二阶求导得到的，而在基学习器上，并没有进行过多的优化，LightGBM则是基于这一点，对基学习器进行优化

12.7. 缺失值处理

Chapter 13 CatBoost

数据处理

Chapter 1 样本不平衡解决办法

1.1. 过采样和欠采样

比如A类少，B类多

- 过采样：就是将A中的样本数量扩增，使其和B类中的样本数量达到一致。最简单的方法就是将A中样本复制粘贴，但是这样做是有缺点的，如果本身的A类样本中有标记错误，或者是噪声，那么它们就会被成倍放大，造成对A类数据的过拟合。

  而SMOTE方法可以解决这个问题，它不是单纯重复正例，而是采用KNN生成新的A类数据，可以理解成直接复制粘贴的Soft Version。缺点为增加运算开销，同时可能会生成一些可疑的点

- 欠采样：将B类数据丢掉一部分，是的其数据量和A类数据量平衡。如果数据量极其庞大，比如A类几千万条，这样做是没有问题的，因为剩余的数据还是能准确地反映系统总体的信息。但如果数据量比较小，这样做就会使得模型产生较大的偏差。所以直接丢B类数据是不行的。

  改进做法则是将B类数据分成多份，每一份都和A类数据进行搭配训练出一个模型，然后集成这些模型的结果。缺点为：训练多个模型造成过大的开销；A类数据被反复利用，和过采样一样，很容易造成模型的过拟合（这其实就是后面的集成方法）

对于采样方法有如下几点总结：

1. 采样方法一般比直接调整阈值的效果要好
2. 采样方法一般可以提升模型的泛化能力，但是两种采样方法都会有过拟合风险，应该搭配正则化使用
3. 过采样大部分时候的效果比欠采样要好
4. 经验做法：使用过采样（SMOTE）加上强正则模型（如XGBoost）可能比较适合不平衡 的数据。拿到新数据可以用这个做法的结果作为Baseline

1.2. 正负样本设置不同惩罚权重

核心思想就是（比如A类少，B类多，A和B类的样本数量比为1：9）将A类对应的损失函数项放大，导致在对损失函数进行优化的时候，使得模型能够更关注A类样本中所含的信息。

1.2.1. 方法一：直接乘权重

用交叉熵举例，原始交叉熵公式如下：

$$L = -ylog(p) - (1-y)log(1-p)$$

A为阳性，则损失函数为第一项，B为阴性，则损失函数为第二项。更改权重之后的公式如下

$$L = -\alpha\, y\, log(p) - (1-\alpha)(1-y)log(1-p)$$

其中每个类别的 α 都不相同，与这个类别的样本数量成反比

如果样本为A类，那么我对这个样本的预测值在损失函数中的贡献就被放大了九倍，所以在求损失函数最小值的时候，会较优先地考虑预测值逼近这个样本的真实值。

下面论述这种思想在SVM中的公式推导变化和具体实现：

```
X, Y = datasets.make_classification(n_samples=1000,
                                    n_features=4,
                                    n_classes=2,
                                    weights=[0.9, 0.1])
train_x, test_x, train_y, test_y = train_test_split(X, Y,
test_size=0.1, random_state=0)

#SVM_model = SVC(random_state=66)
SVM_model = SVC(random_state=66, class_weight="balanced")
SVM_model.fit(train_x, train_y)

pred1 = SVM_model.predict(train_x)
accuracy1 = recall_score(train_y, pred1)
print(accuracy1)

pred2 = SVM_model.predict(test_x)
accuracy2 = recall_score(test_y, pred2)
print(accuracy2)
```

在拟合模型的时候，属性中加上了class_weight="balanced"，它的意思就是按照数据量的反比给他们加上权重，比如说这里，类别为0的样本数量占90%，类别为1的样本数量占10%，那么就会在优化目标函数中给两者分配比例为1：9的权重。在函数中的具体体现如下

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i$$

原先只要对于每个样本，惩罚参数 C 都相同，而现在每个类别分别设置 C，其具体值如下

$$C_j = C * w_j$$
$$w_j = \frac{n}{kn_j}$$

其中 C 是原始的惩罚参数，n 是样本总数，k 是总的类别数，n_j 是第 j 类的样本数量，比如说这里样本总数为1000，总类别数为2，类别为1的样本数为100，那么 $w_1 = 5$，而 $w_0 = \frac{5}{9}$ ，两者的权重和样本数量确实成反比。

这里是在原始的优化问题上做形式改变，最后的优化问题的形式上的变化则是

$$\begin{aligned}
\min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{N}\alpha_i \\
\text{s.t.} \quad & \sum_{i=1}^{N}\alpha_i y_i = 0 \\
& 0 \leqslant \alpha_i \leqslant C, \quad i = 1, 2, \cdots, N
\end{aligned}$$

限制条件的第二项，每一个样本的边界不再全部一样，而是根据类别有不同，这也会影响到SMO算法对于 α 的裁剪

1.2.2. 方法二：**Focal Loss**

方法一的逻辑是在损失函数中通过提高A类的权重，从而给予它更高的关注度，这样做是很有效的，但是比较粗糙。A类中也有容易分类的数据点和难分类的数据点，我们其实不需要关注容易分类的点，而需要着重关注那些难分类的数据点，这就是Focal Loss的基本思想

Focal loss提出的原始想法是用来解决图像检测中类别不均衡的问题，这里多说一句，图像检测领域是一个典型的正负样本不均衡的领域，因为在一张图像中，大多数情况下，目标总是占据图像的一小部分，占据图像大部分的是背景。

