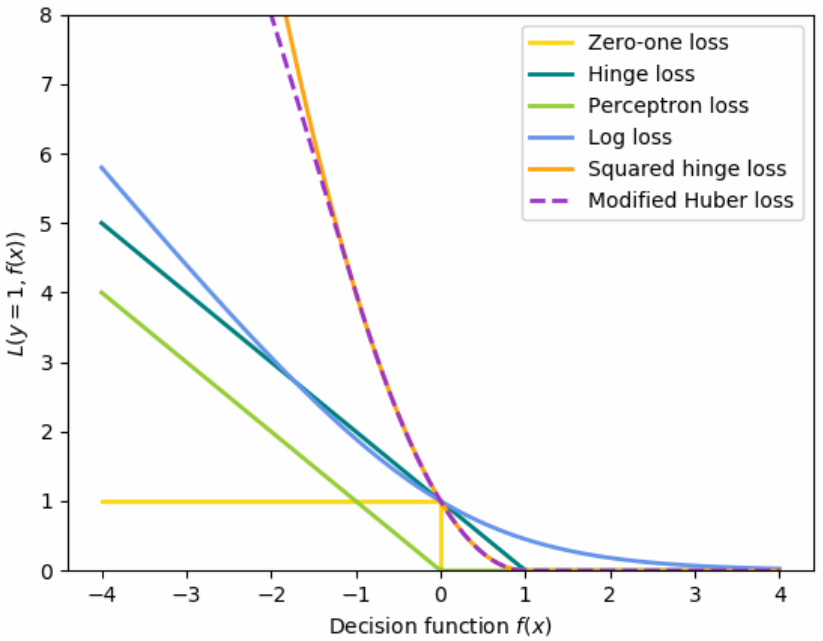
ML

基础

- 有监督学习和无监督学习的区别（根据数据有无标签）
- 说说生成模型和判别模型
    - 生成模型学习的是**联合概率分布 P(X,Y)**，而判别模型直接学习**条件概率分布 P(Y|X)**
    - 生成模型和判别模型的 **loss**：**负对数似然，**-lnP(X,Y)和-lnP(Y|X)

损失函数

- 常见 loss:
    - 负对数似然损失【LR】：它假设样本服从**伯努利分布**（0-1 分布）
    - 平方损失【Linear Regression】：它假设样本和噪声都服从**高斯分布**
    - 指数损失【AdaBoost】
    - 合页损失【SVM】



- 许多网络的输出都可以看作是条件概率分布 **P(Y|X)**，这时使用**最大似然估计（MLE）**对应的代价函数和**交叉熵（cross-entropy）损失**是***等价的***。（此外其他一些的代价函数如**均方误差（MSE）**可以看作**交叉熵损失的一种特殊情况**，即输出概率分布是**高斯条件分布**。）

模型评估&模型选择

- 极大似然估计 MLE(Maximum Likelihood Estimation)和最大后验估计 MAP(Maximum a Posterior)的区别
    - MLE 属于频率学派，MAP 属于贝叶斯学派
    - MLE 是 MAP 的一个特殊情况，其先验是均匀分布；MAP 相当于正则化的 MLE

- Bias 和 variance 的含义，与欠拟合和过拟合有什么联系？结合 ensemble 方法问（见 ensemble 部分）
- 性能度量方法：Accuracy、Precision、Recall、F1 score、PRC、ROC、AUC。各自的优缺点是什么？

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)

$$\text{precision} = \frac{TP}{TP+FP} \quad \text{recall} = \frac{TP}{TP+FN} \quad F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

$$FPR = \frac{FP}{FP+TN} \quad TPR = \frac{TP}{TP+FN}$$

  - ”对于一个给定数目的正负样本数据集，一条曲线在 ROC 空间中比另一条曲线有优势，当且仅当第一条曲线在 PR 空间中也比第二条曲线有优势“，而当这两个曲线在图形中有交错的时候， 无论是 ROC 还是 PRC 都不能确定地告诉你哪个模型的好坏，你只能用 AUC 来比较两个模型。但很多问题中 ROC/PRC 曲线不同位置的面积并不能简单等价。
- 评估模型一般有什么手段（分类问题，回归问题）？常见的分类模型的评价指标（顺便问问交叉熵、ROC 如何绘制、AUC 的物理含义、假设检验，类别不均衡样本）

正则化

- 过拟合的原因?（数据太少 or 模型太复杂）
- 从贝叶斯估计的角度来看，正则化项对应于模型的先验概率。方法有很多：L1 正则、L2 正则（weight decay）、early stop、data augmentation（数据集扩增）、dropout，drop connect
  - Dropout 是在训练过程中以一定概率 1-p 将隐含层节点的输出值清 0，而用 bp 更新权值时，不再更新与该节点相连的权值。DropConnect 和 Dropout 相似的地方在于它涉及在模型中引入稀疏性，不同之处在于它引入的是权重的稀疏性而不是层的输出向量的稀疏性。
  - Dropout 和 DropConnect 都类似模型平均，Dropout 是 2^|m|个模型的平均，而 DropConnect 是 2^|M|个模型的平均（m 是向量，M 是矩阵，取模表示矩阵或向量中对应元素的个数），从这点上来说，DropConnect 模型平均能力更强（因为|M|>|m|）。
- L0，L1，L2 的。 L1 为什么能让参数稀疏？L2 为什么会让参数趋于较小值（防过拟合）？
  - L_p-范数：$\|\vec{x}\|_p = \left(\sum_{i=1}^{d} x_i^p\right)^{1/p}$
  - L0 范数是指向量中非 0 元素的个数
  - L1 范数是向量中各个元素的绝对值求和
  - L2 范数是指向量的各个元素平方求和然后取和的平方根

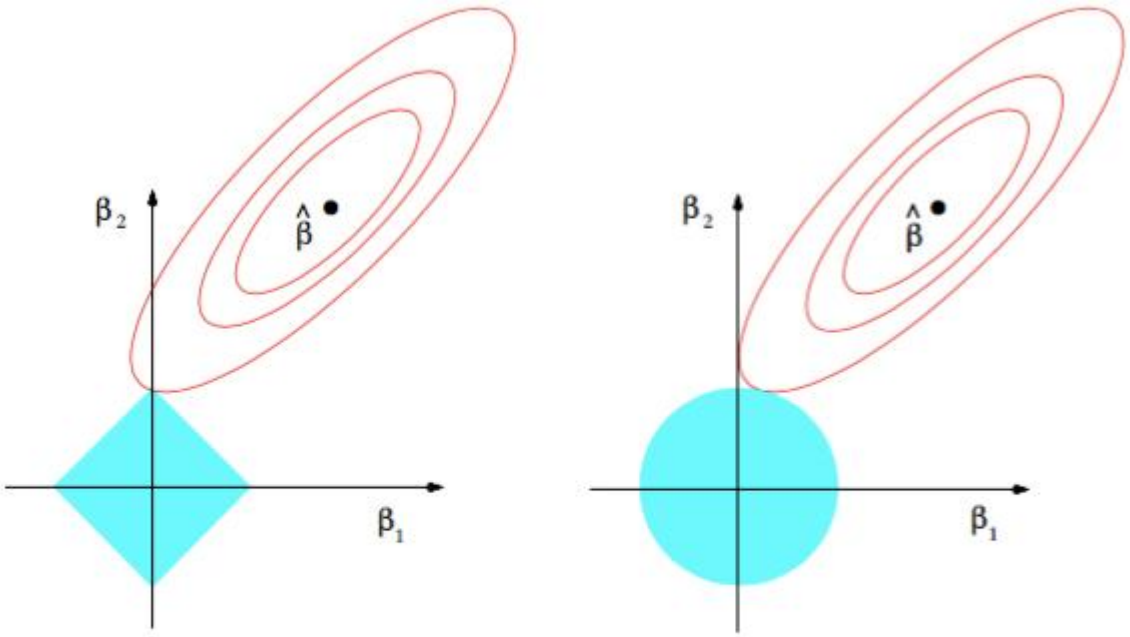
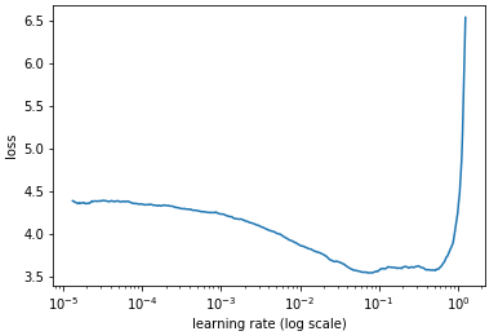


