### 数据预处理

**样本采样**

* 当模型不能使用全部的数据来训练时，需要对数据进行采样，设定一定的采样率
* 随机采样，固定比例采样等
* 根据需要设定样本权重

**样本过滤**

* 结合业务情况进行数据的过滤，例如去除crawler抓取，spam，作弊等数据
* 异常点检测，采用异常点检测算法对样本进行分析
  + 基于统计的异常点检测
    - 例如极差，四分位数间距，均差，标准差
    - 这种方法适合于挖掘单变量的数值型数据
    - 全距(Range)，又称极差，是用来表示统计资料中的变异量数(measures of variation) ，其最大值与最小值之间的差距
    - 四分位距通常是用来构建箱形图，以及对概率分布的简要图表概述
  + 基于距离的异常点检测
    - 主要通过距离方法来检测异常点，将数据中与大多数点之间距离大于某个阈值的点视为异常点
    - 主要使用的距离度量方法有绝对距离(曼哈顿距离)，欧氏距离和马氏距离等方法
  + 基于密度的异常点检测
    - 考察当前点周围密度，可以发现局部异常点，例如LOF算法

**特征分类**

* low level特征
  + 主要是原始特征，不需要或者需要非常少的人工处理和干预
  + 例如文本特征中的词向量特征，图像特征中的像素点，用户id，商品id等
  + 一般维度比较高，不能用过于复杂的模型
  + 比较针对性，覆盖面小
  + 高频样本的预测值主要受low level特征影响
* high level特征
  + 经过较复杂的处理，结合部分业务逻辑或者规则、模型得到的特征
  + 例如人工打分，模型打分等特征
  + 可以用于较复杂的非线性模型
  + 长尾样本的预测值主要受high level特征影响
* 稳定特征（非实时特征）
  + 变化频率(更新频率)较少的特征
  + 例如评价平均分，团购单价格等，在较长的时间段内都不会发生变化
  + 可以建入索引，较长时间更新一次，如果做缓存的话，缓存的时间可以较长
* 动态特征（实时特征）
  + 更新变化比较频繁的特征，有些甚至是实时计算得到的特征
  + 例如距离特征，2小时销量等特征
  + 需要实时计算或准时更新数据，如果做缓存的话，缓存过期时间需要设置的较短
* 二值特征
  + 主要是0/1特征，即特征只取两种值：0或者1
  + 例如用户id特征：目前的id是否是某个特定的id，词向量特征：某个特定的词是否在文章中出现
* 连续值特征
  + 取值为有理数的特征，特征取值个数不定
  + 例如距离特征，特征取值为是0~正无穷
  + 连续值处理为二值特征：先将连续值离散化（见*特征处理*），再将离散化后的特征切分为N个二元特征，每个特征代表是否在这个区间内
* 枚举值特征
  + 主要是特征有固定个数个可能值
  + 例如今天周几，只有7个可能值：周1，周2，…，周日
  + 枚举特征处理为二值特征：将枚举特征映射为多个特征，每个特征对应一个特定枚举值例如今天周几，可以把它转换成7个二元特征：今天是否是周一，今天是否是周二，…，今天是否是周日等

**特征处理**

* 归一化 normalization
  + 不同的特征有不同的取值范围，特征的取值范围在有些算法中会对最终的结果产生较大影响
  + 例如二元特征的取值范围为[0，1]，而距离特征取值可能是[0，正无穷)
  + 在实际使用中会对距离进行截断，例如[0，3000000]，但是这两个特征由于取值范围不一致导致了模型可能会更偏向于取值范围较大的特征，为了平衡需要对特征进行归一化处理到［0，1］区间
  + 方法
    - 线性的映射，例如用全局的最大最小值归一化（取值区间的最大最小值）
    - 非线性函数的映射，例如log函数等
    - 分维度归一化，使用的是局部最大最小值（数据集的最大最小值）
    - 排序归一化，不管原来的特征取值是什么样的，将特征按大小排序，根据特征所对应的序给予一个新的值
* 离散化 discretization
  + 连续值的取值空间可能是无穷的，为了便于表示和在模型中处理，需要对连续值特征进行离散化处理
  + 等值划分
    - 将特征按照值域进行均分，每一段内的取值等同处理
    - 例如某个特征的取值范围为[0，10]，可以将其划分为10段，[0，1)，[1，2)，…，[9，10)。
  + 等量划分
    - 根据样本总数进行均分，每段等量个样本划分为1段
    - 例如距离特征，取值范围［0，3000000］，现在需要切分成10段，如果按照等比例划分的话，绝大部分样本可能都在第1段中。使用等量划分就会避免这种问题，最终可能的切分是[0，100)，[100，300)，[300，500)，..，[10000，3000000]，前面的区间划分比较密，后面的比较稀疏
* 缺失值处理 missing value
  + 有些特征可能因为无法采样或者没有观测值而缺失
  + 例如距离特征，用户可能禁止获取地理位置或者获取地理位置失败，此时需要对这些特征做特殊处理，赋予一个缺省值
  + 例如单独表示，众数，中位数，平均值等

**特征维度处理**

* VC-dimension
  + VC维越高，可容许的模型复杂度越高
  + 在低维不可分的数据，映射到高维是可分（升维）
* 高维的问题
  + 模型容易过拟合，复杂的模型表现力下降
  + 相互独立的特征维数越高，在模型不变的情况下，在test集上达到相同的效果表现所需要的训练样本的数目越大
  + 训练、测试以及存储的开销都会增大
  + 在某些模型例如基于距离计算的模型KMeans，KNN等模型，在进行距离计算时，维度过高会影响精度和性能
  + 可视化分析困难（维度灾难）
* 降维的方法
  + PCA
    - 通过协方差矩阵的特征值分解能够得到数据的主成分
    - 以二维特征为例，两个特征之间可能存在线性关系（例如运动的时速和秒速度），这样就造成了第二维信息是冗余的
    - PCA的目标是发现这种特征之间的线性关系，并去除
  + LDA(? todo)
    - 考虑label，降维后的数据点尽可能地容易被区分

**特征选择**

* 目的
  + 剔除不相关(irrelevant)或冗余(redundant)的特征，从而减少特征个数，提高模型精确度，减少运行时间
  + 选取出真正相关的特征简化模型，协助理解数据产生的过程
  + 主要分为产生过程，评估过程，停止条件和验证过程
* 产生过程
  + 完全搜索(Complete)
    - BFS, 广度优先遍历特征子空间，枚举所有组合，穷举搜索，实用性不高
    - 分支限界搜索(Branch and Bound)：穷举基础上加入分支限界(剪枝)
    - 定向搜索 (Beam Search)
    - 最优优先搜索 (Best First Search)
  + 启发式搜索(Heuristic)
    - 序列前向选择(SFS，Sequential Forward Selection)：从空集开始，每次加入一个选最优
    - 序列后向选择(SBS，Sequential Backward Selection)：从全集开始，每次减少一个选最优
    - 增L去R选择算法 (LRS，Plus-L Minus-R Selection)：从空集开始，每次加入L个，减去R个，选最优(L>R),或者从全集开始，每次减去R个，增加L个，选最优(R>L)
  + 双向搜索(BDS，Bidirectional Search)
  + 序列浮动选择(Sequential Floating Selection)
  + 随机搜索
    - 缺点：依赖随机因素，有实验结果难重现
    - 随机产生序列选择算法(RGSS，Random Generation plus Sequential Selection)：随机产生一个特征子集，然后在该子集上执行SFS与SBS算法
    - 模拟退火算法(SA，Simulated Annealing)：以一定的概率来接受一个比当前解要差的解，而且这个概率随着时间推移逐渐降低
    - 遗传算法(GA，Genetic Algorithm)：通过交叉、突变等操作繁殖出下一代特征子集，并且评分越高的特征子集被选中参加繁殖的概率越高
  + 有效性分析
    - 与模型相关特征权重：使用所有的特征数据训练出来模型，看在模型中各个特征的权重，由于需要训练出模型，模型相关的权重与此次学习所用的模型比较相关，不同的模型有不同的模型权重衡量方法
      * 例如线性模型中，特征的权重系数等
    - 与模型无关特征权重：主要分析特征与label的相关性，这样的分析是与这次学习所使用的模型无关的
      * 例如(1)Cross Entropy，(2)Information Gain，(3)Odds ratio，(4)互信息，(5)KL散度等

**特征监控(实战)**

* 对于重要的特征进行监控与有效性分析，了解模型所用的特征是否存在问题，当某个特别重要的特征出问题时，需要做好备案，防止灾难性结果
* 在发现特征出现异常时，需要及时采取措施，对服务进行降级处理，并联系特征数据的提供方尽快修复

### ML性能评价指标

* 分类
  + 混淆矩阵(confusion matrix)，对于二分类问题的评价指标
    - True Positive（真正，TP）：将正类预测为正类数
    - True Negative（真负，TN）：将负类预测为负类数
    - False Positive（假正，FP）：将负类预测为正类数
    - False Negative（假负，FN）：将正类预测为负类数
  + 精确率(precision,预测为正的样本中有多少是对的) =
    - 不同于准确率(accuracy,有多少预测对的) = ,在非平衡数据的情况下，准确率这个评价指标有很大的缺陷
  + 召回率(recall,样本中的正例有多少被预测正确) =
  + F1(precison和recall的调和均值)：
  + ROC曲线：越接近左上角，分类器的性能越好
    - 横坐标, 代表预测的正类中实际正实例占所有正实例的比例：直观上代表能将正例分对的概率
    - 纵坐标, 代表预测的正类中实际负实例占所有负实例的比例：直观上代表将负类错分为正例的概率
    - 优势：当测试集中的正负样本的分布变化的时候，ROC曲线能够保持不变。在实际的数据集中经常会出现非平衡数据的现象，即负样本比正样本多很多（或者相反），而且测试数据中的正负样本的分布也可能随着时间变化，而Precision-Recall曲线则对非平衡数据敏感
  + AUC(area under curve),作为数值直观地评价分类器的好坏，值越大越好
    - AUC=1，完美分类器，采用这个预测模型时，存在至少一个阈值能得出完美预测，绝大多数场合，不存在完美的分类器
    - AUC=[0.5,1]，优于随机猜测，这个分类器妥善设定阈值的话，有预测价值
    - AUC=0.5，跟随机猜测一样（如丢硬币），模型没有预测价值
    - AUC<0.5，比随机猜测还差；但只要总是反预测而行，就优于随机猜测
* 回归
  + 平均绝对误差(Mean Absolute Error, L1-norm loss)
  + 平均平方误差(Mean Squared Error, L2-norm loss)