我们依旧以为二分类交叉熵损失函数为例。Focal losss损失函数在的基础上从

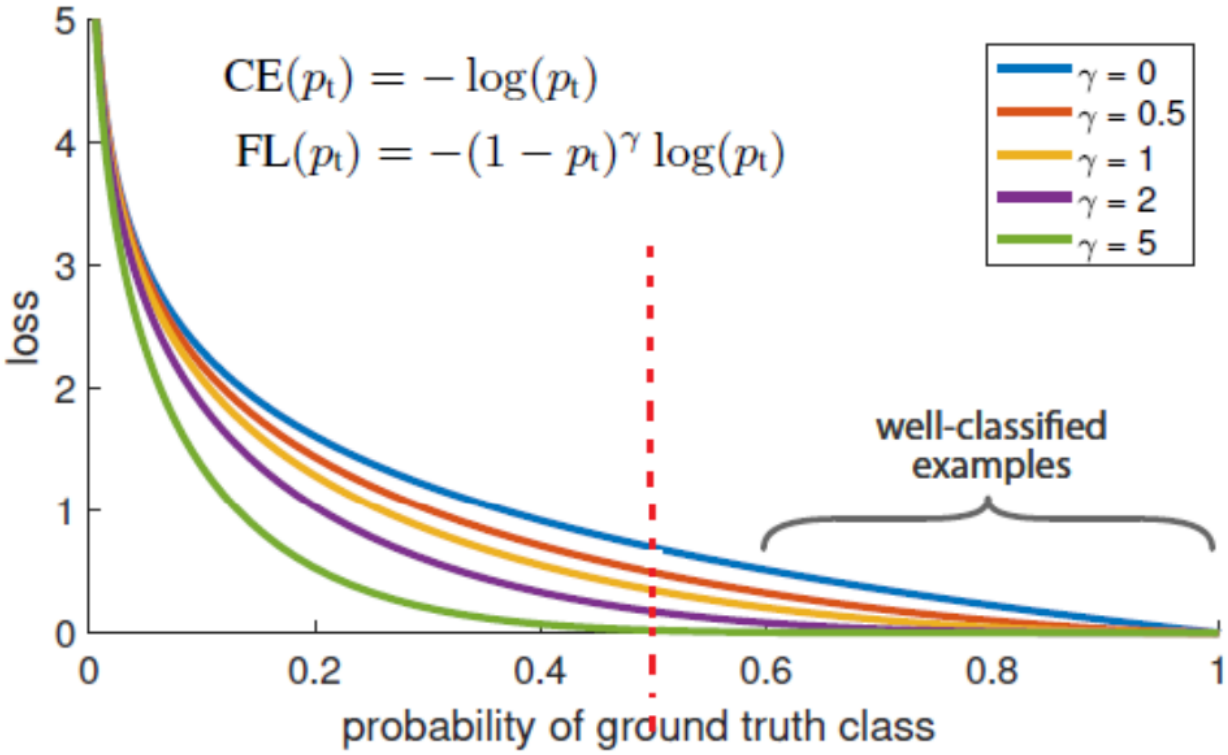
$$L = -ylog(p) - (1-y)log(1-p)$$

变成了

$$FL = -\alpha(1-p)^{\gamma}ylog(p) - （1-\alpha）p^{\gamma}(1-y)log(1-p)$$

变化为两个部分，第一部分就是方法一直接乘权重，而第二项则是对每个类别中比较难分类的那些数据点加上更大的关注，比如某个样本为阳性，则为第一项，如果我的预测值 p 很接近1，就意味着我对这个数据点的预测很有把握，那么前面的 $(1-p)^{\gamma}$ 就会很小，从而降低这个样本在损失函数中的贡献，而如果预测值不是很接近1，就意味着我对预测没有很大的把握，那么 $(1-p)^{\gamma}$ 就会将这个数据点在损失函数中的贡献增大，让我在优化的时候能更加关注到这个点。

从函数图像中体会，则是



将原始的对数函数进行改进，把它往原点压

这种思想在Python中还没有实现包，需要自己实现

1.3. 组合、集成方法

从样本量比较多的类别中随机抽取一定数量的样本，与样本量比较小的类别组合在一块儿训练模型。这样会训练出好几个模型，最后在应用时，使用组合的方法（例如投票、加权投票等）产生分类预测结果。例如，若数据集中的正、负样本的比例为1:9。此时可以将负样本随机分为9份（或者每次随机抽取和正样本相同数量的负样本），然后和所有的正样本组成训练集。这样可以得到9个训练集，分别输入模型之后就可以得到9个训练模型。

这种方式其实借鉴了集成学习中的bagging思想，但是因为这种方法要分别训练多个模型，所以耗时比较长。如果等得起，相对于单一模型绝对可以得到一个更好的提升。

Chapter 2 数据缺失

数值变量缺失值用同类别的平均值，中位数，众数填充；类别变量用同类别的众数填充（这种填充思路还可以先判断两个数据的相似度，加上权重）

另外一种思路是如果缺失值只出现在一栏之中，我们可以将训练集中完整数据分成子训练集，残缺数据分成子测试集，缺失值一栏作为输出。用这样的数据训练一个模型并预测子测试集中的输出，并用此填充缺失值。信用卡评级模型中填充缺失值使用的模型是随机森林

主流的机器学习模型千千万，很难一概而论。但有一些经验法则(rule of thumb)供参考：

树模型对于缺失值的敏感度较低，大部分时候可以在数据有缺失时使用。涉及到距离度量(distance measurement)时，如计算两个点之间的距离，缺失数据就变得比较重要。因为涉及到“距离”这个概念，那么缺失值处理不当就会导致效果很差，如K近邻算法(KNN)和支持向量机(SVM)。线性模型的代价函数(loss function)往往涉及到距离(distance)的计算，计算预测值和真实值之间的差别，这容易导致对缺失值敏感。神经网络的鲁棒性强，对于缺失数据不是非常敏感，但一般没有那么多数据可供使用。贝叶斯模型对于缺失数据也比较稳定，数据量很小的时候首推贝叶斯模型。

总结来看，对于有缺失值的数据在经过缺失值处理后：

- 数据量很小，用朴素贝叶斯
- 数据量适中或者较大，用树模型，优先 xgboost
- 数据量较大，也可以用神经网络
- 避免使用距离度量相关的模型，如KNN和SVM

Chapter 3 特征工程

3.1. 特征归一化

其类别有：线性函数归一化，零均值归一化

3.2. 类别型特征

- 序号编码：类别本身存在大小关系，比如大中小
- 独热编码：类别间不具备大小关系，存在由向量维度过大带来的缺点
- 二进制编码：介于序号和独热，将序号转化成二进制数，数字也是0，1，并且长度肯定小于独热编码