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

- L1 在 0 处不可导，怎么处理（暂不深入）
    - 使用 proximal operator。借助软阈值(Soft Thresholding)函数解决
- 深度学习中防止过拟合的方法？
    - 如正则化(weight decay)，增加训练样本（data augmentation），early stop，dropout，relu，等。怎么处理 weight decay 的权重。

优化

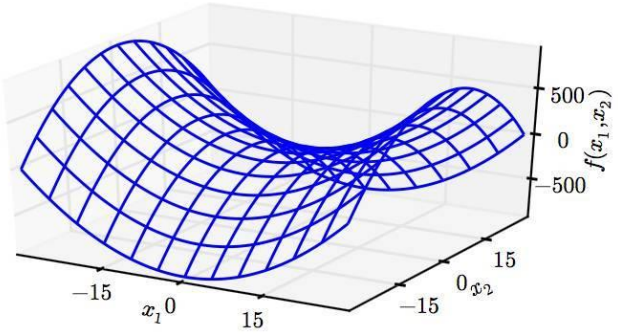
- 对所有优化问题来说, 有没有可能找到比現在已知算法更好的算法?（**没有免费午餐定理**）
- 学习率的选择
    - 初始学习率从一个低学习率开始训练，并在每个批次中指数提高，找到 loss 快速上升的点，就是最大值，然后随着迭代的继续逐步减小学习率，使得模型在训练后期更加稳定，在训练后面指数衰减（exponential_decay）
    - 

- 当机器学习性能不是很好时，你会如何优化？
    - 数据：更多数据、数据清洗、重新取样、特征工程
    - 算法：调参、单一数值评价指标、模型融合

- 深度学习的优化方法有哪些？ sgd、adam、adgrad 区别？ adagrad 详细说一下？为什么 adagrad 适合处理稀疏梯度？（暂时跳过吧）
- 各种最优化方法比较，BGD，SGD，优缺点，优化方向(Adam 之类)（暂不深入）
  - 首先定义：待优化参数： w ，目标函数： $f(w)$ ，初始学习率 α 。

    而后，开始进行迭代优化。在每个epoch t ：

    1. 计算目标函数关于当前参数的梯度： $g_t = \nabla f(w_t)$
    2. 根据历史梯度计算一阶动量和二阶动量： $m_t = \phi(g_1, g_2, \cdots, g_t); V_t = \psi(g_1, g_2, \cdots, g_t)$ ，
    3. 计算当前时刻的下降梯度： $\eta_t = \alpha \cdot m_t / \sqrt{V_t}$
    4. 根据下降梯度进行更新： $w_{t+1} = w_t - \eta_t$
- 梯度下降法的神经网络容易收敛到局部最优吗（No）？为什么网络够深(Neurons 足够多)的时候，总是可以避开较差 Local Optima？
  - 因为你的直觉是错的。在高维空间中绝大多数梯度值为 0 的点不是上图所示的 local minima，而是 saddle point。假设在一个 20,000 维的参数空间中，如果某个点梯度值为 0，那么在每个方向上既可以是凸（convex）函数也可以是凹（concave）函数。但要想该点成为 local minima 的话，所有的 20,000 个方向都必须是凸的，在神经网络构成的巨大的参数空间中，这个概率是十分小的。
  - 【https://www.zhihu.com/question/68109802/answer/261755787】



- 深度学习为什么不用二阶优化？（计算复杂度高，不适合高维下）
- 拟牛顿法和牛顿法区别，哪个收敛快？（拟牛顿法用一个正定阵来近似海塞矩阵的逆，简化了计算）
- 梯度下降法 & 牛顿法 （泰勒展开）

梯度下降法（Gradient Descend Method）

在机器学习任务中，需要最小化损失函数 $L(\theta)$，其中 θ 是要求解的模型参数。梯度下降法常用来求解这种无约束最优化问题，它是一种迭代方法：选取初值 θ^0，不断迭代，更新 θ 的值，进行损失函数的极小化。

- 迭代公式： $\theta^t = \theta^{t-1} + \Delta\theta$
- 将 $L(\theta^t)$ 在 θ^{t-1} 处进行一阶泰勒展开：

$$\begin{aligned} L(\theta^t) &= L(\theta^{t-1} + \Delta\theta) \\ &\approx L(\theta^{t-1}) + L'(\theta^{t-1})\Delta\theta \end{aligned}$$

- 要使得 $L(\theta^t) < L(\theta^{t-1})$，可取：$\Delta\theta = -\alpha L'(\theta^{t-1})$，则：$\theta^t = \theta^{t-1} - \alpha L'(\theta^{t-1})$

这里 α 是步长，可通过line search确定，但一般直接赋一个小的数。

牛顿法（Newton's Method）

- 将 $L(\theta^t)$ 在 θ^{t-1} 处进行二阶泰勒展开：

$$L(\theta^t) \approx L(\theta^{t-1}) + L'(\theta^{t-1})\Delta\theta + L''(\theta^{t-1})\frac{\Delta\theta^2}{2}$$

为了简化分析过程，假设参数是标量（即 θ 只有一维），则可将一阶和二阶导数分别记为 g 和 h：

$$L(\theta^t) \approx L(\theta^{t-1}) + g\Delta\theta + h\frac{\Delta\theta^2}{2}$$

- 要使得 $L(\theta^t)$ 极小，即让 $g\Delta\theta + h\frac{\Delta\theta^2}{2}$ 极小，可令：$\frac{\partial\left(g\Delta\theta + h\frac{\Delta\theta^2}{2}\right)}{\partial\Delta\theta} = 0$

求得 $\Delta\theta = -\frac{g}{h}$，故 $\theta^t = \theta^{t-1} + \Delta\theta = \theta^{t-1} - \frac{g}{h}$

参数 θ 推广到向量形式，迭代公式：$\theta^t = \theta^{t-1} - H^{-1}g$

这里 H 是海森矩阵

从Gradient Descend 到 Gradient Boosting

参数空间

$$\theta^t = \theta^{t-1} + \theta_t$$

第t次迭代后的参数　第t-1次迭代后的参数　第t次迭代的参数增量

$$\theta_t = -\alpha_t g_t$$

参数更新方向为负梯度方向

$$\theta = \sum_{t=0}^{T} \theta_t$$

最终参数等于每次迭代的增量的累加和，

θ_0 为初值

函数空间

$$f^t(x) = f^{t-1}(x) + f_t(x)$$

第t次迭代后的函数　第t-1次迭代后的函数　第t次迭代的函数增量

$$f_t(x) = -\alpha_t g_t(x)$$

同样地，拟合负梯度

$$F(x) = \sum_{t=0}^{T} f_t(x)$$

最终函数等于每次迭代的增量的累加和，

$f_0(x)$ 为模型初始值，通常为常数

从Newton's Method 到 Newton Boosting

参数空间

$$\theta^t = \theta^{t-1} + \theta_t$$

第t次迭代后的参数　第t-1次迭代后的参数　第t次迭代的参数增量

$$\theta_t = -H_t^{-1} g_t$$

与梯度下降法唯一不同的就是参数增量

$$\theta = \sum_{t=0}^{T} \theta_t$$

最终参数等于每次迭代的增量的累加和，

θ_0 为初值

函数空间

$$f^t(x) = f^{t-1}(x) + f_t(x)$$

第t次迭代后的函数　第t-1次迭代后的函数　第t次迭代的函数增量

$$f_t(x) = -\frac{g_t(x)}{h_t(x)}$$

$$F(x) = \sum_{t=0}^{T} f_t(x)$$

最终函数等于每次迭代的增量的累加和，

$f_0(x)$ 为模型初始值，通常为常数

实践

数据

- 如果分类样本的标签只有一定的概率可信，如何处理？（数据清洗：完整性、唯一性、合法性、一致性）
- 数据清洗方法，缺失值处理方法，缺失值填充方法？
- 为什么要做归一化？哪些模型要做归一化？哪些不用？归一化的方式？