### 非平衡数据处理

**概念**

* 举例：广告点击预测（点击转化率一般都很小）、商品推荐（推荐的商品被购买的比例很低）、信用卡欺诈检测
* 对于不平衡数据集，一般的分类算法都倾向于将样本划分到多数类，体现在模型整体的准确率很高
* 但对于极不均衡的分类问题，比如仅有1%的人是坏人，99%的人是好人，最简单的分类模型就是将所有人都划分为好人，模型都能得到99%的准确率，这样的模型并没有提供任何的信息
* 在类别不平衡的情况下，对模型使用F值或者AUC值更好
* 方法
  + 改变数据分布，从数据层面使得类别更为平衡
  + 改变分类算法，在传统分类算法的基础上对不同类别采取不同的加权方式，使得模型更看重少数类
* 本部分对数据层面的方法做一个介绍，改变数据分布的方法主要是*重采样*
  + 过采样：增加少数类样本的数量
  + 欠采样：减少多数类样本的数量
  + 综合采样：将过采样和欠采样结合

**过采样**

* 随机过采样
  + 增加少数类样本数量，可以事先设置多数类与少数类最终的数量比例，在保留多数类样本不变的情况下，根据比例随机复制少数类样本
  + 为保证所有的少数类样本信息都会被包含，可以先*完全*复制一份全量的少数类样本，再随机复制少数样本使得满足数量比例
  + 注意：重复样本过多，容易造成分类器的过拟合
  + 算法
    - 首先在少数类Smin集合中随机选中一些少数类样本
    - 然后通过复制所选样本生成样本集合E
    - 将它们添加到Smin中来扩大原始数据集从而得到新的少数类集合Smin−new
    - Smin 中总样本数增加了|E|个新样本，且对分布均衡度进行了相应的调整
* SMOTE算法(Synthetic Minority Oversampling Technique)
  + 由于随机过采样简单复制样本来增加少数类样本，容易产生模型过拟合的问题，即使模型学习到的信息过于特别（Specific）而不够泛化(General)
  + SMOTE的思想是利用特征空间中现存少数类样本之间的*相似性*来建立人工数据，对于子集Smin中每一个样本xi使用*K-近邻*法
  + K-近邻被定义为考虑Smin中的K个元素本身与xi的欧氏距离，在n维特征空间X中表现为最小幅度值的样本
  + 由于不是简单地复制少数类样本，可以在一定程度上避免过拟合
  + 但对每个少数类样本都生成新样本，容易出现生成样本重叠（overlapping）
  + 算法
    - 对于少数类中的每一个样本(xi)，以欧氏距离为标准计算它到少数类样本集Smin中所有样本的距离，得到K近邻
    - 根据样本不平衡比例设置一个采样比例以确定采样倍率N，对于每一个少数类样本xi，从其K近邻中随机选择若干个样本
    - 假设选择的近邻为x’, 对于每一个随机选出的近邻x’，分别与原样本按照如下的公式构建新的样本:xnew=x+rand(0,1)×(x’−x)
* Borderline-SMOTE算法
  + SMOTE算法对所有的少数类样本都一视同仁，但实际建模过程中处于边界位置的样本更容易被错分，因此利用边界位置的样本信息产生新样本可以给模型带来更大的提升，Borderline-SMOTE算法在SMOTE基础上结合了边界信息算法
  + 首先，对于每个xi⊂Smin确定一系列K-近邻样本集，称该数据集为Si−kNN
  + 然后，对每个样本xi，判断出Si−kNN中属于多数类样本的个数，即：#=|Si−kNN∩Smaj|
  + 最后，选择满足下面不等式的xi:k/2<#<k，将其加入危险集DANGER
  + 对危险集中的每一个样本点（最容易被错分的样本），采用SMOTE算法生成新的少数类样本
  + 也就是说少数样本的kNN里一半以上是多数样本才生成新样本

**欠采样**

* 随机欠采样
  + 随机剔除多数类样本，可以事先设置多数类与少数类最终的数量比例，在保留少数类样本不变的情况下，根据比例随机选择多数类样本
  + 算法
    - 首先从Smaj中随机选取一些多数类样本E
    - 将这些样本从Smaj中移除，就有|Smaj−new|=|Smaj−|E|
  + 优点：操作简单，只依赖于样本分布，不依赖任何距离信息，属于非启发式方法
  + 缺点：会丢失一部分多数类样本的信息，无法充分利用已有信息
* Tomek Links方法
  + 简略：删除相反类最近邻样本之间的多数类
  + 给一个样本对(xi,xj),xi属于多数类样本，xj属于少数类，d(xi,xj)为他们之间的距离
  + 如果不存在任何样本xk，d(xi,xk)<d(xi,xj),那么样本对(xi,xj)被称为Tomek Links
  + 如果两个样本来自Tomek Links，那么他们中的一个样本要么是噪声要么它们都在两类的边界上
  + 所以Tomek Links一般有两种用途
    - 在欠采样中：将Tomek Links中属于是多数类的样本剔除
    - 在数据清洗中，将Tomek Links中的两个样本都剔除
* NearMiss方法
  + 利用距离远近剔除多数类样本的一类方法，也是借助KNN（k=3）
  + NearMiss-1：在多数类样本中选择与最近的3个少数类样本平均距离最小的样本
  + NearMiss-2：在多数类样本中选择与最远的3个少数类样本平均距离最小的样本
  + NearMiss-3：对于每个少数类样本，选择离它最近的给定数量的多数类样本
  + NearMiss-1考虑的是与最近的3个少数类样本的平均距离，是局部的；NearMiss-2考虑的是与最远的3个少数类样本的平均距离，是全局的
  + NearMiss-1方法得到的多数类样本分布也是”不均衡“的，它倾向于在比较集中的少数类附近找到更多的多数类样本，而在孤立的（或者说是离群的）少数类附近找到更少的多数类样本，原因是NearMiss-1方法考虑的局部性质和平均距离
  + NearMiss-3方法则会使得每一个少数类样本附近都有足够多的多数类样本，显然这会使得模型的精确度高、召回率低
  + 实验结果表明得到NearMiss-2的不均衡分类性能最优

**Informed Understanding**

* Informed欠抽样算法可以解决传统随机欠采样造成的*数据信息丢失*问题，且表现出较好的不均衡数据分类性能
* 其中有一些集成（ensemble）的想法，主要有EasyEnsemble和BalanceCascade
* EasyEnsemble算法
  + 多次随机欠抽样，尽可能全面地涵盖所有信息
  + 算法特点是利用boosting减小偏差（Adaboost）、bagging减小方差（集成分类器）
  + 实际应用的时候也可以尝试选用不同的分类器来提高分类的效果
  + 算法
    - 对于多数类样本Smaj，通过n次有放回抽样生成n份子集
    - 少数类样本Smin分别和这n份样本合并训练AdaBoost分类器
    - 得到n个模型，最终的模型采用加权多数表决的方法
    - 加大分类误差率小的弱分类器的权值，使其在表决中起较大的作用
    - 减小分类误差率小的弱分类器的权值，使其在表决中起较小的作用
* BalanceCascade
  + EasyEnsemble算法训练的子过程是独立的，BalanceCascade则是一种级联算法，这种级联的思想在图像识别中用途非常广泛
  + 算法
    - 设定训练层级T，以及每一层级的分类器都要达到的flase positive rate(FPR):f=(|P|/|N|)^((T-1)/2)
    - 对i=1到T训练Adaboost分类器，调整阈值使分类器的FPR为f，将被正确分类的样本剔除
  + 这种级联分类器将若干个强分类器由简单到复杂排列，只有和少数类样本特征比较接近的才有可能输入到后面的分类器，比如边界点，因此能更充分地利用多数类样本的信息，一定程度上解决随机欠采样的信息丢失问题

**综合采样**

* 将欠采样和过采样综合的方法，解决样本类别分布不平衡和过拟合问题
* SMOTE+Tomek Links
  + 利用SMOTE方法生成新的少数类样本，得到扩充后的数据集T
  + 剔除T中的Tomek Links对
  + 普通的SMOTE方法生成的少数类样本是通过*线性插值*得到的，在平衡类别分布的同时也扩张了少数类的样本空间，产生的问题是可能原本属于多数类样本的空间被少数类“入侵”，容易造成模型的过拟合
  + Tomek Links对寻找的是噪声点或者边界点，可以很好地解决“入侵”的问题
* SMOTE+ENN（类似above）
  + 利用SMOTE方法生成新的少数类样本，得到扩充后的数据集T
  + 对T中的每一个样本使用KNN（一般K取3）方法预测，若预测结果与实际类别标签不符，则剔除该样本

### 损失函数

* 用来估量模型的预测值f(x)与真实值Y不一致的程度，是一个非负实数值函数，通常使用L(Y,f(x))来表示
* 损失函数越小，模型越robust
* 模型的结构风险函数包括了经验风险项和正则项，损失函数是经验风险项的核心部分