3.3. 数据分箱

数据分箱（也称为离散分箱或分段）是一种数据预处理技术，用于减少次要观察误差的影响，是一种将多个连续值分组为较少数量的“分箱”的方法。

分箱的数据不一定必须是数字，它们可以是任何类型的值，如“狗”，“猫”，“仓鼠”等。 分箱也用于图像处理，通过将相邻像素组合成单个像素，它可用于减少数据量。

在实际操作上，分箱是将一个连续变量x转化成一个向量[x1, x2, x3]，如果x取10，那么0<10<20，因此对应的向量为[1, 0, 0]

分箱的重要性及其优势：

一般在建立分类模型时，需要对连续变量离散化，特征离散化后，模型会更稳定，降低了模型过拟合的风险。

比如在建立申请评分卡模型时用logsitic作为基模型就需要对连续变量进行离散化，离散化通常采用分箱法。分箱的有以下重要性及其优势：

- 离散特征的增加和减少都很容易，易于模型的快速迭代，比如说我们当前训练集的年龄是20～30岁，假设在20～30之间分了5个箱子离散化然后onehot展开，下一次训练的时候如果出现超过30岁的特征，那么直接增加一列0-1特征用来表示用户是否超过30岁即可；
- 稀疏向量内积乘法运算速度快，计算结果方便存储，容易扩展；
- 离散化后的特征对异常数据有很强的鲁棒性：比如一个特征是年龄>30是1，否则0。如果特征没有离散化，一个异常数据“年龄300岁”会给模型造成很大的干扰；
- 逻辑回归属于广义线性模型，表达能力受限；单变量离散化为N个后，每个变量有单独的权重，相当于为模型引入了非线性，能够提升模型表达能力，加大拟合；
- 特征离散化后，模型会更稳定，比如如果对用户年龄离散化，20-30作为一个区间，不会因为一个用户年龄长了一岁就变成一个完全不同的人。当然处于区间相邻处的样本会刚好相反，所以怎么划分区间是门学问；

3.4. 特征组合

这里论述两个问题：

- 如何挑选特征进行组合：决策树
- 在组合之后参数过多怎么办：将参数用低维向量表示

问题一，二看百面机器学习，下面论述的是问题二

为了提高复杂关系的拟合能力，在特征工程中经常会把一阶离散特征两两组合，构成高阶组合特征，更确切的说，是将两个特征中每一对离散值组合，视为一个新特征。比如A特征离散取值为中文，英文，B特征离散取值为电影，电视剧

	中文	英文
电影	中文电影	英文电影
电视剧	中文电视剧	英文电视剧

那么新特征就是组合出来的四个取值

表 1.2　语言和类型对点击的影响

是否点击	语言	类型
0	中文	电影
1	英文	电影
1	中文	电视剧
0	英文	电视剧

表 1.3　语言和类型的组合特征对点击的影响

是否点击	语言＝中文 类型＝电影	语言＝英文 类型＝电影	语言＝中文 类型＝电视剧	语言＝英文 类型＝电视剧
0	1	0	0	0
1	0	1	0	0
1	0	0	1	0
0	0	0	0	1

那么如果采用比如逻辑回归，原先只要训练两个权重，现在就要训练四个，也就是上述表格的每一个交叉项都要分配一个权重。

问题是如果原始一个特征中的离散值数量小还好，如果离散特征大（比如用户账号，这就上千万计了），那么需要训练的参数就会极其庞大。解决办法是将权重矩阵分解，比如中文电影的权重是中文向量和电影向量内积得到，这就变成了推荐算法中的隐语义模型。

3.5. 面试问题

- 为什么要归一化？

  在梯度下降算法时，算法能收敛得更快

- 什么时候归一化不适用？

  梯度下降需要归一化，而不用梯度下降的算法就不需要归一化，比如说决策树，以 C4.5 为例，决策树在进行节点分裂时主要依据数据集关于特征的信息增益比，而信息增益比跟特征是否经过归一化是无关的，因为归一化并不会改变样本在特征 X 上的信息增益。

优化算法

Chapter 1 梯度下降法

核心思想就是找到函数下降最快的方向，然后往那个方向走

Chapter 2 牛顿法

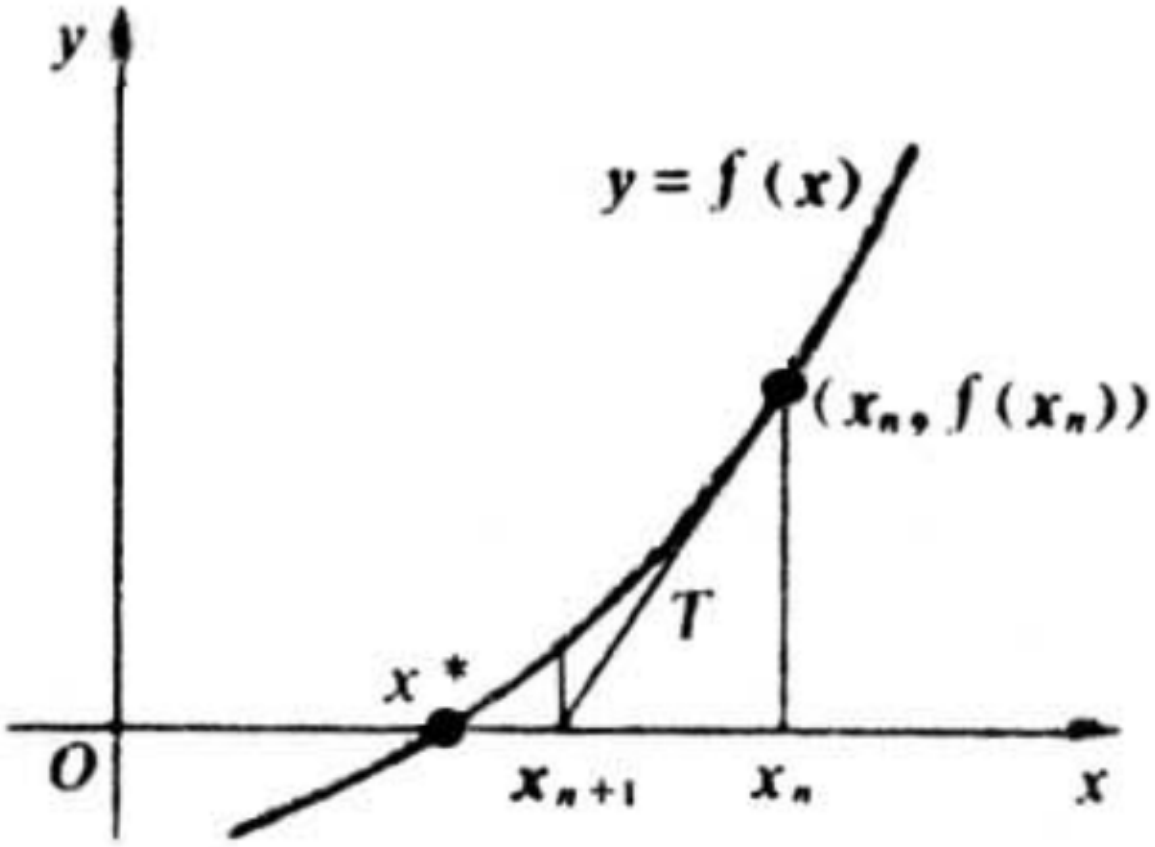
牛顿法一般有两个用法，第一个是求方程的根，第二个是最优化（求函数的极值点），而第二种用法是基于第一种用法的，因此我们先来介绍第一种用法