    - 广义线性模型要做归一化（KNN LR SVM NN）；概率模型和树模型不需要（NB DT）
    - 优点：
        - feature scaling 会使 gradient descent 的收敛更好
        - 可能提高精度
    - 常用的归一化方法有以下两种：
        - 1. min-max 标准化：也称为离差标准化，是对原始数据的线性变换，使结果值映射到[0 - 1]之间，x*=(x-min)/(max-min)，缺陷就是当有新数据加入时，可能导致 max 和 min 的变化，需要重新定义。
        - 2. Z-score 标准化方法：这种方法给予原始数据的均值（mean）和标准差（standard deviation）进行数据的标准化。经过处理的数据符合标准正态分布，即均值为 0，标准差为 1，x*=(x-μ)/σ
- 数据 imbalance 怎么办？
    - 不用 accuracy，用 precision 和 recall 或 F1 score
    - Undersampling 或 oversampling 或 改变样本权重
- 数据 sparse 怎么办？（数据平滑（data smoothing）。注意和特征稀疏区的问题区分开。）
- 异常值的影响，如何消除？（跳过）

特征

- 特征工程有哪些方法（数据处理上的） 主要是统计意义上的一些操作：(1.OneHotEncoding 2.标准化 3.归一化 4.连续特征离散化 5.L1 正则 6.GBDT 特征组合 7.特征重要性分析方法 卡方检验 8.FM 实现离散特征 embeding)
- 怎么样提的特征 做了哪些特征工程 说说对特征工程的理解
- 怎么做特征选择（包裹式，过滤式，嵌入式），怎么做数据压缩（PCA，自编码）

样例分析

- 如何构建欺诈交易识别的模型？
    - 基本的二分类问题，要处理样本不平衡的问题。主要是做好特征工程（分析、可视化、特征选择）
- 如何继续提升（模型融合提升的效果肯定没特征融合以及高阶特征效果好，按照这个思路回答的，对方满意）

NB

- **原理**
    - 基于贝叶斯定理和特征条件独立假设，学习 X 和 Y 联合概率分布 P(X,Y)，多类分类器
- **公式推导（见草稿）**

    - **loss：**对数似然函数

- **优化&trick：**
  - 概率计算公式，EM，
  - 后验概率计算，运用极大似然估计（加上拉普拉斯平滑，可运用正态分布概率密度函数）

$$\phi(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

- **适用性&优缺点**
  - 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。（离散、连续，值极大极小之类的）
- **比较**
- 问题
  - 对没出现词的解决：如 laplace 平滑等
  - 遇到特征不独立的问题：参考改进的贝叶斯网络，使用 DAG 来进行概率图的描述

KNN

- 介绍一下（原理&距离度量&归一化）
  - 属于在线学习，不用训练，没有 loss，唯一参数是 k
  - 找到距离最近的 k 个，投票表决决定类别(可以加权投票)
  - 对 n 维实数向量空间 Rn，经常用 Lp 距离，
  - $L_p(x_i,x_j)=\left(\sum_{l=1}^{n}|x_i^{(l)}-x_j^{(l)}|^p\right)^{\frac{1}{p}}$ L1 是曼哈顿距离，L2 是欧式距离
  - 为了保证每个特征同等重要性，我们这里对每个特征进行归一化。
- 适用性&优缺点
  - 适用于多类分类，回归
  - 优点：
    - 1.KNN 分类方法是一种非参数的分类技术，简单直观，易于实现！只要让预测点分别和训练数据求距离，挑选前 k 个即可，非常简单直观。
    - 2.KNN 是一种在线技术，新数据可以直接加入数据集而不必进行重新训练
  - 缺点及改进：
    - 1.当样本不平衡时，比如一个类的样本容量很大，其他类的样本容量很小，输入一个样本的时候，K 个邻近值大多数都是大样本容量的那个类，这时可能会导致分类错误。
      - 改进方法：对 K 邻近点进行加权，也就是距离近的权值大，距离远的点权值小。
    - 2.计算量较大，每个待分类的样本都要计算它到全部点的距离，根据距离排序才能求得 K 个临近点。
      - 改进方法：先对已知样本带你进行裁剪，事先去除分类作用不大的样本，采取 kd 树以及其它高级搜索方法 BBF 等算法减少搜索时间。

- kd 树（高维特征最近领域搜索算法）
    - kd 树是二叉树，表示对 k 维空间的划分
    - 通常依次选择维度，选择实例在选定维度上的中位数为切分点，得到是平衡树
    - kd 树的搜索（参考书上）
- KNN 中的 K 怎么选择？

  李航博士《统计学习方法》的书上式这么写的
    - 在实际应用中，K 值一般取一个比较小的数值，例如采用交叉验证法（简单来说，就是一部分样本做训练集，一部分做测试集）来选择最优的 K 值。
    - 如果选择较小的 K 值，就相当于用较小的领域中的训练实例进行预测，“学习”近似误差会减小，只有与输入实例较近或相似的训练实例才会对预测结果起作用，与此同时带来的问题是“学习”的估计误差会增大，换句话说，K 值的减小就意味着整体模型变得复杂，容易发生过拟合；
    - 如果选择较大的 K 值，就相当于用较大领域中的训练实例进行预测，其优点是可以减少学习的估计误差，但缺点是学习的近似误差会增大。这时候，与输入实例较远（不相似的）训练实例也会对预测器作用，使预测发生错误，且 K 值的增大就意味着整体的模型变得简单。
    - K=N，则完全不足取，因为 此时无论输入实例是什么，都只是简单的预测它属于在训练实例中最多的累，模型过于简单，忽略了训练实例中大量有用信息。

★LR（分类）

- **原理**
    - LR 的原理？
        - 是由条件概率分布 **P(Y|X)**表示的，**输出 y 的对数几率是输入 x 的线性函数（联系 softmax 和 sigmoid 来说）**
            - 标签-1 和 1 情况下的目标，和 0 1 的区别（是有本质区别的）
                - 联系 softmax 概率函数，标签 0 1 时，负标签输入 z 固定为 0；当标签-1 1 时，那负标签的输入 z 就固定为-1 了，此时的 sigmoid 函数图像是左移了一个单位的，在 z=-1 处正例概率为½
        - LR 为什么用 sigmoid？
            - **对输入 x，z=wx+b，其概率正比于 exp(z)**，在二分类是就是 sigmoid，在多分类时就是 softmax。（这里假设 Y|X 服从 bernoulli distribution，根据最大熵原则，可以推出熵最大的概率分布是指数分布

              【https://www.zhihu.com/question/35322351】）
        - Softmax loss 函数是什么
            - Softmax 回归是逻辑回归在多分类的推广，相应的模型也可以叫做多元逻辑回归

$$P(y=i|x,\theta)=\frac{e^{\theta_i^T x}}{\sum_j^K e^{\theta_j^T x}} \quad y^*=\text{argmax}_i P(y=i|x,\theta) \quad J(\theta)=-\frac{1}{N}\sum_i^N\sum_j^K 1[y_i=j]\log\frac{e^{\theta_i^T x}}{\sum e^{\theta_k^T x}}$$

