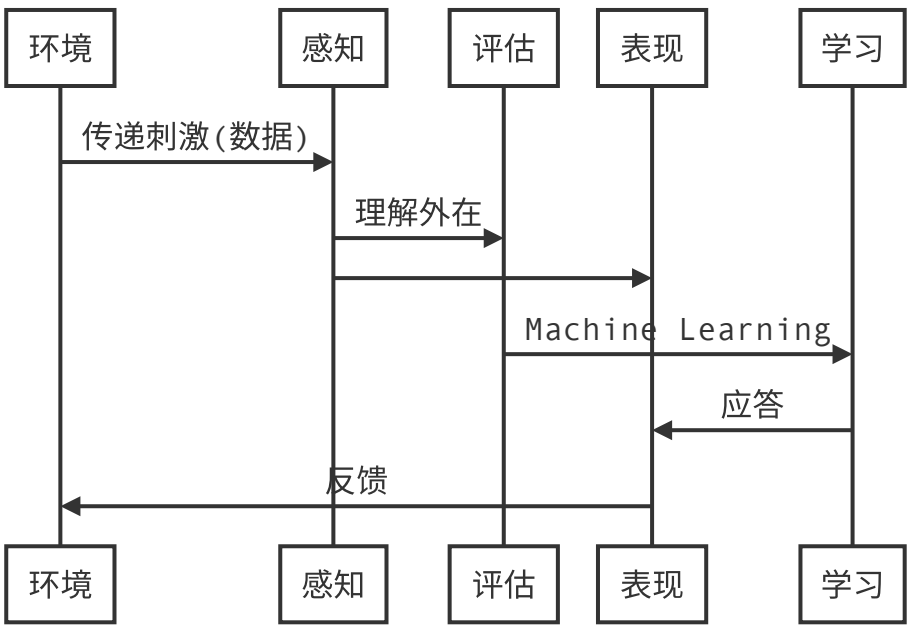
Machine Learning

Before

1. 机器学习(学习算法)的定义

   假设用P来估计计算机程序的某任务类T上的性能，若一个程序可以通过利用经验E在T上获得了性能改善，我们就说关于特定的P T,改程序对经验E进行了学习

2. P性能度量
3. T具体任务
   - 分类
   - 回归
4. E经验数据
5. 我们的角度
   1. 算法性能
   2. 算法效率
6. 困难
   1. 训练困难(维数灾难，特定算法的训练困难(e.g. ANN / RNN))
   2. 泛化困难(过拟合)
7. 前景
   1. 数据挖掘 : 探索(尽可能逼近)大数据背后的真实的概率分布
   2. 理解人类
8. 学习流程



9. 学习的理解
   1. 目标函数
   2. 优化方式
   3. 学习方法(加入我的理解)

        - 监督学习
        - 无监督学习
        - 强化学习
10. 学习什么

    学习函数

    - 数据集
    - 离散函数
    - 连续函数
    - 概率分布
11. 解决的问题
    - 分类
    - 预测
        - 回归
        - 数值型预测

Supervised L earning

拟合函数

- 定义函数
- 优化函数

1.决策树

1. 用来解决离散数据进行多分类的问题
2. 定义问题 : 从解空间中尽可能找到最好的一棵树(简单并且泛化能力强)
3. 优化方案 : 利用信息增益递归进行最有特征选择
4. 避免过拟合

   奥卡姆剃刀原则作为知道思想尽可能的简化树(MDL作为判断复杂程度的工具)
    - 暂停生长
    - 归并树
5. 优点
    - 利于表示和理解
6. 缺点
    - 非常容易过拟合
    - 问题解决是连续的，不能考虑多个变量特征同时作用的情况(类似于朴素贝叶斯分析里的对特征的简单的假设) : 引入贝叶斯分析

2.朴素贝叶斯分析

基于贝叶斯公式

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- 利用先验概率和收集的数据

针对不同的问题的算法变体的采用

- 朴素贝叶斯分析 :
    - 特征没有任何关系
    - 特征是独立同分布的
- 贝叶斯信念网络

    - 特征之间存在部分关系
    - 网络(存在关联关系的特征之间存在连接)
- 高斯混合模型
    - 对所有的特征利用高斯混合模型累计
    - 为了扩大函数空间(引入加权和)

Unsupervised L earning

概率分布函数

- 学习数据的内在的分布的概率(没有人类的先验知识的情况下)
- 可能的问题
    - 学习的效果难以检验
    - 学习的结果是否正确或者是否足够逼近真实分布一无所知(局部最优)
    - 相似度的计算
    - 如何划分数据集
    - 如何处理过拟合和维数灾难