**类别**

* 对数损失函数（logistic loss）
  + 逻辑回归的损失函数是log损失函数，L(Y,f(x))=-log P(Y|X)
  + 假设样本服从伯努利分布（0-1分布），然后求得满足该分布的似然函数，接着取对数求极值
  + 逻辑回归没有求似然函数的极值，而是把极大化当做是一种思想，进而推导出它的经验风险函数为：最小化负的似然函数（即maxF(y,f(x))—>min−F(y,f(x)))
  + 取对数是为了方便计算极大似然估计，因为在MLE中，直接求导比较困难，所以通常都是先取对数再求导找极值点
  + 损失函数表达的是样本在分类Y的情况下，使概率P(Y|X)达到最大值（换言之，就是利用已知的样本分布，找到最有可能导致这种分布的参数值；或者什么样的参数才能使我们观测到目前这组数据的概率最大）
  + 因为log函数单调递增，所以logP(Y|X)也会达到最大值，加上负号之后等价于最小化L
  + 之所以有人认为逻辑回归是平方损失，是因为在使用梯度下降来求最优解的时候，它的迭代式子与平方损失求导后的式子非常相似，给人一种直观上的错觉
* 平方损失函数（Least Square Loss，最小二乘法，OLS)
  + 最小二乘法是线性回归的一种，OLS将问题转化成了一个凸优化问题
  + L(Y,f(x))=
  + 线性回归假设样本和噪声都服从高斯分布
    - 假设Gaussian Dist的原理是CLT中心极限定理，通过极大似然估计（MLE）可以推导出最小二乘式
    - 最小二乘的基本原则是最优拟合直线应该是使各点到回归直线的距离和最小的直线，即平方和最小
    - OLS是基于我们用的最多的欧几里得距离MSE（原因：）
      * 简单，计算方便
      * 欧氏距离是一种很好的相似性度量标准similarity measurement
      * 在不同的表示域变换后特征性质不变
* 指数损失函数（exp loss，Adaboost）
  + Adaboost实际上是前向分步加法算法的特例，是一个加和模型，损失函数是指数函数，标准形式为L(y,f(x))=exp[-yf(x)]
  + 在Adaboost中，经过m此迭代之后，可以得到
  + 每次迭代时的目的是为了找到最小化下列式子时的参数a和G
  + 给定N个样本的情况下，Adaboost的损失函数为
* Hinge合页损失函数（SVM）
  + SVM原始最优化问题的dual form就是最优化
  + 目标函数的第一项是经验损失函数，也就是hinge loss，“+”相当于Relu激活函数
  + 目标函数的第二项是系数为λ的w的L2范数正则化项
  + 相比感知机当样本点被正确分类时，损失是0；Hinge不仅要分类正确，而且确信度足够高时损失才是0，对学习有更高的要求
* 对比logistic和hinge
  + 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只考虑support vectors，也就是和分类最相关的少数点学习分类器
  + 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，根本目的是一样的
  + svm考虑局部（支持向量），而logistic回归考虑全局
  + LR对异常值敏感，SVM对异常值不敏感
  + 在训练集较小时，SVM较适用，而LR需要较多的样本
  + LR模型找到的那个超平面，是尽量让所有点都远离他；而SVM寻找的那个超平面，是只让最靠近中间分割线的那些点尽量远离，即只用到支持向量的样本
  + 对非线性问题的处理方式不同，LR主要靠特征构造，必须组合交叉特征，特征离散化。SVM也可以这样，还可以通过kernel
  + svm更多的属于非参数模型，LR是参数模型，本质不同（todo more）
  + 如何选择
    - 如果Feature数量很大，跟样本数量差不多，选用LR或Linear Kernel的SVM
    - 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
    - 如果Feature的数量比较小，而样本数量很多，需要手工添加一些feature变成第一种情况

### 过拟合和欠拟合

* Bias-Variance Tradeoff
  + 是针对Generalization泛化性来说的
  + 训练数据集的Loss与一般化的数据集（预测数据集）的Loss之间的差异叫做Generalization error
  + 可以细分为Random Error、Bias和Variance三个部分
  + 随机误差
    - 数据本身的噪声带来的，这种误差不可避免
    - 其次如果我们能够获得所有可能的数据集合，并在这个数据集合上将Loss最小化，这样学习到的模型就可以称之为“真实模型”，但我们是无论如何都不能获得并训练所有可能的数据的，所以真实模型一定存在，但无法获得，我们的最终目标就是去学习一个模型使其更加接近这个真实模型
  + Bias
    - 描述对于测试数据集，“用所有可能的训练数据集训练出的所有模型的输出预测结果的期望”与“真实模型”的输出值（样本真实结果）之间的差异
    - 简单讲就是在样本上拟合的好不好
    - 要想在bias上表现好，low bias，就是复杂化模型，增加模型的参数，但容易过拟合overfitting
  + Variance
    - “不同的训练数据集训练出的模型”的输出值之间的差异
  + Bias与Variance不能兼得的根本原因
    - 我们总是希望试图用有限训练样本去估计无限的真实数据
    - 当我们更加相信这些数据的真实性，而忽视对模型的先验知识，就会尽量保证模型在*训练样本*上的准确度，这样可以减少模型的Bias
    - 但是，这样学习到的模型，很可能会失去一定的泛化能力，从而造成过拟合，降低模型在*真实数据*上的表现，增加模型的不确定性
    - 相反，如果更加相信我们对于模型的先验知识，在学习模型的过程中对模型增加更多的限制，就可以降低模型的variance，提高模型的稳定性，但也会使模型的Bias增大
    - Bias与Variance的trade-off是机器学习的基石之一，可以在各种机器模型中发现它的影子
  + 数学推论(f’(X)-fitted, f(X)-true, ε-Random Error)
    - 均方误差
* 如何Trade Off
  + 最佳平衡点：偏差与方差的加和
  + 模型复杂度小于平衡点，模型的偏差会偏高，模型倾向于欠拟合
  + 模型复杂度大于平衡点，模型的方差会偏高，模型倾向于过拟合
* 过拟合与欠拟合的外在表现
  + 欠拟合(high bias)：训练集误差很高，验证集误差和训练集误差差不多大
  + 过拟合(high variance)：训练集误差较低，验证集误差很高
* 处理欠拟合
  + 增加模型迭代次数
  + 训练复杂度更高的模型：比如在神经网络中增加神经网络层数、在SVM中用非线性SVM（核技术）代替线性SVM
  + 获取更多的特征以供训练使用：特征少，对模型信息的刻画不足够
  + 降低正则化权重：正则化是为了限制模型的灵活度（复杂度）而设定的，降低其权值可以在模型训练中增加模型复杂度
* 处理过拟合
  + 根本办法是降低模型的复杂度
  + 获取更多的数据：训练数据集和验证数据集是随机选取的，它们有不同的特征，以致在验证数据集上误差很高。更多的数据可以减小这种随机性
  + 减少特征数量
  + 增加正则化权重：方差很高时，模型对训练集的拟合很好。实际上，模型很有可能拟合了训练数据集的噪声，拿到验证集上拟合效果不好。可以增加正则化权重，减小模型的复杂度

### 正则化

**经验风险最小化到结构经验最小化**

* 经验风险最小化（empirical risk minimization），min(L(Y,f(X)))
  + 认为经验风险最小的模型是最优的模型，当样本容量足够大的时候，经验风险最小化学习效果良好
  + 比如极大似然估计MLE，当模型是条件概率分布，损失函数是对数损失函数时，经验风险最小化就等价于极大似然估计
  + 但是当样本容量很小时，经验风险最小化学习会过拟合，这就引出了结构风险最小化，它等价于正则化（regularization）
* 结构风险最小化，min(L(Y,f(X))+λJ(f))
  + 认为结构风险最小的模型是最优的模型
  + 其中J(f)为模型的复杂度，模型f越复杂，复杂度J(f)就越大
  + λ≥0 是正则化系数，用以权衡经验风险和模型复杂度
  + 结构风险小的模型往往对训练数据以及未知的测试数据都有较好的预测
  + 比如贝叶斯估计中的最大后验概率估计就是结构风险最小化的一个例子
  + 当模型是条件概率分布、损失函数是对数损失函数、模型复杂度由模型的先验概率表示时，结构风险最小化就等价于最大后验概率估计