    - **最大熵原则**，从最大似然估计导出**负对数损失**

      根据非真实分布 q 得到的平均编码长度 H(p,q)大于根据真实分布 p 得到的平均编码长度 H(p)

      在信息论中，基于相同事件测度的两个概率分布p和q的**交叉熵**是指，当基于一个“非自然”（相对于“真实”分布p而言）的概率分布q进行编码时，在事件集合中唯一标识一个事件所需要的平均比特数（bit）。

      基于概率分布p和q的交叉熵定义为：

      $$H(p,q)=\mathrm{E}_p[-\log q]=H(p)+D_{\mathrm{KL}}(p\|q),$$

      其中$H(p)$是p的熵，$D_{\mathrm{KL}}(p\|q)$是从p到q的KL散度(也被称为p相对于q的*相对熵*)。

      对于离散分布p和q，这意味着：

      $$H(p,q)=-\sum_x p(x)\,\log q(x).$$

      （伯努利分布的数学期望）（伯努利分布：1 重伯努利试验；二项分布：n 重伯努利试验「成功」次数离散概率分布）

      【https://www.zhihu.com/question/41252833/answer/108777563】

- **公式推导（见草稿）**
    - 手推 LR 公式，loss 函数的推导， 推导 sgd 的参数更新结果（见草稿）
    - **loss：**
        - 从最大似然推导出
        - 为什么选用对数极大似然函数作为优化目标，用平方 loss 有什么问题。
            - 首先，的确可以使用「普通最小二乘」，也就是 OLS 做 Y 为 0/1 的回归。但是我们一般不用，为什么呢？因为一般我们的 Y 为 0/1 的时候，我们想得到的是 Y=1 的概率，而概率是不能小于 0，不能大于 1 的，而**用 OLS 则很容易出现小于 0 或者大于 1 的概率预测值。**【https://www.zhihu.com/question/23817253】
    - **优化&trick**
        - LR 的训练方法（除了 SGD，BGD）
            - 牛顿法和拟牛顿法（见上面基础部分）
            - 还有改进的迭代尺度算法（略）
- **适用性&优缺点**
- **比较**
- 逻辑回归对特征有什么要求，是否需**要做离散化**，离散化的好处与坏处。
    - 对**异常数据的鲁棒性、可特征交叉引入非线性、简化模型**降低过拟合

      【https://www.zhihu.com/question/31989952】
- 逻辑回归为何线性模型

    - logistics 是广义线性模型，只是在特征到结果的映射中加入了一层函数映射，即先把特征线性求和，然后使用函数 g(z)将最为假设函数来预测。g(z)可以将连续值映射到 0 和 1 上。
- 线性回归，最优解表达式如下，是解析解，不好求
    - $\theta = (X^T X)^{-1} X^T y$
- 逻辑回归能不能用来做非线性分类？（可以，对特征做非线性变换 比如 kernel）
- 逻辑回归的参数是否可以分布式求解，如何做分布式？（把数据集划分成 C 块，分别求其对 w 偏导）
    - 如我们所知，Logistic 回归偏导数的求法是这样的：

$$\frac{\partial J}{\partial w} = \frac{1}{N}\sum_{i=1}^{N}(h_\theta(x^{(i)}) - y^{(i)}) \times x^{(i)}$$

    - 在这里，我们如果把数据集打散（注：即“划分”，要满足不重、不漏两个条件）成 C 块

$$\frac{\partial J}{\partial w} = \frac{1}{N}\sum_{k=1}^{K}\sum_{i \in C_k}(h_\theta(x^{(i)}) - y^{(i)}) \times x^{(i)}$$

    - 就可以加速了，就是这么简单。

★ SVM

- **原理**
    - 是一种二分类模型，是特征空间上的间隔最大的线性分类器，学习策略是**间隔最大化**，学习算法是求解一个**凸二次规划的最优化问题**。
- **公式推导（见草稿）**（问得最多）
    - 手推 svm 算法，svm 推导一直到序列最小化(SMO)求解

Linear SVM dual formulation - The lagrangian

$$\begin{cases} \min\limits_{\mathbf{w},b} & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{with} & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \qquad i = 1, n \end{cases}$$

Looking for the lagrangian saddle point $\max\limits_{\alpha}\min\limits_{\mathbf{w},b}\mathcal{L}(\mathbf{w}, b, \alpha)$ with so called lagrange multipliers $\alpha_i \geq 0$

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n}\alpha_i\left(y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1\right)$$

α_i represents the influence of constraint thus the influence of the training example (x_i, y_i)

    - **loss**
        - 合页损失(hinge loss)（左移一个单元就是感知机 Loss）

- **SVM in the "natural" form**

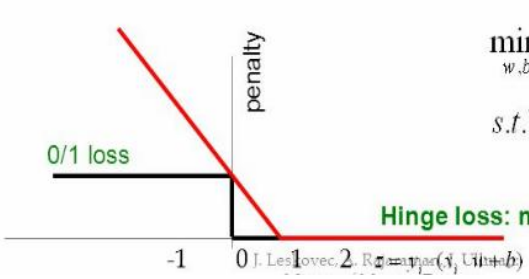
$$\arg\min_{w,b} \frac{1}{2} w \cdot w + C \cdot \sum_{i=1}^{n} \max\{0, 1 - y_i(w \cdot x_i + b)\}$$

Margin

Regularization parameter

Empirical **loss L** (how well we fit training data)

- **SVM uses "Hinge Loss":**



$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i,\ y_i \cdot (w \cdot x_i + b) \geq 1 - \xi_i$$

- **优化&trick**
  - 为什么要用拉格朗日乘数法，对偶问题是什么，为什么要转换成对偶问题，两者的适用情况分别是?（通过求对偶问题得到原始问题最优解；**优点：容易求解、引入核函数**）
  - KKT 条件都有什么？（物理几何意义？？先跳过）

$$\frac{\partial}{\partial w_\nu} L_P = w_\nu - \sum_i \alpha_i y_i x_{i\nu} = 0 \quad \nu = 1, \cdots, d \tag{17}$$

$$\frac{\partial}{\partial b} L_P = -\sum_i \alpha_i y_i = 0 \tag{18}$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad i = 1, \cdots, l \tag{19}$$

$$\alpha_i \geq 0 \quad \forall i \tag{20}$$

$$\alpha_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1) = 0 \quad \forall i \tag{21}$$

  - 如何求解 svm 的最优化问题（SMO 序列最小最优化，SMO 算法原理）

    SMO 要求解如下凸二次规划的对偶问题（原始问题推导出的的对偶问题）：

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{N}\alpha_i$$

$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i y_i = 0$$

$$0 \leqslant \alpha_i \leqslant C, \quad i = 1, 2, \cdots, N$$

    按照坐标下降法的思路（对 max_f(x1,x2,...,xn)，固定其他 n-1 维，求 x_i 使 f 最小）。

    不同的是，根据限制条件，需要一次选取两个参数做优化，比如α_1和α_2，此时α_1可以由α_2和其他参数表示出来。这样回带到优化问题中，可解。可以通过更改优化顺序来使 W 能够更快地增加并收敛。