2.1. 求方程的根

将函数进行一阶泰勒展开，并求解这个一阶形式等于0的点

$$f(x) \approx f(x_0) + f^{'}(x_0)(x - x_0) = 0$$

从图上看可以直观地看到，我们求的并不是原函数的零点，而是在当前点的切线的零点



牛顿法求实根图示

虽然求的并不是原函数的零点，但是我们可以看到切线的零点相较于原来的位置更加靠近原函数的零点，所以我们可以基于这个新点，再去做原函数泰勒展开，求切线零点，再得到一个更加靠近原函数零点的点，直到最后一定会收敛到原函数零点。这就是牛顿法的基本原理

2.2. 最优化（求函数极值）

函数极值的必要条件是一阶导数为零，如果想要求最大值或最小值，只要将所有极值点求出，一个个验证即可。最优化的核心任务就是求一阶导数为零的点，就是求方程的根

这里的符号需要说明一下，因为在最优化的时候，x 一般不会是单个值，而是一个向量，所以会涉及向量的求导。$g(x)$ 是一阶导数，它是一个向量，$H(x)$ 是二阶导数，因为每一个分量能够搭配，它是一个矩阵，名字为黑塞矩阵。我们要求的为

$$g(x) \approx g(x^{(k)}) + H(x^{(k)})(x - x^{(k)}) = 0$$

其中的 $x^{(k)}$ 是第 k 次迭代所在的点，解得

$$x = x^{(k+1)} = x^{(k)} - H^{-1}(x^{(k)})g(x^{(k)})$$

如果将 $-H^{-1}(x^{(k)})g(x^{(k)}) = p_k$，则有

$$x^{(k+1)} = x^{(k)} + p_k$$

用这个关系迭代，最终就能找到一阶导数为0的点。

牛顿法和梯度下降法的本质区别在于：如果将函数想象成山坡，梯度下降法考虑的是在我站的这个点，去看往哪个方向上走能够最快下降，这里并没有涉及到最低点，只不过累积走，最后的结果是走到最低点。而牛顿法求极值是在当前我就尽量去看到函数的最低点（一阶导数的一阶导数为0，虽然不是真实的最低点，我把它当作最低点），然后往那个方向上跨步。

牛顿法比梯度下降法要快的原因：

- 基于我现在的位置，直接尝试去看到我认为的最低点，而不是只看周围的梯度
- 直接跨到到我认为的最低点，而不是根据梯度慢步走

从迭代形式上可以分析出另外一种说法：

$$x^{(k+1)} = x^{(k)} - \lambda g(x^{(k)})$$
$$x^{(k+1)} = x^{(k)} - H^{-1}(x^{(k)})g(x^{(k)})$$

引入二阶导的信息能够使我看到我视野范围内的最低点，直接影响到我走的方向以及走的步子大小（少了 $lambda$，多了 $H^{-1}(x^{(k)})$ ）

2.3. 牛顿法求极值的缺点

需要算二阶导数，因为函数自变量是向量，二阶导数还是个矩阵

2.4. 牛顿法和XGBoost的关系

一句话，两者的下一个目标点都是 $x = -\frac{g}{H}$，只不过牛顿法的变量就是纯粹的变量，我能够直接令 x 等于这个目标点，而XGBoost的变量是一棵树，只能尽可能地让这棵树的值靠近目标点。

函数形式上来看，令一阶导函数的一阶泰勒展开为零，等价于找二阶泰勒展开的极值点。这也算是解答了在XGBoost中我的一个疑惑点：为什么能够直接去拟合二阶泰勒展开的极值点，而跳过“讨论这个极值点在不在有效的拟合范围之内”的步骤。用这里的逻辑来解释就是：用一阶导函数的一阶泰勒展开为零求解，有时确实会超出有效拟合范围，但不过我找到的这个点确确实实会比之前那个点要更靠近真实的最低点，并且要比梯度下降快得多，这就足够了。

Chapter 3 拟牛顿法

牛顿法的优点也是它的缺点：需要算二阶导数——黑塞矩阵，并且还要求它的逆矩阵，拟牛顿法就是尝试找到一个能够完成目的的替代矩阵，省去计算逆黑塞矩阵的繁琐步骤，并且每一步的替代矩阵都可以由上一步的替代矩阵算出来

模型评估

Chapter 1 概述

	PREDICT POSITIVE	PREDICT NEGTIVE
Real Positive	True Postive(TP)	False Negtive(FN)
Real Negtive	False Positve(FP)	True Negtive(TN)