**范数与正则项**

* 范数（Norm）
  + 一个函数赋予某个向量空间（或矩阵）中每个向量的长度或大小
  + 对于零向量，另其长度为零
  + 直观的说，向量或矩阵的范数越大，这个向量或矩阵也就越大
  + 有时范数有更为常见的叫法，如绝对值其实便是一维向量空间中实数或复数的范数，而Euclidean距离也是一种范数
  + 范数满足通常意义上长度的三个基本性质
    - 非负性：||x⃗ ||≥0
    - 齐次性：||c⋅x⃗ ||=|c|||x⃗ ||
    - 三角不等式：||x⃗ +y⃗ ||≤||x⃗ ||+||y⃗ ||
  + 我们最关注的是「非负性」。损失函数通常是一个有下确界的函数。而这个性质保证了可以对损失函数做最优化求解。如果要保证目标函数依然可以做最优化求解，那么正则项也必须有一个下界。非负性提供了这样的下界，而且它是一个下确界——由齐次性保证（当 c=0 时）
  + 因此，范数的性质使得它天然地适合作为机器学习的正则项
  + 定义
    - 设p≥1的实数，p−norm定义为
    - 常用的范数有L1,L2,Lp,L∞范数
    - p=1时称之为taxicab Norm，也叫Manhattan Norm
    - 其来源是曼哈顿的出租车司机在四四方方的曼哈顿街道中从一点到另一点所需要走过的距离，也即L1范数，表示某个向量中所有元素绝对值的和
    - p=2时是最为常见的Euclidean norm，也称为Euclidean distance，也即L2范数
    - p=0时，因其不再满足三角不等性，严格的说此时p已不算是范数了，但很多人仍然称之为L0范数。
    - 这三个范数在正则化以及稀疏编码（Sparse Coding）有非常有趣的应用
* LASSO regularizer(L0->L1)
  + 考虑一种普遍的情况，预测目标背后的真是规律，可能只和某几个维度的特征有关；而其它维度的特征，要不然作用非常小，要不然纯粹是噪声
  + 这种情况下，除了这几个维度的特征对应的参数之外，其它维度的参数应该为零
  + 若不然，则当其它维度的特征存在噪音时，模型的行为会发生预期之外的变化，导致过拟合
  + 于是有了避免过拟合的第一个思路：使尽可能多的参数为零
  + 为此最直观地引入L0范数：
  + 如果使得一个参数非零得到的收益（损失函数上的收益）不足 ℓ0；那么增加这样的复杂度得不偿失
  + 优点：可以使模型稀疏化且易于解释，并且在某种意义上实现了「特征选择」
  + 缺点：非连续，非凸，不可求导
  + 因此L0正则项虽好，但是求解这样的最优化问题难以在多项式时间内找到有效解（NP-Hard ）
  + 于是转而考虑L0范数最紧的凸放松（tightest convex relaxation）L1范数：
  + 参数额外更新过程：，多出的项使得wi→0，实现稀疏化sparsity
* Ridge Regularizer(L2)
  + 我们注意到，当多项式模型过拟合时，函数曲线倾向于靠近噪声点
  + 这意味着函数曲线会在噪声点之间来回扭曲跳跃
  + 也就是说，在某些局部，函数曲线的切线斜率会非常高（波动大，函数导数的绝对值大，也就是多项式系数的一个线性加和大）
  + 过拟合的多项式模型，它的参数的绝对值大（至少某几个参数分量的绝对值大）
  + 因此，如果有办法使得这些参数的值，比较稠密均匀地集中在0附近，就能有效地避免过拟合
  + 引入L2正则项之后，相当于衰减（decay）了参数的权重，使参数趋近于0
* L1与L2的区别
  + L1旨在删除不重要的特征项，于是使得参数变得稀疏；L2旨在平衡参数的大小，于是使得参数变得稠密
  + 简便起见，比方说考虑模型只有两个参数w1和w2的情形
  + 有三组等值线，位于同一条等值线上的w1与w2映射到相同的平方损失项、L1范数和L2范数
  + 当(w1,w2)沿等值线法线方向向外扩张，则对应的值增大；反之对应的值减小
  + 对于目标函数Obj(F)来说，要在正则项的等值线与损失函数的等值线中寻找一个交点，使得二者的和最小
  + 对于L1正则项来说，因为L1正则项是一组菱形，这些交点容易落在坐标轴上。因此，另一个参数的值在这个交点上就是0，从而实现了稀疏化
  + 对于L2正则项来说，因为L2正则项是一组圆形，这些交点可能落在整个平面的任意位置，所以它不能实现「稀疏化」。但另一方面，由于(w1,w2)落在圆上，所以它们的值会比较接近。这就是为什么L2正则项可以使得参数在零附近稠密而平滑
* 贝叶斯先验
  + 在数据维度很高的情况下，模型参数很多，模型复杂度高，容易发生过拟合，也就是常说的“small n, large p problem”（一般用n表示数据点个数，p表示变量个数，即数据维度）
  + 这个时候，我们可以对参数w引入先验分布，降低模型复杂度
  + Ridge Regression
    - 可以解决multicolinearity，然而L2 penalty并不能shrink parameters to 0，所以没法做variable selection
    - 对参数w引入了协方差为 α 的零均值高斯先验，等价于L2正则化
    - 实验得到的参数在零附近很密集，实际上ridge regression并不具有产生稀疏解的能力，也就是说参数并不会真出现很多零
    - 假设预测结果与两个特征相关，L2正则倾向于综合两者的影响，给影响大的特征赋予高的权重；而L1正则倾向于选择影响较大的参数，而舍弃掉影响较小的那个
    - 实际应用中L2正则表现往往会优于L1正则，但 L1正则会大大降低计算量
  + LASSO (least absolute shrinkage and selection operator)
    - 针对Ridge Regression没法做variable selection的问题提出，L1 penalty虽然算起来麻烦，没有解析解，但是可以把某些系数shrink到0
    - 对参数w引入了拉普拉斯先验，等价于L1正则化
    - LASSO仍然是一个convex optimization 问题，不具有解析解。它的优良性质是能产生稀疏性，导致w中许多项变成零
  + Elastic Net
    - 在L1和L2penalty之间做权重
    - lasso在解决“small n, large p problem”存在一定缺陷：当n很小时至多只能选出n个变量，不能做group selection
    - L1部分产生稀疏模型，L2部分解除了特征选择数量的限制，鼓励了grouping effect，稳定了L1的正则path
  + todo: MCP，SCAD
* 总结
  + 正则化参数等价于对参数引入先验分布，使得模型复杂度变小（缩小解空间）
  + 对于噪声以及 outliers 的鲁棒性增强（泛化能力）
  + 整个最优化问题从贝叶斯观点来看是一种贝叶斯最大后验估计
  + 其中正则化项对应后验估计中的先验信息，损失函数对应后验估计中的似然函数，两者的乘积即对应贝叶斯最大后验估计的形式

### 梯度下降

**最速下降法**

* 又称为梯度下降法，是无约束最优化领域中最简单的算法
* 理念是其他某些算法的组成部分，或者说是在其他算法中，也有最速下降法的影子
* 一种迭代算法，每一步需要求解目标函数的梯度向量
* 当目标函数是凸函数时，梯度下降法的解是全局最优解
* 一般情况下，其解不保证是全局最优解，梯度下降法的收敛速度也未必是很快的

**牛顿法**

* 推导概念：海森矩阵，牛顿方向
* 是一种具有二次收敛性的算法，基本思想是在现有极小值估计值的附近对f(x)作二阶泰勒展开，进而寻找极小点的下一个估计值
* 优点：对于非二次函数，若函数的二次形性态较强，或迭代点已进入极小点的领域，则其收敛速度也很快
* （原）缺点：原始牛顿法由于迭代中没有步长因子，而是定步长迭代，对于非二次型目标函数，有时会使函数值上升，不能保证函数值稳定地下降，在严重的情况下甚至可能造成迭代点列{Xk}的发散而导致计算失败
  + 为了消除这个弊病，提出了“阻尼牛顿法”，每次迭代需沿迭代方向作一维搜索（line search），寻求最优的步长因子λk
* 牛顿法是梯度下降法的进一步发展，梯度下降法利用目标函数的一阶偏导数信息、以负梯度方向作为搜索方向，只考虑目标函数在迭代点的局部性质；而牛顿法不仅使用目标函数的一阶偏导数，还进一步利用了目标函数的二阶偏导数，这样就考虑了梯度变化的趋势，因而能更全面地确定合适的搜索方向加快收敛，它具二阶收敛速度
* 缺点
  + 对目标函数有较严格的要求，函数必须具有连续的一、二阶偏导数，海森矩阵必须正定
  + 相当复杂，除需要计算梯度以外，还需要计算二阶偏导数矩阵和它的逆矩阵。计算量、存储量均很大，且均以维数N的平方比增加，当N很大时问题更加突出

**拟牛顿法**

* 为克服牛顿法的问题，提出了拟牛顿法，基本思想是不用二阶偏导数而构造出可以近似海森矩阵或者海森矩阵的逆的正定对称阵，在*拟牛顿条件*下优化目标函数
* 对算法中用来计算搜索方向的海森矩阵（或海森矩阵的逆）作了近似计算
* 拟牛顿条件（todo）
* DFP算法
  + 通过迭代的方法，对海森矩阵的逆的正定对称阵做近似
* BFGS算法
  + 求解无约束非线性优化问题最常用的方法之一，有较完善的局部收敛理论
  + 通过迭代的方法，直接逼近海森矩阵
* L-BFGS算法（todo）

### 线性回归

* 线性回归假设特征和结果满足线性关系
* 每个特征变量可以首先映射到一个函数，然后再参与线性计算，这样就可以表达特征与结果之间的非线性关系
* y=ax+…+?x+b 权重a，截距b
* 在一些应用场合中，需要将输入空间x=(1,x1,x2,…,xn)映射到特征空间（feature space）中，然后建模,特征映射相关技术包括特征哈希、特征学习、Kernel等

**目标函数**

* 损失函数（loss function），平方损失函数（squared loss）
* 极大似然估计（根据中心极限定理，认为变量之和服从高斯分布，MLE把高斯分布的条件概率相乘）与损失函数极小化等价

**参数估计**

* 最小二乘法
* （批量，随机）梯度下降法（Batch/Stochastic/GD）
* 训练样本集较大时，一般使用SGD

**逻辑回归**

* logistic dist，对数几率（oods）,sigmoid
* 实际场景中，我们用其预测值作为事件发生的概率，而非绝对的分类问题
* MLE把伯努利分布（Bernoulli dist）的结果相乘

**参数估计**

* MLE就是最小化交叉熵误差（Cross Entropy Error）
* Batch GD，共轭梯度法（Conjugate Gradient），拟牛顿法（LBFGS），ADMM分布学习算法
* 分类边界：0.5

**其他**

* 线性不可分的数据，可以通过特征变换的方式把低维空间转换到高维空间（kernel trick），线性可分的几率会高一些。
  + 不过，通常使用的kernel都是隐式的，也就是找不到显式地把数据从低维映射到高维的函数，而只能计算高维空间中的数据点的内积。
  + 在这种情况下，LR就不能再表示成原始形式wTx+b （primal form），而只能表示成对偶形式（dual form）,不仅需要存储各个ai，还要存储训练数据xi本身，这个存储量就大了。
* 相比之下，SVM的对偶形式是稀疏的，即只有支持向量的ai才非零，才需要存储相应的xi，所以，在非线性可分的情况下，SVM用的更多
* LR是一种判别模型（discriminative），直接对条件概率P(y|x)建模，而不关心背后的数据分布P(x,y),而高斯贝叶斯（Gaussian Naive Bayes）是一种生成模型（generative），先对数据的联合分布建模，再通过贝叶斯公式来计算属于各个类别的后验概率（posterior）
* Softmax 回归是对LR在多分类的推广，也叫多元逻辑回归（Multinomial LR）
* LR与SVM
  + 损失函数不同，LR用的是logistical loss，SVM是hinge loss
  + 但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只考虑support vectors，也就是和分类最相关的少数点，去学习分类器
  + LR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，根本目的一样
  + 根据需要，两个方法都可以增加不同的正则化项，如l1,l2等等
  + LR模型易于理解，特别是大规模线性分类时比较方便；SVM的理解和优化相对来说复杂，转化为对偶问题后，分类只需要计算与少数几个support vector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。
  + 对异常的敏感度
    - LR中每个样本都是有贡献的，最大似然后会自动压制异常的贡献
    - SVM对异常比较敏感，因为其训练只需要支持向量，有效样本本来就不高，一旦被干扰，预测结果难以预料