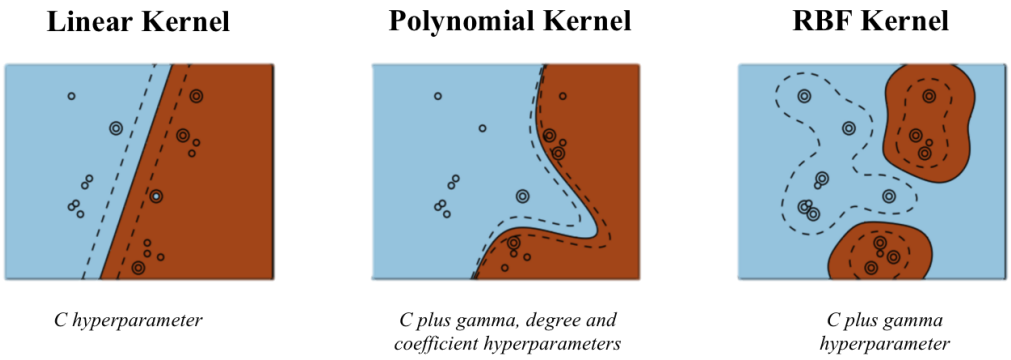
  - 对于非线性核的理解？核函数如何体现？

    核函数：**K(x,z)=Φ(x)·Φ(z)**，（**映射函数的内积**）

    其中 Φ(x): X->H 为输入空间到特征空间的映射函数，Φ(x)·Φ(z)为内积运算

    在 SVM 的对偶问题中，目标函数和决策函数都只涉及输入实例与实例之间的内积 **xi·xj**，可以用核函数来代替，这**等价于经过映射函数 Φ 对输入空间做了一次变换**。

- 核函数有哪些类型，如何选择核函数？



一般用线性核和高斯核，也就是 Linear 核与 RBF 核

Linear 核：主要用于线性可分的情形。参数少，速度快。

RBF 核：主要用于线性不可分的情形。参数多，分类结果非常依赖于参数。

使用 libsvm，默认参数，RBF 核比 Linear 核效果稍差。通过进行大量参数的尝试，一般能找到比 linear 核更好的效果。

然后一般情况下 RBF 效果是不会差于 Linear

下面是吴恩达的见解：

1. 如果 Feature 的数量很大，跟样本数量差不多，这时候选用 LR 或者是 Linear Kernel 的 SVM

2. 如果 Feature 的数量比较小，样本数量一般，不算大也不算小，选用 SVM+Gaussian Kernel

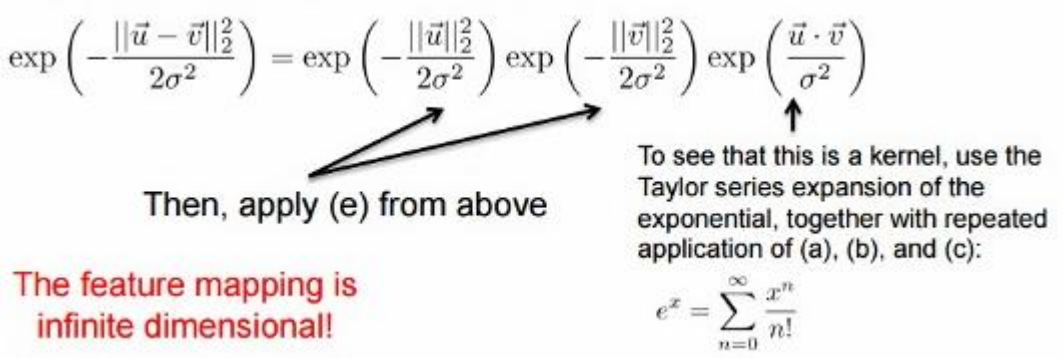
- 为什么 svm 中的高斯核能映射到无穷维?

以二维升维为例：

$$k(x,y) = (x1y1 + x2y2)^2 = x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2$$

则 $\Phi(x) = (x_1^2, \sqrt{2}x_1^2x_2^2, x_2^2)$ 将 R^2 的点映射到 R^3

对核函数找到特征映射，要**展开为乘积**。对高斯核中的**指数式**可以作**泰勒展开到无穷项（所以核函数是一种特殊的多项式核）**



【https://www.zhihu.com/question/35602879】

- **适用性&优缺点**
  - SVM 适合求解什么类的问题

        - 小样本、高维（比如参数数量大于样本数等等）、非线性可分（松弛变量和核函数）
    - 优点：
        - 分类思想简单，就是间隔最大化，效果好
        - 使用核函数可以向高维空间进行映射，解决非线性的分类
    - 缺点：
        - 对大规模数据训练比较困难
        - 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；
- **比较**
    - LR 和 LinearSVM 的区别（下面回答），各自的优缺点和应用场景（见各自的部分）

      本质区别是 loss 不同。LR 基于最大熵原理，假设样本符合伯努利分布，通过极大化对数似然函数得到参数估计，是 logistic loss；LinearSVM 基于几何间隔最大化原理，是 hinge loss
        - 都是线性：Linear SVM 和 LR 都是线性分类器
        - 受数据分布的影响：Linear SVM 不直接依赖数据分布，分类平面不受一类点影响；LR 则受所有数据点的影响，如果数据不同类别 strongly unbalance 一般需要先对数据做 balancing
        - 归一化：Linear SVM 依赖数据表达的距离测度，所以需要对数据先做 normalization；LR 不受其影响（但一般最好也归一化，梯度下降会收敛更快）
    - LR 为什么一般不用核函数而 svm 线性不可分的时候用核函数？
        - 分类模型的结果就是计算决策面，模型训练的过程就是决策面的计算过程。通过上面的第二点不同点可以了解，在计算决策面时，**SVM 算法里只有少数几个代表支持向量的样本参与了计算，也就是只有少数几个样本需要参与核计算**（即 kernal machine 解的系数是稀疏的）。然而，LR 算法里，每个样本点都必须参与决策面的计算过程，也就是说，假设我们在 LR 里也运用核函数的原理，那么每个样本点都必须参与核计算，这带来的计算复杂度是相当高的。所以，在具体应用时，LR 很少运用核函数机制。
- 支持向量越多越好还是越少越好
    - 惩罚参数 C 是对支持向量的控制，C 值小时对误分类惩罚小，支持向量多，Margin 也大；
    - 反之，C 值大时，Margin 小，支持向量也少，也更加容易过拟合
- SVM 多分类问题
    - **一对一(libsvm 实现)**：任意两个类都训练一个分类器，n*(n-1)/2 个分类器。
    - 一对多：其中某个类为一类，其余 n-1 个类为另一个类（1 对 M 分类，会存在偏置，很不实用）
    - 直接法：直接在目标函数上进行修改，将多个分类面的参数求解合并到一个最优化问题中，通过求解该优化就可以实现多分类（计算复杂度很高，实现起来较为困难）

★ DT

- **原理**

  决策树学习的本质是从训练数据集上归纳出**一组分类规则**，但与训练集不相矛盾的决策树可能有多个，也可能一个都没有，我们的目标是找到一个和训练集矛盾较小，且具有很好的**泛化能力**的决策树

  - 对于**不纯度**（**impurity**）的度量：

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

    - **信息增益**（偏向选择取值较多的特征）& **信息增益比**

      输出：特征 A 对训练数据集 D 的信息增益 $g(D,A)$.