- precision：精准率，又叫查准率，TP/(TP+FP)
- recall：召回率, 又叫查全率，TP/(TP+FN)
- sensitivity(TPR)：灵敏度 = TP/(TP + FN) 所有正例中, 被识别出来的比例，和recall是一回事
- specifcity(TNR) ：特异性= TN/(TN + FP) 所有反例中, 被识别出来的比例
- PPV：阳性预测值=TP/(TP + FP) 所有识别为正例的样本中, 预测正确的比例，和precision是一回事
- NPV： 阴性预测值=TN/(TN + FN) 所有识别为反例的样本中, 预测正确的比例
- Acc：准确率= (TP + TN)/(TP + FN + FP + TN) #所有预测正确的样本占总样本的比例。注：在正负样本不平衡的情况下，准确率这个评价指标有很大的缺陷。比如在互联网广告里面，点击的数量是很少的，一般只有千分之

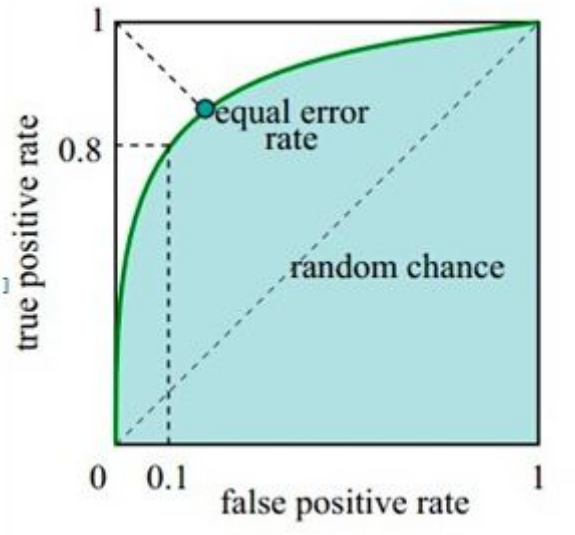
几，如果用acc，即使全部预测成负类（不点击）acc 也有 99% 以上，没有意义。

Chapter 2 ROC曲线

	PREDICT POSITIVE	PREDICT NEGTIVE
Real Positive	True Postive(TP)	False Negtive(FN)
Real Negtive	False Positve(FP)	True Negtive(TN)

它是评判二分类模型的预测，这种评判方法能够消除样本不平衡的影响，它只关注每一种类别里面被预测为阳性的比例，而与具体的样本数量没有关系

TPR = TP/(TP+FN)为纵轴、FPR = FP/(FP+TN)为横轴，图形如下



阈值越大，那么预测为阳性的样本数量就会减少，那么TPR和FPR都会在曲线的左下方，当阈值为1，样本全部被预测成阴性，坐标为 (0,0)。当我逐渐减小阈值，TPR和FPR都会增大，点往右上方移动，当阈值为0，样本全部被预测成阳性，坐标为 (1,1)。如果曲线为 $y = x$ ，代表在真实阳性样本和真实阴性样本中，我预测为阳性的比例都相同，意思就是我的预测和样本的阴阳性根本没有关系，所以这条曲线代表我建立的模型完全没有效果。

而预测有效在这两个指标中的体现应该是，TPR尽可能大，FPR尽可能小，所以曲线会往上凸。用曲线和x轴围成的面积来衡量的话，就是面积越接近于1预测效果越好，这就是AUC（Area Under the Curve）