### K近邻

* 当训练集、距离度量、K值以及分类决策规则确定后，对于任何一个新的输入实例，它所属的类唯一地确定。

**距离度量**

* 欧式距离
* 曼哈顿距离（L1）
* 切比雪夫距离
* 闵可夫斯基距离（包含前三种）
* 标准化欧氏距离
* 夹角余弦

**K值的选择**

* 较小的k值，学习的近似误差（training error）会减小，估计误差（test error）会增大，整体模型变得复杂，容易过拟合（容易受到训练数据的噪声影响）
* 较大的k值，学习的估计误差会减小，近似误差会增大，整体的模型变得简单
* k=N，无论输入实例是什么，都将简单地预测它属于在训练实例中最多的类，模型过于简单，完全忽略最近邻列表中可能包含远离其近邻的数据点，是不可取的
* K值一般取一个比较小的数值，通常采用交叉验证法（cv）来选取最优的K值（经验规则：K一般低于训练样本数的平方根）

**分类决策规则**

* 多数表决

**优点**

* 简单、易于理解、易于实现
* 无需估计参数、无需训练
* 适合对稀有事件（rare event)进行分类
* 特别适合多分类问题（multiclass），如根据基因特征来判断其功能分类，KNN比SVM的表现要好

**缺点**

* 懒惰算法，分类计算量大，内存开销大（因为要扫描全部训练样本并计算距离），已经有一些方法提高计算的效率，例如压缩训练样本量
* 可解释性较差，无法像决策树那样
* 当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的K个邻居中大容量类的样本占多数
* 消极学习法（lazy learner），相对应的是积极学习（eager learner）的决策树

### 决策树

* 具有可读性，分类速度快
* （内部-一个特征或属性/叶-一个类）结点，有向边
* 由根结点到叶节点的每一条路径构建*规则*；路径上内部结点的特征对应着规则的*条件*，而叶结点的类对应着规则的*结论*
* 性质：互斥且完备（每一个实例都被一条路径或一条规则所覆盖，且只被一条路径或一条规则所覆盖，这里的覆盖是指实例的特征与路径上的所有特征或规则的条件一致）

**特征选择**

* 准则：Info Gain
* 熵：, 当pi=0时，定义熵为0，单位为比特或者纳特
* 条件熵：
* 经验熵和经验条件熵：当熵和条件熵中的概率由数据估计（特别是极大似然估计）得到时，所对应的熵与条件熵分别称为经验熵和条件经验熵
* 信息增益：表示得知特征X的信息而使得类*Y的不确定性减少的程度*，对数据集D，信息增益
* 信息增益比：信息增益与训练数据集D关于特征A的值的熵之比
* Gini系数：假设有K个类，样本点属于第k类的概率为pk, 则，基尼系数越大，样本集合的不确定性越大，与熵类似

**算法**

* ID3
  + 从根结点开始，对结点计算所有可能的特征的信息增益，选择信息增益最大的特征作为结点的特征，由该特征的不同取值建立子结点
  + 再对子结点递归地调用以上方法，构建决策树
  + 直到所有特征的信息增益均很小，或没有特征可以选择为止
  + 用极大似然法进行概率模型的选择
  + 容易过拟合
* C4.5
  + 改进了ID3，用信息增益比来选择特征
* CART
  + 由特征选择、树生成及剪枝组成，既可用于分类也可用于回归
  + 假定决策树是二叉树，用基尼系数（Gini index）最小化准则，进行特征选择
  + 用验证数据集对已生成的树进行剪枝，并选择最优子树，用损失函数最小作为剪枝的标准

**剪枝**

* 为对训练数据的预测误差（如基尼系数），为子树的叶结点个数，为参数,为参数是a时的子树T的整体损失，参数a权衡训练数据的拟合程度与模型的复杂度
* a 大的时候，最优子树Ta偏小；当a小的时候，最优子树Ta偏大；a=0时，完整的树是最优的；a→∞ 时，根结点组成的单结点树是最优的
* 在剪枝得到的子树序列T0,T1,…,Tn中通过交叉验证选取最优子树Ta

### 随机森林

* bootstrap抽样，特征维度M，选择维度m
* 可生成一个矩阵，用于度量样本之间的相似性： 表示样本i和j出现在随机森林中同一个叶子结点的次数，N为随机森林中树的颗数
* 对generlization error使用的是无偏估计

**优点**

* 随机性的引入，使得RF不容易过拟合
* 随机性的引入，使得RF有很好的抗噪声能力
* 很好的高维度（feature很多）的数据
* 不用做特征选择
* 对数据集的适应能力强：既能处理离散型数据，也能处理连续型数据，数据集无需规范化
* 训练速度快，可以得到变量重要性排序（两种：基于OOB误分率的增加量，基于分裂时的GINI下降量）
* 在训练过程中，能够检测到feature间的互相影响
* 容易做成并行化方法
* 实现比较简单

**影响分类效果的参数**

* 森林中任意两棵树的相关性：相关性越大，错误率越大
* 森林中每棵树的分类能力：每棵树的分类能力越强，整个森林的错误率越低
* 减小特征选择个数m，树的相关性和分类能力也会相应的降低；增大m，两者也会随之增大，所以关键问题是如何选择最优的m

**袋外误差率(OOB)**

* n个样本的训练集随机采样，采集n次，当n→∞时，，也就是Out of Bag的数据，这些数据没有参与训练集模型的拟合，因此可以用来检测模型的泛化能力
* oob计算方式
  + 对每个样本计算它作为oob样本的树对它的分类情况
  + 以简单多数投票作为该样本的分类结果
  + 最后用误分个数占样本总数的比率作为随机森林的oob误分率

**算法**

* 两个随机：抽样随机，特征选择随机
* 有放回抽样的原因：如果不是有放回的抽样，那么每棵树的训练样本都是不同的，没有交集，这样每棵树都是片面的，训练出来有很大的差异，随机森林最后分类取决于多棵树（弱分类器）的投票表决，这种表决应该是”求同”，因此使用完全不同的训练集来训练每棵树这样对最终分类结果没有帮助
* RF相比bagging的改进
  + RF选与输入样本的数目*相同*多的次数（可能一个样本会被选取多次，同时也会造成一些样本不会被选取到），而bagging一般选取比输入样本的数目少的样本
  + bagging是用全部特征来得到分类器，而RF是需要从全部特征中选取其中的*一部分*特征来训练得到分类器； 效果更好
* 不剪枝的原因：之前的两个随机采样的过程保证了随机性，所以就算不剪枝，也不会出现over-fitting。，按这种算法得到的随机森林中的每一棵都很弱，但是组合起来很厉害
* 变体：不用决策树，反而把SVM，LR的分类器当作一棵树的方法也叫RF。比如回归问题，做100次bootstrap，每次得到的数据Di长度为N，对于每一个Di，使用局部回归(LOESS)拟合一条曲线，将这些曲线取平均，得到的最终拟合曲线更加稳定，并且过拟合明显减弱

### AdaBoost

**集成学习**

* ensemble learning是指通过构建多个弱学习器，然后结合为一个强学习器来完成分类任务。集成学习并不算是一种分类器，而是一种学习器结合的方法，通过使用多个决策者共同决策一个实例的分类从而提高分类器的泛化能力
  + 首次按产生一组“个体学习器”，这些个体学习器可以是同质的（homogeneous）（例如全部是决策树），这一类学习器被称为基学习器（base learner），相应的学习算法称为“基学习算法”
  + 集成也可包含不同类型的个体学习器（例如同时包含决策树和神经网络），这一类学习器被称为“组件学习器”（component learner）
* 条件
  + 分类器之间应该具有差异性，多样性
  + 每个个体分类器的分类精度必须大于0.5，如果p<0.5那么随着集成规模的增加，分类精度会下降；反之，最后最终分类精度是可以趋于1的
* 分类
  + 个体学习器之间存在强依赖关系、必须串行生成的序列化方法，代表是Boosting
  + 个体学习器间不存在强依赖关系、可同时生成的并行化方法，代表是Bagging和RF
  + Bagging通过降低基分类器方法来改善泛化能力，因此性能依赖于基分类器的稳定性
    - 如果基分类器是不稳定的，Bagging有助于减低训练数据的随机扰动导致的误差
    - 如果基分类器是稳定的，即对数据变化不敏感，那么Bagging方法就得不到性能的提升，甚至会降低
  + Boosting是一个迭代的过程，通过改变样本分布，使得分类器聚集在那些很难分的样本上，对那些容易错分的数据加强学习，增加错分数据的权重，这样错分的数据再下一轮的迭代就有更大的作用（对错分数据进行惩罚）
  + Bagging与Boosting的区别
    - 取样方式不同
      * Bagging采用均匀取样，Boosting根据错误率来取样，因此Boosting的分类精度要优于Bagging
      * Bagging的训练集的选择是随机的，各轮训练集之间相互独立，而Boostlng的各轮训练集的选择与前面各轮的学习结果有关
      * Bagging的各个预测函数没有权重，而Boosting是有权重的
      * Bagging的各个预测函数可以并行生成，而Boosting的各个预测函数只能顺序生成
      * 对于神经网络这样极为耗时的学习方法，Bagging可通过并行训练节省大量时间开销
    - bagging是减少variance，而boosting是减少bias

**算法**

* 每一轮改变训练数据的权值或概率分布：提高前一轮弱分类器错误分类样本的权值，降低正确分类样本的权值
* 对弱分类器的组合：加权多数表决。加大分类误差率小的弱分类器的权值，使其在表决中起较大的作用；减小分类误差率较大的弱分类器的权值，使其在表决中起较小的作用
* 具体步骤
  + 初始化训练样本的权值分布。如果有N个样本，则每一个训练样本最开始时都被赋予相同的权值：1/N
  + 训练弱分类器。每一轮改变训练数据的权值或概率分布，权值更新过的样本被用于训练下一个分类器，整个训练过程迭代地进行下去
  + 组合弱分类器。误差率低的弱分类器在最终分类器中权重较大，否则较小，迭代得到最终分类器
* 可以证明Adaboost的训练误差是以指数速率下降的
* 算法的数学推导
  + 加法模型additive model
  + 损失函数极小化
  + 前向分布算法forward stagewise algorithm