      （1）计算数据集 D 的经验熵 $H(D)$

      $$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|}$$

      （2）计算特征 A 对数据集 D 的经验条件熵 $H(D|A)$

      $$H(D|A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|}$$

      （3）计算信息增益

      $$g(D,A)=H(D)-H(D|A)$$

      定义（信息增益比）：特征A对训练数据集D的信息增益比$g_R(D,A)$定义为其信息增益$g(D,A)$与训练数据集D关于特征A的值的熵$H_A(D)$之比，即

      $$g_R(D,A)=\frac{g(D,A)}{H_A(D)}$$

      其中，$H_A(D)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\log_2\frac{|D_i|}{|D|}$

    - Gini 系数(简化熵模型的对数运算)

      $$\text{Gini}(p)=\sum_{k=1}^{K}p_k(1-p_k)=1-\sum_{k=1}^{K}p_k^2$$

- **公式推导**

  - **loss**

    - loss 的极小化等价于**正则化的极大似然估计**（对数似然损失）

    树 T 叶节点个数为 |T| ，叶节点 t 上的**经验熵**：

    $$H_t(T)=-\sum_{k}\frac{N_{tk}}{N_t}\log(\frac{N_{tk}}{N_t})$$

    损失函数：

    $$C_\alpha(T)=C(T)+\alpha|T|=\sum_{t=1}^{|T|}N_tH_t(T)+\alpha|T|=-\sum_{t=1}^{|T|}\sum_{k=1}^{K}N_{kt}\log\frac{N_{kt}}{N_t}+\alpha|T|$$

  - **优化&trick**

    - 递归地选择特征和切分点（选择不纯度降低最大的）
      - ID3：信息增益；
      - C4.5：信息增益比；
      - CART：**平方误差[回归]** or **基尼指数[分类]**；
    - 决策树生成（局部最优）

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
属性集 $A = \{a_1, a_2, \ldots, a_d\}$.
过程： 函数 TreeGenerate(D, A)
1: 生成结点 node;
2: **if** D 中样本全属于同一类别 C **then**
3: 将 node 标记为 C 类叶结点; **return**
4: **end if**
5: **if** $A = \varnothing$ **OR** D 中样本在 A 上取值相同 **then**
6: 将 node 标记为叶结点, 其类别标记为 D 中样本数最多的类; **return**
7: **end if**
8: 从 A 中选择最优划分属性 a_*;
9: **for** a_* 的每一个值 a_*^v **do**
10: 为 node 生成一个分支; 令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
11: **if** D_v 为空 **then**
12: 将分支结点标记为叶结点, 其类别标记为 D 中样本最多的类; **return**
13: **else**
14: 以 TreeGenerate(D_v, $A \setminus \{a_*\}$)为分支结点
15: **end if**
16: **end for**
输出： 以 node 为根结点的一棵决策树

- 决策树剪枝（全局最优）（下面的算法为后剪枝）

算法 5.4（树的剪枝算法）
输入： 生成算法产生的整个树 T，参数 α；
输出： 修剪后的子树 T_α.
（1）计算每个结点的经验熵.
（2）递归地从树的叶结点向上回缩.

设一组叶结点回缩到其父结点之前与之后的整体树分别为 T_B 与 T_A，其对应的损失函数值分别是 $C_\alpha(T_B)$ 与 $C_\alpha(T_A)$，如果

$$C_\alpha(T_A) \leqslant C_\alpha(T_B) \tag{5.15}$$

则进行剪枝，即将父结点变为新的叶结点.
（3）返回 (2)，直至不能继续为止，得到损失函数最小的子树 T_α. ■

- 决策树的输出（一颗决策树对应着输入空间的一个划分，以及在划分单元上的输出值。分类就是标签值或投票，回归就是 yi 或平均值）

- **适用性&优缺点**
  - 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**
    - 剪枝，**先剪枝**（限定高度，限定阈值）or **后剪枝**（通过大多数原则，向上减）
    - 后续出现了**随机森林**，减小了过拟合现象
- **比较**
- 缺失值处理
  - 1. *在选择分裂属性的时候*，训练样本存在缺失值，如何处理？

分裂时，计算的增益乘以无缺失的样本比例：计算分裂损失减少值时，忽略特征缺失的样本，最终计算的值乘以比例（实际参与计算的样本数除以总的样本数）

- 假如你使用 ID3 算法，那么选择分类属性时，就要计算所有属性的熵增(信息增益，Gain)。假设 10 个样本，属性是 a,b,c。在计算 a 属性熵时发现，第 10 个样本的 a 属性缺失，那么就把第 10 个样本去掉，前 9 个样本组成新的样本集，在新样本集上按正常方法

计算 a 属性的熵增。然后结果乘 0.9（新样本占 raw 样本的比例），就是 a 属性最终的熵。

- 2. *分类属性选择完成*，对训练样本分类，发现样本属性缺失怎么办？

  训练时，让同一样本以不同权重划入所有子节点：将该样本分配到所有子节点中，权重由 1 变为具有属性 a 的样本被划分成的子集样本个数的相对比率，计算错误率的时候，需要考虑到样本权重。

  - 比如该节点是根据 a 属性划分，但是待分类样本 a 属性缺失，怎么办呢？假设 a 属性离散，有 1,2 两种取值，那么就把该样本分配到两个子节点中去，但是权重由 1 变为相应离散值个数占样本的比例。然后计算错误率的时候，注意，不是每个样本都是权重为 1，存在分数。
- 3. *训练完成，给测试集样本分类*，有缺失值怎么办？

  分类时，多数投票：分类时，如果待分类样本有缺失变量，而决策树决策过程中没有用到这些变量，则决策过程和没有缺失的数据一样；否则，如果决策要用到缺失变量，决策树也可以在当前节点做多数投票来决定（选择样本数最多的特征值方向）。

★ Boosting & Bagging

（更多理解参考这个 slides【http://wepon.me/files/gbdt.pdf】）

- 对集成(ensemble)的理解；
- Boosting 算法有那几种　bagging 算法有哪几种
- Bagging 和 boosting 的联系和区别
  - boosting 主要关注降低 bias，bagging 主要减小 variance

AdaBoost（分类 or 回归）

- **原理**

  Adaptive Boost，通过**改变训练样本的权重**（概率分布 or 权值分布，提高前一轮误分类样本权值），学习多个分类器，并将这些分类器线性组合，提高分类性能。（也可以用来做回归）
- **公式推导（根据代码理解）**

  AdaBoost 是**加法算法**的特例，学习算法是**前向分步算法**，损失函数是**指数损失**，二分类算法。

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$.
- Update, for $i = 1, \ldots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

算法 8.2（前向分步算法）

输入：训练数据集 $T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$；损失函数 $L(y, f(x))$；基函数集 $\{b(x; \gamma)\}$；

输出：加法模型 $f(x)$.

（1）初始化 $f_0(x) = 0$

（2）对 $m = 1, 2, \cdots, M$

（a）极小化损失函数

$$(\beta_m, \gamma_m) = \arg\min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) \tag{8.16}$$

得到参数 β_m，γ_m

（b）更新

$$f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m) \tag{8.17}$$

（3）得到加法模型

$$f(x) = f_M(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m) \tag{8.18}$$

- **Loss（指数损失 -> 希望 yi 与 F(xi)符号相同）**

AdaBoost and Exponential Loss

- so AdaBoost is greedy procedure for minimizing exponential loss

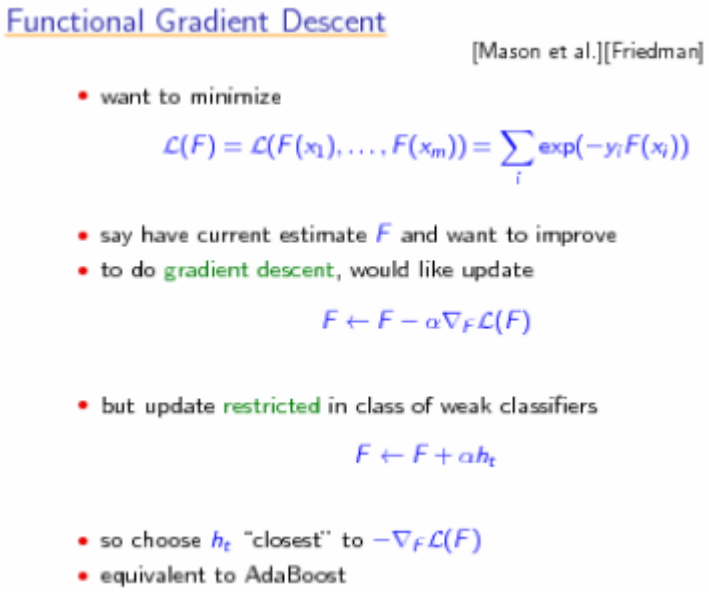
$$\prod_t Z_t = \frac{1}{m} \sum_i \exp(-y_i F(x_i))$$

where

$$F(x) = \sum_t \alpha_t h_t(x)$$

- why exponential loss?
  - intuitively, strongly favors $F(x_i)$ to have same sign as y_i
  - upper bound on training error
    - smooth and convex (but very loose)
- how does AdaBoost minimize it?