Chapter 3 AUC

AUC是ROC的一个子话题，因AUC的物理意义需要详细阐述，所以另起一段

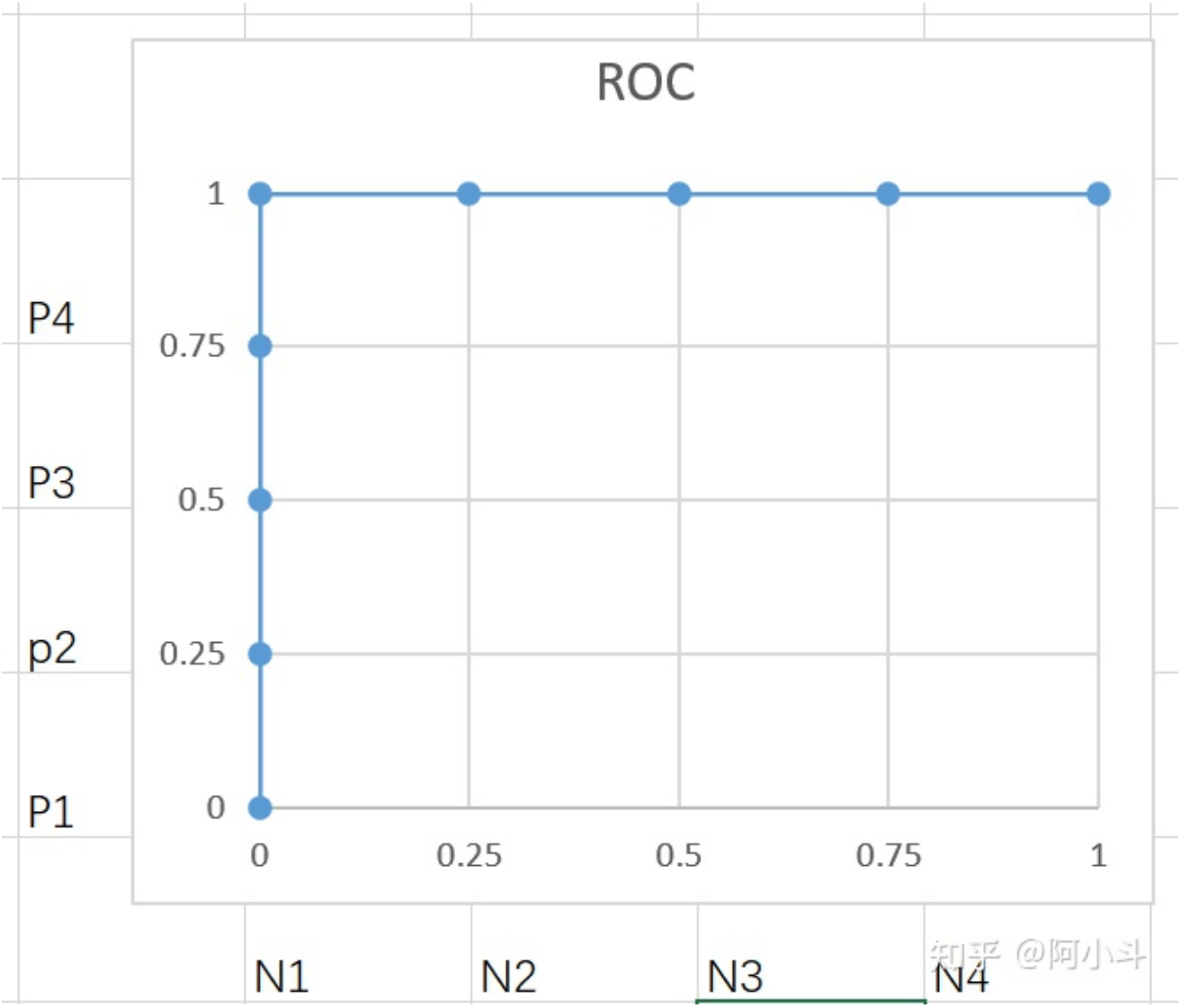
假如我们有一组预测值和真实值，概率大小从大到小已经排列好

P1	P2	P3	P4	N1	N2	N3	N4
0.9	0.8	0.7	0.6	0.4	0.3	0.2	0.1
1	1	1	1	0	0	0	0

总共8个样本，真实值是四个证例标签为1，四个反例标签为0，如果我们用逻辑回归0.5为界限的话，可以看出我们的预测是100%正确的。那么我们开始绘制ROC曲线

一般绘制ROC曲线习惯上用阈值，但是其实这个阈值就是每个样本的预测概率，本质就是我从P1开始逐个取考虑样本，所以你也可以不说使用阈值而是说逐个考虑样本，这是一样的。

- 从P1开始，用阈值0.9（或者不说阈值，就说只考虑P1），只有P1是预测为1，其余都预测为0，那么TPR =1/4=0.25，FPR=0，ROC点（0， 0.25）
- 再考虑P2，用阈值0.8，那么P1,P2预测为1，那么TPR =2/4=0.5，FPR=0. ROC点（0， 0.5）
- 考虑P3, 同理得到ROC点（0， 0.75）
- 考虑P4，ROC点（0， 1）
- 考虑N1，这个时候ROC点是（0.25， 1）开始右移动了！
- N2,N3 N4 点分别为（0.5, 1）(0.75,1) (1,1)



我们可以看到，随着阈值逐渐调小，曲线慢慢往右上角走，最终生成ROC曲线，并且横轴纵轴分别四个点加上XY轴可以将曲线和X轴包围的区域分割成16块，所以我们可以认为，每一小块对应着一对正负样本的搭配

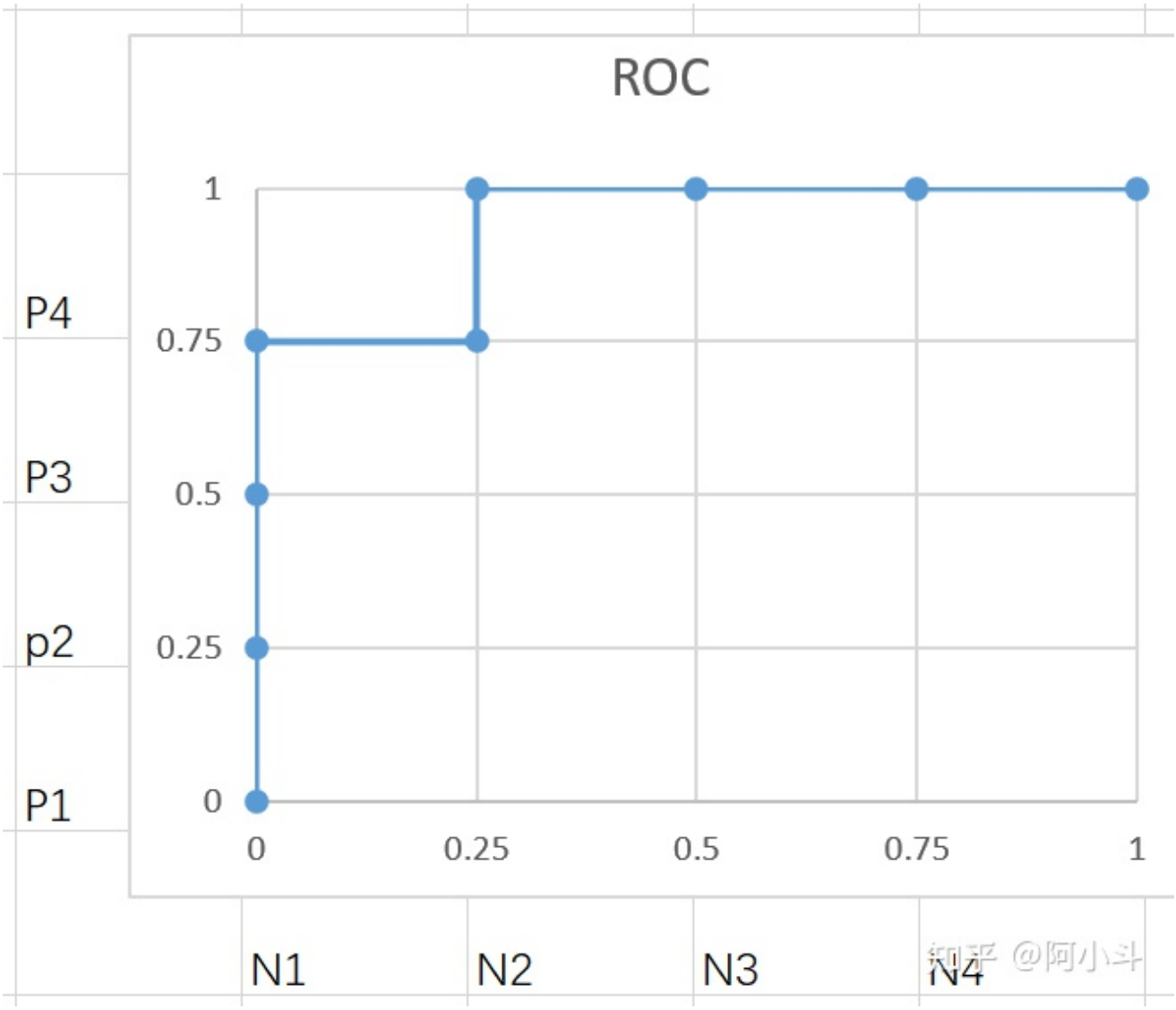
为了更好理解，我们来看一个P4和N1预测错误的情况

P1	P2	P3	N1	P4	N2	N3	N4
0.9	0.8	0.7	0.6	0.4	0.3	0.2	0.1
1	1	1	0	1	0	0	0
预测为1	预测为1	预测为1	预测为1	预测为0	预测为0	预测为0	预测为0