### GBDT

* Gradient Boosting Decision Tree，又叫 MART（Multiple Additive Regression Tree），广泛应用在搜索排序、点击率预估，竞赛中最为常用的一种机器学习算法

**回归树Regression DT**

* 最优切分变量和最优切分点衡量的准则不再是分类树中的基尼系数，而是平方误差最小化
* 分枝直到每个叶子节点上的值唯一或者达到预设的终止条件（比如叶子个数上限），那则预测为该节点上的平均值
* GBDT的核心在与叠加每一步每一棵树拟合的残差和选择分裂点的评价方式

**提升树Boosting DT**

* 提升方法采用加法模型（即基函数的线性组合）与前向分布算法，提升树是迭代多棵回归树来共同决策
* 当采用平方误差损失函数时，每一棵回归树学习的是之前所有树的结论和残差，拟合得到一个当前的残差回归树，提升树即是整个迭代过程生成的回归树的累加。模型表示为，其中T为决策树，为决策树的参数，M为树的个数
* Boosting DT算法
  + 初始化模型
  + 对
    - 计算残差
    - 拟合残差学习一个回归树，得到
    - 更新

**算法**

* 假设前一轮得到的强学习器为，对应的损失函数则为,新一轮迭代的目的就是找到一个弱分类器，使得损失函数达到最小
* 初始化弱分类器，估计使损失函数极小化的一个常数值，此时树仅有一个根结点：
* 对
  + 对i=1,2,…,N，计算损失函数的负梯度值在当前模型的值，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
  + 对拟合一个回归树，得到第m棵树的叶结点区域
  + 对，计算，即利用线性搜索估计叶结点区域的值，使损失函数极小化
  + 更新回归树：

**问题**

* 为什么xgboost/gbdt在调参时为什么树的深度很少就能达到很高的精度？
  + 参考bagging和boosting的核心不同点

### XGBoost

**监督学习的要素**

* 模型，参数，目标函数：误差函数+正则化项
* 误差函数：如平方损失，logistic损失:
* 正则化项：如L1正则(lasso稀疏)，L2正则(Ridge平滑)，通过约束噪声和outlier数据的权重，降低了模型复杂度，从而预防了过拟合，提高了泛化性

**推导**

* xgboost的误差函数：平方误差->二阶泰勒展开
* 目标函数的最小化：解一个关于W的一元二次方程最小值问题，得到最小（最好）的结构分数（structure score，类比Gini）
* xgboost利用贪心法计算信息增益，比枚举每个树的结构分数效率高
* 为了限制树的生长加入阈值γ，当增益大于阈值时才让节点分裂，它是正则项里叶子节点数T的系数，所以xgboost在优化目标函数的同时相当于做了预剪枝
* 系数λ是正则项里leaf score的L2模平方的系数，对leaf score做了平滑，也起到了防止过拟合的作用，GBDT里不具备

**问题**

* GBDT和XGBOOST的区别？
  + 基分类器的选择：GBDT只能以CART作为基分类器，XGBoost可以是CART或者线性分类器，后者相当于带L1和L2正则化项的LogitR（分类问题）或者LinearR（回归问题）
  + 二阶泰勒展开：GBDT在优化时只用到一阶导数，XGBoost用到了二阶。XGBoost工具支持自定义损失函数，只要函数可一阶和二阶求导
  + 方差-方差权衡（Bias-variance tradeoff)：XGBoost在正则项里包含了树的叶子节点个数T、每个叶子节点上输出分数的L2模的平方和。正则项降低了模型的variance，使学习出来的模型更加简单，防止过拟合
  + 列抽样（column subsampling）：XGBoost借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算
  + 缺失值处理：XGBoost考虑了训练数据为稀疏值的情况，可以为缺失值或者指定的值指定分支的默认方向
  + XGBoost支持并行：XGBoost也是一次迭代完才能进行下一次迭代，但其并行是在特征上的
    - 决策树的学习最耗时的是对特征的值进行排序（因为要确定最佳分割点）
    - XGBoost在训练之前，预先对数据进行了排序，然后保存为block(块)结构，后面的迭代中重复地使用这个结构，大大减小计算量
    - block结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以多线程进行
  + 线程缓冲区存储：按照特征列方式存储能优化寻找最佳的分割点，但是当并行计算梯度数据时会导致内存的不连续访问，严重时会导致cache miss，降低算法效率
    - 可先将数据收集到线程内部的buffer（缓冲区），主要是结合多线程、数据压缩、分片的方法，然后再计算，提高算法的效率
  + 可并行的近似直方图算法：树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点
    - 当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以xgboost还提出了一种可并行的近似直方图算法，用于高效地生成候选的分割点
    - 根据百分位法列举几个可能成为分割点的候选者，然后从候选者中根据上面求分割点的公式计算找出最佳的分割点
* 为什么在实际的 kaggle 比赛中 gbdt 和 random forest 效果非常好？
  + ensemble，更不容易overfit
  + 基于树的算法抗噪强，比如容易对缺失值进行处理
  + feature一般有很多，更适合tree based

**其他**

* Shrinkage（缩减）：学习速率（xgboost中的ϵ）。XGBoost每次迭代后会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。一般把ϵ设置得小一点，然后迭代次数设置得大一点。（传统GBDT也有学习速率）

### 朴素贝叶斯

* Naive Bayes对于给定的训练数据集，首先基于特征条件独立假设学习输入、输出的*联合分布*；然后基于此模型，对给定的输入x，利用贝叶斯定理求出后验概率最大的输出y
* NB的强假设：各个特征互相独立

**推导**

* 有k个特征，
* 求
* 贝叶斯定理为，分母可以分解为
* 其中先验(prior)概率从数据集已知
* 强假设可得，参数规模为，是第i个特征可以取值的个数
* 于是贝叶斯分类器为
* 分母对argmax没有影响，可以表示为

**参数估计**

* 先验概率极大似然估计是用样本中ck的出现次数除以样本容量
* 如果训练数据中没有出现某种参数与类别的组合，MLE估计会是0，会影响到后验概率的计算结果，使分类产生偏差
  + 贝叶斯估计：多了一个λ，当λ=0时就是极大似然估计，常取λ=1，这时称为*拉普拉斯平滑*Laplace Smoothing
  + 先验概率的Bayes Estimate：
  + 条件概率的Bayes Estimate：

### 聚类

* clustering，无监督算法，对于给定的类别数目K，首先给出初始划分，通过迭代改变样本和簇的隶属关系，使得每一次改进之后的划分方案都较前一次好

**分类**

* 基于分层的聚类（hierarchical methods）
  + BIRCH算法，CURE算法，CHAMELEON算法
  + 对给定的数据集进行逐层，直到某种条件满足为止
  + 分为合并型的“自下而上”和分裂型的“自下而上”
  + 自下而上：初始时每一个数据记录都组成一个单独的组，在迭代中把邻近的组合并成一个组，直到所有的记录组成一个分组或者某个条件满足为止
* 基于划分的聚类（partitioning methods）
  + K-means算法，K-medoids算法，CLARANS算法
  + 给定一个有N个记录的数据集，分裂法构造K个分组，每一个分组就代表一个聚类，k < N
  + 每一个分组至少包含一个数据记录
  + 每一个数据记录属于且仅属于一个分组
  + 对于给定的K，首先给出一个初始的分组方法，反复迭代改变分组，使得每一次改进之后，同一分组中的记录更近，不同分组中的记录更远
* 基于密度的聚类（density-based methods）
  + DBSCAN算法，OPTICS算法，DENCLUE算法，WaveCluster算法
  + 不基于距离，只要一个区域的点的密度大过某个阈值，就把它加到与之相近的聚类中去
* 基于网格的聚类（grid-based methods）
  + STING算法，CLIQUE算法，WaveCluster算法
  + 将数据空间划分成为有限个单元（cell）的网络结构，所有的处理都是以单个的单元为对象的
  + 优点：处理速度快，通常与数据数量无关，只与分成多少单元有关
* 基于模型的聚类（model-based methods）
  + 给每一个聚类假定一个模型，然后去寻找能够很好地满足这个模型的数据集
  + 目标数据集是由一系列的概率分布所决定
  + 通常有两种尝试方向：统计的方案和神经网络的方案

**方法索引**

* 闵可夫斯基距离（Minkowski）
  + p=1，曼哈顿距离（Manhattan）
  + p=2，欧几里得距离
  + p=∞，最大值距离
* 杰卡德相似系数（Jaccard）
* 余弦相似度（Cosine Similarity）
* pearson相似系数
* 相对熵（K-L）距离
* Hellinger距离

### K-Means

* 基于划分(partitioning method)的聚类，是一种最简单的无监督学习方法
* 通过迭代来实现，每次确定K个类别中心，然后将各个结点归属到与之距离最近的中心点所在的Cluster，然后将类别中心更新为属于各Cluster的所有样本的均值，反复迭代，直至类别中心不再发生变化或变化小于某阈值。
* 基本假设是对于每一个cluster，可以选出一个中心点 (center) ，使得该 cluster中的所有的点到该中心点的距离小于到其他cluster的中心的距离。虽然实际情况中得到的数据并不能保证总是满足这样的约束，但这通常已经是我们所能达到的最好的结果，而那些误差通常是固有存在的或者问题本身的*不可分性*造成的。

**算法**

* 初始输入为
* 选择K个类别中心。
  + 这个过程通常是针对具体地问题有一些启发式的选取方法，或者大多数情况下采用随机选取的办法
  + 因为K-Means并不能保证全局最优，而*是否能收敛到全局最优解和初值的选取有很大的关系*
  + 有时候我们会多次选取初值跑一个K-Means，并取其中最好的一次结果
* 对于每个样本，将其标记为距离类别中心最近的类别
* 将每个类别中心更新为隶属于该类别的所有样本的均值
* 重复直到类别中心的变化小于某阈值或者达到最大迭代次数