- **优化&trick（“近似”梯度下降 & learning rate 正则化）**
  - F 的更新是被限制在弱分类器上的，**实际是通过选择弱分类器来“近似”梯度下降**



    - **正则化**：**学习率**（learning rate）缩减（shrinkage）了每棵树的贡献

- **适用性&优缺点**
    - **优点：低泛化误差，不易过拟合，**分类准确率较高；容易实现，没有太多参数可以调；
    - **缺点**：**对异常样本敏感**(异常样本在迭代中可能会获得较高的权重)，容易受到噪声干扰；**效果依赖弱分类器选择**；训练时间长
- **比较**
- AdaBoost 为啥不容易过拟合？
    - adaboost 模型，比起构成它的基础分类器来说，更容易过拟合。**但当基础分类器选择的非常简单时，模型在实际应用中很难发生过拟合**（这就是为什么基础分类器经常选用树桩的原因），而且在比较好分割的数据集合上，adaboost 的迭代还有**增加 Margin** 的效果。而基础分类器有些复杂的时候，模型在复杂的数据分布上是非常容易发生过拟合的。

      【https://www.zhihu.com/question/41047671】

GBDT（分类 or 回归树）

- **原理（回归树是学习残差，而分类树是做样本扰动，注意这个关键区别）**
    - GBDT 也是迭代，使用了前向分布算法，但是弱分类器限定了**只能用 CART**
    - Gradient Boost **与传统的 Boost 的区别**是，每一次的计算是为了减少上一次的**残差(residual)**，每个**新模型的建立是为了使得之前模型的残差往梯度方向减少**，与传统 Boost 对正确、错误的样本进行加权有着很大的区别。
    - 提升树算法采用前向分步算法。针对不同问题的提升树算法，其**主要区别在于使用的损失函数**不同。包括用**指数损失**的**分类问题**，用**平方损失**的**回归问题**。
    - **GBDT 解决分类问题：**

      GBDT 的分类算法从思想上和 GBDT 的回归算法没有区别，但是由于样本输出不是连续的值，而是离散的类别，导致我们无法直接从输出类别去拟合类别输出的误差。 为了解决这个问题，主要有两个方法，一个是用指数损失函数，此时 GBDT 退化为 Adaboost 算法。另一种方法

是用类似于逻辑回归的对数似然损失函数的方法。也就是说，我们用的是类别的预测概率值和真实概率值的差来拟合损失。

用指数损失（退化为 AdaBoost）；用 Logloss（类似 LR 的对数似然损失）

Name	Loss	Derivative	f^*	Algorithm
Squared error	$\frac{1}{2}(y_i - f(\mathbf{x}_i))^2$	$y_i - f(\mathbf{x}_i)$	$\mathbb{E}[y\|\mathbf{x}_i]$	L2Boosting
Absolute error	$\|y_i - f(\mathbf{x}_i)\|$	$\text{sgn}(y_i - f(\mathbf{x}_i))$	$\text{median}(y\|\mathbf{x}_i)$	Gradient boosting
Exponential loss	$\exp(-\tilde{y}_i f(\mathbf{x}_i))$	$-\tilde{y}_i \exp(-\tilde{y}_i f(\mathbf{x}_i))$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	AdaBoost
Logloss	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	LogitBoost

当损失函数时平方损失和指数损失函数时，每一步的优化很简单，如**平方损失函数学习残差回归树**。但对于一般的损失函数，往往每一步优化没那么容易。针对这一问题，Freidman 提出了梯度提升算法：利用**最速下降的近似方法**，即利用**损失函数的负梯度**在当前模型的值，作为**回归问题中**提升树算法的**残差的近似值**，拟合一个回归树。（私以为，与其说负梯度作为残差的近似值，不如说残差是负梯度的一种特例）

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。比如 A 的真实年龄是 18 岁，但第一棵树的预测年龄是 12 岁，差了 6 岁，即残差为 6 岁。那么在第二棵树里我们把 A 的年龄设为 6 岁去学习，如果第二棵树真的能把 A 分到 6 岁的叶子节点，那累加两棵树的结论就是 A 的真实年龄；如果第二棵树的结论是 5 岁，则 A 仍然存在 1 岁的残差，第三棵树里 A 的年龄就变成 1 岁，继续学。这就是 Gradient Boosting 在 GBDT 中的意义。

每一次的计算是为了减少上一次的残差(residual)，而为了消除残差，我们可以在残差减少的梯度方向上建立一个新的模型。所以说，**在 Gradient Boost 中，每个新的模型的建立是为了使得之前模型的残差往梯度方向减少**（当然也可以变向理解成每个新建的模型都给予上一模型的误差给了更多的关注），**与传统 Boost 对正确、错误的样本进行直接加权还是有区别的。**

算法 8.3（回归问题的提升树算法）

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，$x_i \in \mathcal{X} \subseteq \mathbf{R}^n$，$y_i \in \mathcal{Y} \subseteq \mathbf{R}$；

输出：提升树 $f_M(x)$.

（1）初始化 $f_0(x)=0$

（2）对 $m=1,2,\cdots,M$

（a）按式 (8.27) 计算残差

注意这里是把残差作为下一步的拟合项，这是与提升方法Adaboost最关键的区别！

$$r_{mi}=y_i-f_{m-1}(x_i),\quad i=1,2,\cdots,N$$

（b）拟合残差 r_{mi} 学习一个回归树，得到 $T(x;\Theta_m)$

（c）更新 $f_m(x)=f_{m-1}(x)+T(x;\Theta_m)$

算法 8.4（梯度提升算法）

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，$x_i\in\mathcal{X}\subseteq\mathbf{R}^n$，$y_i\in\mathcal{Y}\subseteq\mathbf{R}$；损失函数 $L(y,f(x))$；

输出：回归树 $\hat{f}(x)$.

（1）初始化

$$f_0(x)=\arg\min_c\sum_{i=1}^{N}L(y_i,c)$$

（2）对 $m=1,2,\cdots,M$

（a）对 $i=1,2,\cdots,N$，计算

$$r_{mi}=-\left[\frac{\partial L(y_i,f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

（b）对 r_{mi} 拟合一个回归树，得到第 m 棵树的叶结点区域 R_{mj}，$j=1,2,\cdots,J$

（c）对 $j=1,2,\cdots,J$，计算

$$c_{mj}=\arg\min_c\sum_{x_i\in R_{mj}}L(y_i,f_{m-1}(x_i)+c)$$

（d）更新 $f_m(x)=f_{m-1}(x)+\sum_{j=1}^{J}c_{mj}I(x\in R_{mj})$

（3）得到回归树

$$\hat{f}(x)=f_M(x)=\sum_{m=1}^{M}\sum_{j=1}^{J}c_{mj}I(x\in R_{mj})$$ ■

算法第 1 步初始化，估计使损失函数极小化的常数值，它是只有一个根结点的树. 第 2 (a) 步计算损失函数的负梯度在当前模型的值，将它作为残差的估计. 对于平方损失函数，它就是通常所说的残差；对于一般损失函数，它就是残差的近似值. 第 2 (b) 步估计回归树叶结点区域，以拟合残差的近似值. 第 2 (c) 步利用线性搜索估计叶结点区域的值，使损失函数极小化. 第 2 (d) 步更新回归树. 第 3 步得到输出的最终模型 $\hat{f}(x)$.