和上面完美算法结果相比，这个例子中我们有一个N1比P4预测概率还要大，根据逻辑回归0.5的界限的话，这个N1我们预测为1

我们同样按照上述步骤绘制ROC曲线

- 从P1开始，用阈值0.9（或者不说阈值，就说值考虑P1），只有P1是预测为1，其余都预测为0，那么TPR =1/4=0.25，FPR=0，ROC点（0, 0.25）
- 再考虑P2，用阈值0.8，那么P1,P2预测为1，那么TPR =2/4=0.5，FPR=0. ROC点（0, 0.5）
- 考虑P3, 同理得到ROC点（0, 0.75）
- 这里个上面有区别了，P3下面是N1，所以我们要向右移动了不再上向上移动，那么ROC点为（0.25, 0.75）
- N1下面是P4，所以又要向上移动了，ROC点是（0.25, 1）
- 后面都是N于是都往右移动，（ 0.5,1 ）（ 0.75,1 ）（1,1）

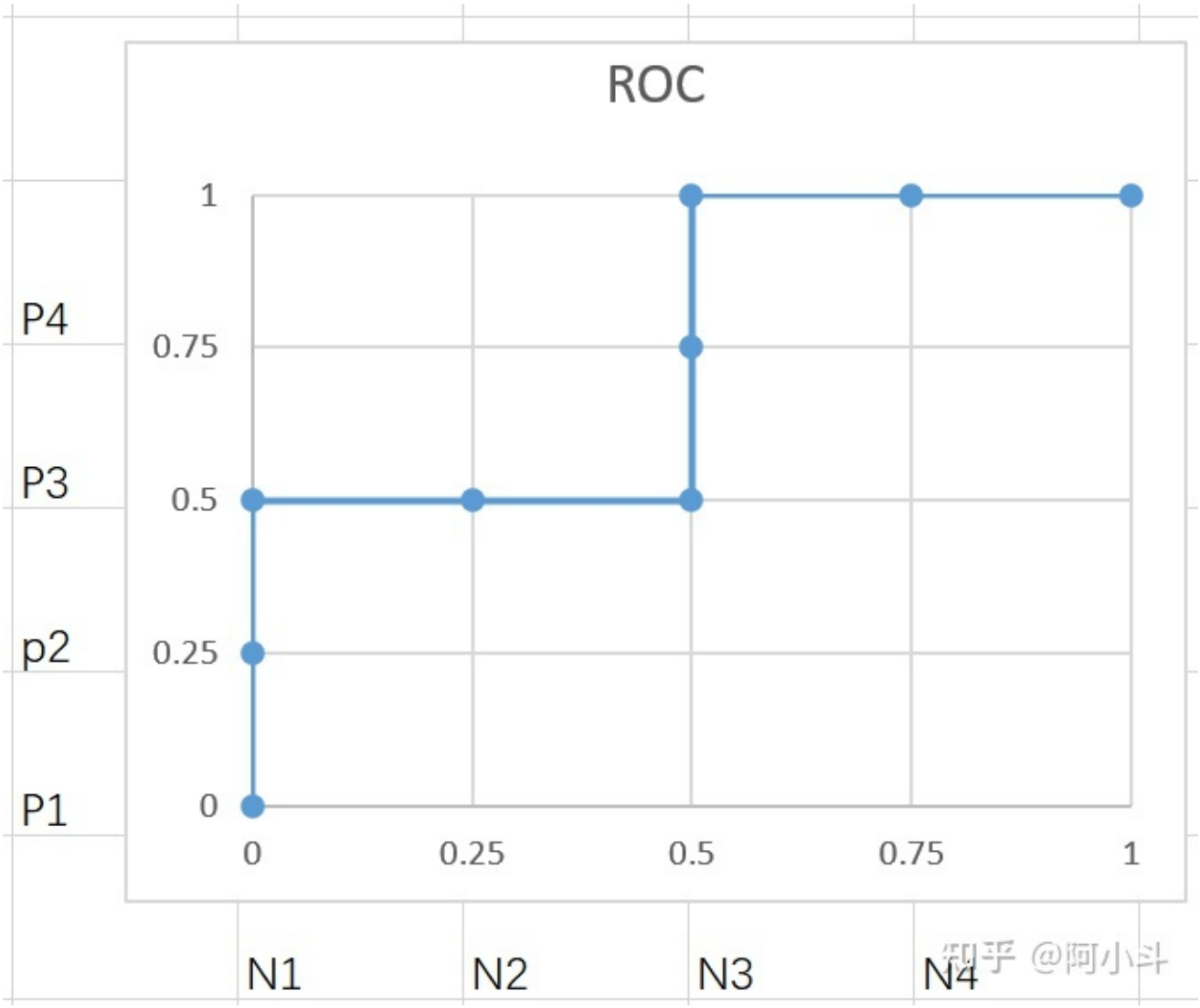


可以看到曲线包围少了一块矩形，和（P4，N1）这个少了的样本对正好对应，所以我们可以得出ROC包住的矩形块数是随机取一对正负样本，正样本预测概率大于负样本预测概率的对数

为了更好理解，我们再来看一个P3,P4 N1 N2搞错的例子

P1	P2	N1	N2	P3	P4	N3	N4
0.9	0.8	0.7	0.6	0.4	0.3	0.2	0.1
1	1	0	0	1	1	0	0
预测为1	预测为1	预测为1	预测为1	预测为0	预测为0	预测为0	预测为0