**分析**

* 设我们一共有N个数据点需要分为K个Cluster，K-Means需要最小化的损失函数
  + 其中在在数据点n被归类到Cluster(j)的时候为1，否则为0
  + 直接寻找和来最小化J并不容易，但可以对两者分别迭代
  + 固定，选择最优的，只要将数据点归类到离它最近的那个中心就能保证J最小，因为r不是0就是1，要想让J最小，就要保证当一个样本的rij=1时，与类别中心距离的平方和达到最小
  + 固定，求最优的，将J对μk求导并令导数等于0，得到μj 的值是所有Cluster(j)中的数据点的平均值
  + 由于每一次迭代都是取到J的最小值，因此J只会不断地减小或者保持不变，而不会增加，这保证了K-Means最终或到达一个极小值。
* 优点
  + 聚类问题的经典算法，简单快速
  + 对处理大数据集，该算法保持可伸缩性和高效率
  + 当簇近似为高斯分布时，它的效果较好
* 缺点
  + 在簇的平均值可被定义的情况下才能使用，可能不适用于某些应用
  + 必须事先给出K，对初值敏感，对于不同的初始值，结果可能不同
  + 只能发现球状Cluster，不适合于发现非凸形状的簇或者大小差别很大的簇
  + 对噪声和孤立点数据敏感，如簇中含有异常点，将导致均值偏离严重
    - 因为均值体现的是数据集的*整体特征*，容易掩盖数据本身的特性
    - 比如数组1，2，3，4，100的均值为22，显然距离“大多数”数据1、2、3、4比较远，如果改成数组的中位数3，在该实例中更为稳妥，这种聚类也叫作K-mediods聚类

### DBSCAN

* 密度聚类
  + 只要样本点的密度大于某阈值，则将该样本添加到最近的簇中
  + 这类算法能克服基于距离的算法只能发现“类圆”（凸）的聚类的缺点，可发现任意形状的聚类，且对噪声数据不敏感
  + 但计算密度单元的计算复杂度大，需要建立空间索引来降低计算量
  + 代表算法是DSBSCAN和密度最大值算法
* DBSCAN(Density-Based Spatial Clustering of Applications with Noise)

**概念**

* 对象的ε−领域：给定对象在半径ε内的区域
* 核心对象(core points)：对于给定的数目m(minPts)，如果一个对象的ε−领域至少包含m个对象，则称该对象为核心对象
* 直接密度可达(directly reachable)：给定一个对象集合D，如果p是在q的ε−领域内，而q是一个核心对象，则从对象q出发到对象p直接密度可达
* 密度可达(reachable)：如果存在一个对象链p1p2⋅⋅⋅pn, p1=q, pn=p, 每个都是从pi直接密度可达，则从对象q出发到对象p密度可达（也就是说只有pn可以例外）
* 噪声(outliers/noise points)：从任何点出发都不能reachable的点是噪声
* 密度相连(density connected)：如果对象集合D中存在一个对象O，使得对p和q从O密度可达，那么对象p和q密度相连，密度相连是symmetric的
* 簇(cluster)：所有在簇里的点密度相连，簇是最大的密度相连对象的集合
* 特征
  + 每个簇至少包含一个核心对象
  + 非核心对象可以是簇的一部分，构成了簇的边缘（edge）
  + 包含过少对象的簇是噪声

**算法**

* 给定ε和m，从一个为没有被访问的随机点开始，如果它是核心对象，开始聚类，否则标记为噪声（但也可能以后会成为edge）
* 对核心对象所有直接密度可达的对象，如果他们是噪声，改为edge，如果是核心对象，添加到自己的簇中，直到一个最大密度相连的簇被找到，重复上一步开始新的簇直到所有点被访问

**总结**

* 优点
  + 无需确定聚类个数，相对于k-means
  + 可以发现任意形状的聚类，甚至是环状包括式的簇
  + 可有效处理噪声
  + 参数可由领域专家设置
  + 只需两个参数，对数据输入顺序不敏感 (然而，如果点的顺序发生变化，位于两个不同集群边缘的点可能会交换集群成员资格，并且集群分配仅在同构时是唯一的)
  + m(MinPts)参数减少了单链接效应（single-link effect, 不同的集群由细线点连接）
  + 加快区查询，DBSCAN被设计用于可以加速区域查询的数据库，例如使用 R\* 树
* 缺点
  + 边界点不完全确定性
    - 从多个集群可到达的边界点可以是任一集群的一部分，具体取决于数据处理的顺序
    - 幸运的是，这种情况并不经常出现，并且对聚类结果影响很小
      * 无论是在核心点还是噪声点上，DBSCAN 都是确定性的。
      * DBSCAN有一种将边界点视为噪声的变体，这种方式可以获得完全确定性的结果以及对密度连接性的更一致的统计解释
  + 维数灾难导致欧几里得距离度量失效
    - DBSCAN 的质量取决于函数 regionQuery(P,ε) 中使用的距离度量
    - 最常用的距离度量是欧几里得距离
    - 然而特别是对于高维数据，由于所谓的“维数灾难” (curse of dimensionality)，这个度量几乎变得毫无用处，很难找到合适的 ε 值
    - 但这种影响也存在于任何其他基于欧几里德距离的算法中
  + 不能处理密度差异过大（密度不均匀）的聚类
    - DBSCAN不能很好地聚类密度差异很大的数据集，因为不能为所有聚类适当地选择 minPts-ε 组合
  + 参数选择在数据与规模不能很好理解的情况下，很难选择，若选取不当，聚类质量下降

### 密度最大值聚类

* todo

### SVM

**线性硬间隔SVM**

* 利用间隔最大化求分离超平面，解是唯一的
* 函数间隔functional margin
  + 一个点距离分离超平面的远近表示分类预测的确信程度
  + 在超平面确定的情况下|wT⋅x+b|相对地表示点x距离超平面的远近
  + wT⋅x+b的符号与类标记的符号是否一致表示分类是否正确
* 几何间隔geometric margin
  + 改进不足：在选择分离超平面时，只要成比例地改变w和b，超平面并没有变化，而函数间隔却以同样比例变化
  + 几何间隔对分离超平面的法向量w加上某些约束，使得间隔确定
  + 几何间隔γ=函数间隔γ’/||w||
* 最大间隔分离超平面
  + 最大化几何间隔γ，且每个训练样本点的几何间隔至少是γ
  + max(w,b) γ, s.t. for all i, yi(wxi+b)/||w|| ⩾ γ
  + 等同于max(w,b)函数间隔γ’/||w||, s.t. for all i, yi(wxi+b) ⩾ γ’
  + 函数间隔γ’的取值不影响最优化的解。函数间隔因为w，b按比例改变为λw，λb而成为λγ’，但是对最优化问题中的不等式约束没有影响，对目标函数的优化也没有影响，即两者等价
  + 于是可以取γ’=1，代入后注意到最大化1/||w||和最小化1/2||w||2也是等价的，得到线性可分支持向量机学习的最优化问题：min(1/2||w||2) s.t. for all i, yi(wxi+b)-1 ⩾ 0
  + 得到分割超平面 w’x+b’=0，分类决策函数f(x)=sign(w’x+b)
  + 本质是一个凸二次规划convex quadratic programming 问题(todo)
* 支持向量support vector
  + 在线性可分情况下，训练数据集的样本中与分离超平面距离最近的样本点的实例成为支持向量，是使约束条件式等号成立的点
  + 对yi=+1的实例点，支持向量在超平面H1：wxi+b=1
  + 对yi=−1的负例点，支持向量在超平面H2：wxi+b=-1
  + 它们到中间的分离超平面的距离相等，因为如果某超平面没有碰到任意一个点的话，就可以进一步地扩充中间的gap，这个就不是最大的margin
  + H1和H2称为间隔边界H1和H2平行，中间形成一条长带分离超平面，没有实例点落在其中
  + 长带的宽度即H1与H2之间的距离成为间隔margin，等于2/||w||
  + 在决定分离超平面时只有支持向量起作用，其他实例点不起作用
  + 如果移动支持向量将改变所求的解；但如果在间隔边界以外移动其他实例点，甚至去掉这些点，解不会改变
  + 由于支持向量在确定分离超平面中起决定性作用，所以将这种分类模型称为支持向量机
  + 支持向量的个数一般都很少，所以支持向量机由很少的“重要的”训练样本确定，这样的特性在实际中有一个最直接的好处就在于存储和计算上的优越性
  + 例如，如果使用100万个点求出一个最优的超平面，其中支持向量有100个，那么只需要记住这100个点的信息，后续分类也只利用这100个点做计算
* 对偶算法dual algorithm
  + 应用拉格朗日对偶性，通过求解对偶问题得到原始问题的最优解
  + 优点：dual更容易求解，引入核函数可以推广到非线性分类问题
  + 首先构造*拉格朗日函数*L，对每一个不等式约束引进拉格朗日乘子(Lagrange multiplier)ai：
  + 然后令θ(w)=max(ai≥0) L(w,b,a)
  + 当某个约束条件不满足时(如yi(wxi+b)<1)，令ai=∞会使θ(w)=∞
  + 当所有约束条件都满足时，θ(w)=1/2||w||\_2，即我们最初要最小化的量
  + 于是在要求约束条件得到满足的情况下最小化1/2||w||2等价于最小化θ(w)：min(w,b) θ(w)=min(w,b) max(ai≥0) L(w,b,a)=p’
  + 把最小和最大的位置交换得到：max(ai≥0) min(w,b) L(w,b,a)=d’
  + d’≤p’，因为最大值中最小的一个不小于最小值中最大的一个
  + d’提供了p’的一个下界，在满足*KKT条件*下，两者相等，就可以通过求解d’来间接地求解p’
  + 要解d’，因为min(w,b)在内部，可以直接求对w,b的偏导并令其为0；之后求对a的对偶最优化问题，最终形式为：
  + 至此对于线性可分问题，上述硬间隔最大化算法是完美的，但训练数据集往往是线性不可分的，即在样本中出现噪声或特异点，此时有更一般的学习算法
* KKT条件(Karush-Kuhn-Tucker)
  + todo，见https://zhuanlan.zhihu.com/p/26514613