- **公式推导（背下来）**
  - **loss（基于什么？）**
  - **优化&trick**
- **适用性&优缺点**
  - GBDT 是一个加性回归模型，通过 boosting 迭代的构造一组弱学习器，相对 LR 的优势如**不需要做特征的归一化**，**自动进行特征选择**，模型可解释性较好，**可以适应多种损失函数**如 SquareLoss，LogLoss 等等。但作为非线性模型，其相对线性模型的缺点也是显然的：boosting 是个串行的过程，**不能并行化**，计算复杂度较高，同时其**不太适合高维稀疏特征，通常采用稠密的数值特征**
- **比较**
  - GBDT 分裂节点的选择？
  - gbdt 处理回归问题和分类问题的时候分别是怎么解决的（loss function 不一样）
  - Gbdt 中为什么有时候用梯度来代替残差计算

XGBoost

- **原理**【http://blog.csdn.net/sb19931201/article/details/52557382】

  侧重于 Gradient Boosting 方面，给出了 GBDT，GBRT，GBM 具体实现。

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

一阶　　二阶

- Use the statistics to greedily grow a tree $f_t(x)$

$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

贪心算法寻找切分点，生产每一轮新的树

- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
  - Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$

称之为：缩减因子
同样为了避免过拟合

  - ϵ is called step-size or shrinkage, usually set around 0.1
  - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting

- **公式推导（背下来）**
  - **loss（基于什么？）**
    - Xgboost 目标函数？其中正则项 w 表示什么（叶子的权重），
    - Xgboost 里面的 lambda rank 的 loss 函数是什么（叶节点 score 的 L2 正则项的系数）

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \qquad (2)$$
$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2$$

T 为叶子节点的数量，**W 为叶子的权重**。

Y 帽子 为预测值，Y 为目标值。(gamma ,lambda 为参数)

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

$$\text{where } g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \text{ and } h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

  - **优化&trick**
    - Xgboost 建多高合适（不用太高，一般 6~9）
    - Xgboost 中是怎么处理缺失值的？哪些模型对缺失值敏感？哪些不敏感？
      - 对于特征的值有缺失的样本，xgboost 可以自动学习出它的分裂方向 — **稀疏感知算法**（Sparsity-aware Split Finding）。xgboost 把缺失值当做稀疏矩阵来对待，缺失值数据会被**分到左子树和右子树分别计算损失，选择较优的那一个**。如果训练中没有数据缺失，预测时出现了数据缺失，那么默认被分类到右子树。
      - **树模型**对于缺失值的敏感度较低；涉及到**距离度量**的比较敏感（KNN SVM）
    - Xgboost 在什么地方做的剪枝，怎么做的？（根据 loss 收益，递归地向上回缩）
    - XGBoost 的特征重要性是如何得到的？

        - 某个特征的重要性（feature score），等于它被选中为树节点分裂特征的次数的和，比如特征 A 在第一次迭代中（即第一棵树）被选中了 1 次去分裂树节点，在第二次迭代被选中 2 次…..那么最终特征 A 的 feature score 就是 1+2+….
    - xgboost 如何分布式？特征分布式和数据分布式？ 各有什么存在的问题？

      查找最佳分裂点，有两种并行方式
        - 特征分布式（特征间并行方式）

          每个线程处理一维特征，遍历数据累计统计量(grad/hess)得到最佳分裂点
        - 数据分布式（特征内并行方式）

          在每个线程里汇总各个线程内分配到的数据样本的统计量(grad/hess);

          每个线程输出对应样本整体的统计量，得到一个线程级别统计量数组

          在这个组内进行枚举选出最佳分裂点，进一步定位到对应线程的最优分割点
- **适用性&优缺点**
    - Xgboost 相对于 gbdt，可能有以下优势：
        - **对损失函数做了改进（二阶泰勒展开，牛顿提升）**：公式推导里用到了二阶导数信息（二阶泰勒展开），而普通的 GBDT 只用到一阶。（GBDT 用牛顿法也是二阶信息）
        - 正则化
            - 可用**列抽样（和行抽样）**，防止过拟合，借鉴了 Random Forest
            - Xgboost 在损失函数里加入了**正则项**（叶节点总数 & 叶节点 score 的 L2 正则项）
            - **Shrinkage(缩减)**，系数 eta，用于更新叶子节点权重时，乘以该系数，避免步长过大。通过这种方式来减小每棵树的影响力，给后面的树提供空间去优化模型。
        - 并行化
            - 数据事先排序并以 **block** 的形式存储，利于并行计算
            - 支持**分布式计算**可以运行在 MPI 和 yarn 上实现了**一种分裂节点寻找的近似算法**，用于加速和减小内存消耗
        - **稀疏感知算法：**对于特征的值有缺失的样本，xgboost 可以自动学习出它的分裂方向
        - xgboost **支持线性分类器**，传统 GBDT 以 CART 作为基分类器
- **比较**
    - GBDT 和 xgboost 的区别（见上）
    - lightgbm 和 xgboost 有什么区别？他们的 loss 一样么？ 算法层面有什么区别？
        - lightgbm 改进的地方主要是特征选择方法上，使用的是直方图的方式。直方图算法的基本思想是先把连续的浮点特征值离散化成 k 个整数，同时构造一个宽度为 k 的直方图。在遍历数据的时候，根据离散化后的值作为索引在直方图中累积统计量，当遍历一次数据后，直方图累积了需要的统计量，然后根据直方图的离散值，遍历寻找最优的分割点。
- xgboost 使用经验总结
    - 多类别分类时，类别需要从 0 开始编码
    - 类别特征必须编码，因为 xgboost 把特征默认都当成数值型的

    - 训练的时候，为了结果可复现，记得设置随机数种子。
    - XGBoost 的特征重要性（feature score）是如何得到的？某个特征的重要性，等于它被选中为树节点分裂特征的次数的和。比如特征 A 在第一次迭代中（即第一棵树）被选中了 1 次去分裂树节点，在第二次迭代被选中 2 次…..那么最终特征 A 的 feature score 就是 1+2+….

Random Forest

- **原理**
    - 一般 Bagging 特点：
        - 1. 从 N 样本中**有放回的采样** N 个样本（约出现百分之六十的样本）
        - 2. 对这 N 个样本在全属性上建立分类器（默认是 Decision Tree）
        - 3. 重复上面的步骤，建立 m 个分类器
        - 4. 预测的时候使用投票的方法得到结果
    - **Bagging（默认仅样本扰动）**
        - Bagging 全称 bootstrap aggregating。这种方法，把个体预测器当做黑盒子处理，不进行进一步修改，个体预测器可以是任何机器学习算法。它对训练数据集进行 bootstrap sampling（**N 次有放回有重复采样，约 2/3 不重复**），并使用取样后的数据子集，训练每一个个体预测器。在预测时，每一个预测器都会做出预测，整体结果则是每个预测的**投票 or 平均。**
    - **RF（样本扰动+属性扰动）**
        - 随机森林在以决策树为基学习器构建集成的基础上，进一步在决策树的训练过程中引入了随机属性选择。具体来讲，传统决策树在选择划分属性时， 在当前节点的属性集合（假设有个属性）中选择一个最优属性；而**在随机森林中，对基决策树的每个节点，先从该节点的属性集合中随机选择一个包含个属性的子集**，然后再从这个子集中选择一个最优属性用于划分。 这里的参数控制了随机性的引入程度。
- **适用性&优缺点**
- **比较**
    - **Bagging 主要关注降低 variance。**一般而言，**bagging 需要使用 variance 高、比较复杂的强个体模型。**具体到常用的决策树上，就体现在**单个决策树的层数更深。**
    - **boosting 需要使用 bias 高、比较简单的弱个体模型。**
    - 说说 RF 的泛化能力（略）
    - 对于特征值缺失的情况，用 LR 和 RF 来预测缺失值，哪种方法更好（RF）
    - RF 在有很多树的情况下会不会过拟合（不容易过拟合，由于两个随机性）
    - Adaboost 跟随机森林的区别（略）
    - GBDT 和 RF 的区别，应用时的特点（略）

=