绘制ROC曲线，从P1开始 网上走，P2继续往上走碰到N1往右，N2继续往右，然后P3P4往上，N3N4往右，最终的ROC曲线



4对样本对，N1，N2 和P3，P4组成负样本概率大于正样本概率的情况，也对应着图形中缺了四个小矩形

而随机取正负样本对，正样本预测概率大于负样本预测概率的对数占比，就等于曲线下矩形个数占总矩形个数的比例，也就是曲线下的面积

所以总结AUC的意义：从所有正样本中随机选择一个样本，从所有负样本中随机选择一个样本，然后根据你的学习器对两个随机样本进行预测，把正样本预测为正例的概率为 p_1 ，把负样本预测为正例的概率为 p_2 ， $p(p_1 > p_2)$ 的值就是AUC

面试回答**AUC**的物理意义：**AUC**表示预测的正例排在负例前面的概率

Chapter 4 PR曲线

	PREDICT POSITIVE	PREDICT NEGTIVE
Real Positive	True Postive(TP)	False Negtive(FN)
Real Negtive	False Positve(FP)	True Negtive(TN)

精准率（Precision）又叫查准率，它是针对预测结果而言的，它的含义是在所有被预测为正的样本中实际为正的样本的概率，意思就是在预测为正样本的结果中，我们有多少把握可以预测正确，其公式如下：

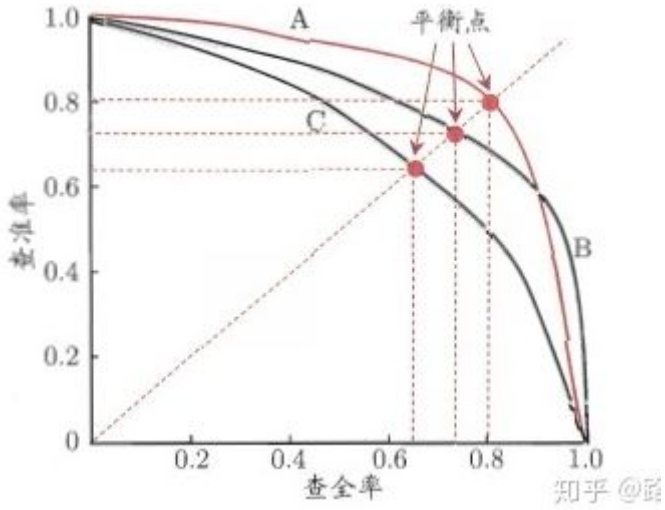
精准率 = TP/(TP+FP)

召回率（Recall）又叫查全率，它是对原样本而言的，它的含义是在实际为正的样本中被预测为正样本的概率，其公式如下：

召回率 = TP/(TP+FN)，这个和TPR定义相同

召回率的应用场景：比如拿网贷违约率为例，相对好用户，我们更关心坏用户，不能错放过任何一个坏用户。因为如果我们过多的将坏用户当成好用户，这样后续可能发生的违约金额会远超过好用户偿还的借贷利息金额，造成严重偿失。所以我们需要将真实阳性样本中被我预测成阳性样本的比例尽量提高，也就是提高召回率（而尽管这样会将FPR也提高）。所以召回率越高，实际阳性样本被预测成阳性的比例越高，同时作为collateral damage，实际阴性样本中被预测成阳性的比例也越高。所以形象来说提高召回率就是：宁可错杀一千（FPR），不能放过一个（TPR）

Precision为纵轴，Recall为横轴，图形如下：



评估分类器的性能，一个比较好的方法是：观察当阈值变化是，precision和recall值的变化情况。如果一个分类器性能比较好，那么应该让recall值增长的同时，保持precision值在一个较高的水平。而性能比较差的分类器，是通过损失很多precision值才能换来较高的recall。

若一个学习器B的P-R曲线包住另一个学习器A的P-R曲线，则称：B的性能优于A。若A和B的曲线发生了交叉，则谁的曲线下的面积大，谁的性能更优。但一般来说，曲线下的面积是很难进行估算的，所以衍生出了“平衡点”（Break-Event Point，简称BEP），即当P=R时的取值，平衡点的取值越高，性能更优。

Chapter 5 ROC和PR的关系

研究问题的特征：关不关心负样本？

数据的特征：负样本数据量是否远大于正样本数据量？

PR优于ROC的情况：不关心负样本，同时负样本数据量远大于正样本数据量

	PREDICT POSITIVE	PREDICT NEGATIVE
Real Positive	True Positive(TP)	False Negative(FN)
Real Negative	False Posiitve(FP)	True Negative(TN)

ROC横轴是第二行，纵轴是第一行

PR横轴是第一行，纵轴是第一列

关心负样本的问题，拿猫和狗分类举例，我们关心的是猫的图片尽量被预测为猫，狗的图片尽量被预测为狗，我们并不关心猫和狗样本之间的关系

不关心负样本的问题，比如癌症，我们根本不用关心没得癌症的人，而重点全在True Positive上

关不关心负样本，数据量向负样本倾斜都可能会发生，下表情况都有可能发生，但是只有不关心负样本时下表情况才会成为一个问题，才会让我们觉得这个算法不行。

换句话说，不关心负样本同时数据量向负样本倾斜会出现我们不能忽视的情况，而这个情况用ROC还区分不了，那就只能用PR。

	PREDICT POSITIVE	PREDICT NEGTIVE
Real Positive	80	20
Real Negtive	1520	13680

网上说负样本数量大于正样本数量就用PR，应该是为了简单：我不管研究的问题是什么，只要数据往负向倾斜，我就用PR

Chapter 6 F-Score

F-Measure是Precision和Recall的加权调和平均，能兼顾精确率和召回率

$$\frac{1}{F_\beta} = \frac{1}{1+\beta^2}(\frac{1}{P} + \frac{\beta^2}{R})$$

$$so$$

$$F_\beta = \frac{(1+\beta^2)PR}{\beta^2 P + R}$$

当 $\beta = 1$ 时，F 为调和平均

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

带入P和R本身的公式

$$F = \frac{2TP}{\text{总样本数} + TP - TN}$$

Chapter 7 混淆矩阵

混淆矩阵（Confusion Matrix）又被称为错误矩阵，通过它可以直观地观察到算法的效果。它的每一列是样本的预测分类，每一行是样本的真实分类（反过来也可以），顾名思义，它反映了分类结果的混淆程度。对之进行可视化，对角线代表预测正确的样本数