**线性软间隔SVM**

* 软间隔最大化
  + 通常情况是训练数据中有一些outlier，除去后剩下大部分的样本点组成的集合线性可分，线性不可分意味着某些样本点不能满足函数间隔大于等于1的约束条件
  + 可以对每个样本点引进一个松弛变量ξ⩾0，使函数间隔加上松弛变量大于等于1，约束条件遂变为 yi(wxi+b) ⩾ 1-ξi
  + 同时对每个松弛变量ξ⩾0，支付一个代价ξ，但如果我们允许ξ⩾0任意大，那任意的超平面都符合条件，所以在原来的目标函数后面加上一项，使得这些 ξ⩾0的*总和*最小，目标函数遂变为
  + C>0 为惩罚参数，一般事先由应用问题决定，控制目标函数中“寻找margin最大的超平面”和“保证数据点偏差量最小”之间的权重，C越大时对误分类的惩罚越大
  + 于是，最小化目标函数使分离超平面间隔尽量大，同时使误分类点的个数尽量小，C是调和二者的系数
  + 可证明w的解唯一，但b的解存在于一个区间，不唯一
  + 用对偶算法解得最终最优化问题的拉格朗日乘子a多了一个上限C，其他不变
  + 对任一适合条件都可求得一个b’，实际计算时可以取在所有符合条件的样本点上的平均值
* 支持向量
  + 线性不可分时将对偶问题的解中对应于ai’>0的样本点(xi,yi)的实例xi称为支持向量
  + 软间隔的支持向量xi要么在间隔边界上，要么在间隔边界与分离超平面之间，要么在分离超平面误分类一侧
  + 若ai’<C，则ξi=0，支持向量在间隔边界H上（硬分类无惩罚的上限）
  + 若ai’=C，0<ξi<1，则分类正确，xi在间隔边界与分离超平面之间
  + 若ai’=C，ξi=1，则xi在分隔超平面上
  + 若ai’=C，ξi>1，则xi位于分离超平面误分一侧
* Hinge损失函数
  + 软间隔SVM的dual优化问题等价于，其中第一项是经验损失函数，称为合页损失函数(hinge loss)
  + hinge loss是0-1损失函数的上界，由于0-1损失函数不连续可导，直接优化其构成的目标函数比较困难，可以认为线性SVM是优化由0-1损失函数的上界构成的目标函数。这时的上界损失函数又称为代理损失函数（surrogate function）

**非线性SVM**

* 当输入空间为欧式空间或离散集合、特征空间为希尔伯特Hibert空间时，核函数表示将输入从输入空间映射到特征空间得到的特征向量之间的内积
* 核函数可以学习非线性SVM，等价于隐式地在高维的特征空间中学习线性SVM
* *核方法*kernel method
  + 是比SVM更为一般的机器学习方法
  + 基本思想：首先使用一个变换将原空间的数据映射到新空间；然后在新空间里用线性分类学习方法从训练数据中学习分类模型
  + SVM的情形：假设原始的数据非线性，通过ϕ(⋅)将其映射到一个高维空间中，数据变得线性可分，然后使用原来的推导来进行计算，只是所有的推导现在是在新的空间，而不是原始空间中进行
    - 推导过程不可以简单地直接类比，例如，原本要求超平面的法向量w，但是如果映射后新空间是无穷维的（如高斯核函数todo），要表示无穷维向量就比较麻烦；而且映射数目呈爆炸性增长，这给计算带来了很大困难
    - 核技巧：在学习与预测中只定义核函数K(x,z)，而不显式地定义映射函数ϕ（⋅）。不像之前映射到高维空间中，然后根据内积公式进行计算，而是直接在原来的低维空间中进行计算，而不需要显式的写出映射后的结果。通常，直接计算K(x,z)比较容易，而通过ϕ(x)和ϕ(z)计算K(x,z)并不容易
  + aside
    - 最理想的情况下希望知道数据的具体形状和分布，从而得到一个刚好可以将数据映射成线性可分的ϕ（⋅），然后通过这个ϕ（⋅）得到对应的K(⋅，⋅)进行内积计算
    - 然而，第二步通常是非常困难甚至完全没法做的。不过，由于第一步也是几乎无法做到的，因为对于任意的数据分析其形状找到合适的映射本身就不是什么容易的事情
    - 所以人们通常是“胡乱”选择一个核函数——直到它对应了某个映射，虽然我们不知道这个映射具体是什么，由于计算只需要核函数即可，所以我们也并不关心，也没有必要求出所对应的映射的具体形式
* 核函数
  + 多项式核函数polynomial kernel function
    - 对应的SVM是一个p次多项式分类器
    - 分类决策函数为
  + 高斯核函数gaussian kernel function
    - 这个核会将原始空间映射为无穷维空间
    - 但如果σ选得很大，高次特征上的权重衰减的非常快，所以实际上相当于一个低维的子空间
    - 反之如果σ选得很小，则可以将任意的数据映射为线性可分，但这并不一定是好事，因为随之而来的可能是非常严重的过拟合
    - 总的来说，通过调控参数σ，高斯核具有相当高的灵活性，也是使用最为广泛的核函数之一
    - 对应的SVM是高斯径向基函数分类器
    - 分类决策函数为
  + 字符串核函数
    - 核函数不仅可以定义在欧式空间上，还可以定义在离散数据的集合上
    - 字符串核实定义在字符串集合上的核函数，它在文本分类、信息检索、生物信息学等方面都有应用
  + 线性核 κ(x1,x2)=⟨x1,x2⟩
    - 实际上就是原始空间中的内积
    - 这个核存在的主要目的是使得“映射后空间中的问题”和“映射前空间中的问题”两者在形式上统一
* 总结
  + 对于非线性的情况，SVM选择一个核函数 κ(⋅,⋅) ，通过将数据映射到高维空间来解决在原始空间中线性不可分的问题
  + 由于核函数的优良品质，这样的非线性扩展在计算量上并没有比原来复杂多少
  + 主要归功于核方法，除了SVM之外，任何将计算表示为数据点的内积的方法，都可以使用核方法进行非线性扩展

**SMO启发算法**

* todo

### EM

### PCA

### SVD

### 感知机

* 线性方程对应特征空间中的一个超平面S，其中w是超平面的法向量，b是超平面的截距，超平面将特征空间划分为两个部分

**算法**

* Perceptron采用不同的初值或选取不同的误分类点，解可以不同
* 原始形式
  + 只有当一个实例点被*误分类*时，调整w,b的值，使分离超平面向该误分类点的一侧移动，减少该误分类点与超平面的距离，直至超平面越过该误分类点使其被正确分类
* 对偶形式
  + 将w和表示为实例和标记线性组合的形式
  + 当时，表示第i个实例点由于误分而进行更新的次数。实例点更新次数越多，意味着它距离分离超平面越近，也就越难正确分类，这样的实例对学习结果影响最大
  + 对偶形式的训练实例仅以内积的形式出现。为了方便，可预先将训练实例间的内积计算出来并以矩阵的形式存储，这个矩阵就是*Gram矩阵*，
  + 对偶形式迭代是收敛的，存在多个解
* 对偶形式算法
  + *线性可分的*训练数据集
  + 输入
  + choose random
  + if
  + else done
* 原始形式适合online learning，即根据新数据的到来继续更新感知机；对偶形式更适合offline learning，因为储存了一个Gram内积矩阵
* 感知机收敛定理证明：当线性可分时，必定能找到解（待补充）

### 神经网络简述

### 神经网络优化方法

### 神经网络防止过拟合方法

### Batch Normalization

### CNN

### RNN

### LSTM

### 词向量和语言模型

### Word2Vec

### TF-IDF

### 问题汇总

**判别式和生成式模型的区别**

* 生成式模型估计它们的联合概率分布P(x,y)，关注数据是如何生成的，反映同类数据本身的相似度，不关心到底划分不同类的边界在哪
* 判别式模型估计条件概率分布P(y|x)，关注类别之间的差别
* 生成式模型可以根据贝叶斯公式得到判别式模型，但反过来不行
* 判别式模型主要有：
  + logit R, Linear R
  + SVM, Neural Network
  + KNN
  + Gaussian Process高斯过程
  + CRF条件随机场
  + Boosting
  + CART
  + LDA(linear Discriminant analysis)线性判别分析
* 生成式模型主要有:
  + NB(Naive Bayes), BN(Bayes Network)
  + HMM
  + Mixture Gaussians
  + Sigmoid Belief Network
  + Markov Random Fields
  + DBN深度信念网络
  + Latent Dirichlet Allocation

**时间序列模型**

* AR模型：自回归模型，是一种线性模型，已知N个数据，可由模型推出第N点前面或后面的数据（设推出P点），所以其本质类似于插值
* MA模型：移动平均法模型，使用趋势移动平均法建立直线趋势的预测模型
* ARMA模型：自回归滑动平均模型，模型参量法高分辨率谱分析方法之一，研究平稳随机过程有理谱的典型方法。它比AR模型法与MA模型法有较精确的谱估计及较优良的谱分辨率性能，但参数估算比较繁琐
* GARCH模型：广义ARCH模型，是ARCH模型的拓展， GARCH对误差的 方差进行建模，特别适用于波动性的分析和 预测

**无监督学习**

* 模型
  + 聚类
  + 自编码器auto-encoder
  + 主成分分析PCA
  + GAN
* 算法
  + EM

**GBDT和RF比较**

* 参考bagging和boosting的对比
* 都是由多棵树组成和决定结果
* RF可以是分类树，也可以是回归树；GBDT只由回归树组成
* RF可以并行；GBDT只能串行
* RF采用多数投票；GBDT将结果加权累加
* RF对异常值不敏感，GBDT对异常值非常敏感
* RF对数据一视同仁，GBDT是基于权值的弱分类器的集成
* RF减少variance提高性能，GBDT减少bias提高性能

### 参考资源

* <https://plushunter.github.io/tech-stack/>