1.分类聚类

- K-Meams聚类
    - 优点
        1. 速度快(适用于大规模计算)
        2. 保证收敛
        3. 容易实现
    - 缺点
        1. 噪声敏感(孤立点对均值的影响过大)
        2. 易陷入局部最优
        3. 初始值敏感
        4. 超参数K
    - 优化目标
        - 均值距离之和最小化
    - 为了得到更好的效果
        1. 我们可以多次执行K均值聚类，选取最优值
        2. 模糊K均值聚类
        3. 最优化算法跳出局部最优
- K-Medoids(PAM聚类
    - 思想 : 使用某一个中心点来代替均值表示一个簇，数据被按照到中心点的距离进行归类

      需要计算替换代价(目标)
    - 优点
        - 对噪声和孤立点不敏感
    - 缺点
        - 收敛速度慢，运算开销大
        - 算法伸缩性不好,比较适合小规模数据
- CLARA(全称是大型应用聚类方法,对K-Medoids的伸缩性优化)

  将优化的K-Medoids为了提高计算效率，适用于大规模数据集上，采用CLARA算法作为优化
    1. 思想 : 抽样上进行K中心点算法，之后将所有的数据分配上去
    2. 仍然需要多次抽样取最优值
- CLARANS

  随机CLARA,利用随机搜索技巧提高聚类的效果

1. 思想
    - 构建一个图(图中的顶点是任意一种K组中心点的选取方案)
    - 两个节点之间如果只有一个节点不同存在连边否则不连边
    - 在图上搜索最有的K分类
    - 为了提高图的搜索效率，在随机选择的子图上执行(随机选取一个开始点执行)，知道找到一个局部最优解
    - 仍然需要多次执行取最优值

2.层次聚类

1. 分类
    - 自底向上
    - 自顶向下
2. 避免过拟合
    1. 中途停止策略
3. 合并和分解的依据
    1. 数据相似性
    2. 簇相似性
    3. 不同的相似性度量方式带来不一样的结果
4. 缺点
    1. 效率底　下
    2. 错误不可逆转(算法有严重的顺序依赖性)
5. 优化算法
    1. BIRCH算法(利用分层的平衡迭代缩减和聚类的方法)
        1. 思想
            - 簇信息提取(三元组),聚类特征(CF)
                1. 数据个数
                2. 线性数据之和
                3. 数据平方和
                4. 额外的 : 均值(质心)，SSE(簇误差平方和)，簇直径，簇间距
            - CF具有可加性(合并)
            - 凝聚聚类只需要操作CF
        2. 算法

            构建一个针对簇CF的平衡树

            $(\boldsymbol{B}, \boldsymbol{L}, \boldsymbol{T})$

            - $\boldsymbol{B}$: 扇出系数最大值(直接子节点个数)
            - $\boldsymbol{L}$: 每个叶节点最多的项的个数(项 = 簇)
            - $\boldsymbol{T}$: 簇直径不超过阈值 $\boldsymbol{T}$,会影响树的尺寸

            需要保证每一个簇可以在一页框下存放，根据当前的页框大小决定 $\boldsymbol{B}, \boldsymbol{L}$

            1. 初始化CF树
            2. 简化CF树(合并簇，删除簇)
            3. 在生成的CF树的所有簇的基础上，进行簇为单位的聚类(提高计算效果，算法随意选择)
        3. 递增构建CF树
            1. 从根开始查找路径，知道找到合适的插入数据的叶子节点
            2. 尝试插入叶子节点，否则会执行分裂操作
            3. 下层的插入会影响上层的CF，并且如果实现了节点的分裂的话，下层的解放店分裂还要查看上层的非叶节点的约束
            4. 最后在插入数据之后的树上我们还需要进行合并操作，提高空间利用率
        4. 优缺点

- 优点
    1. 运算规模是线性的(使用质心来代替簇来进行聚类操作)
    2. 聚类效果可以提升
    3. 混合其他的聚类算法
- 缺点
    1. 只是和数值类型数据
    2. 对输入的数据的顺序比较敏感(建树的方面的局限性)

2. CURE算法(代表点聚类算法) - 凝聚聚类
    1. 特点
        - 使用多个代表点来表示一个簇
        - 簇间距离 : 簇代表点之间的距离最小值
        - 代表点开始分布在簇的周围边界上，代表点逐渐的向中心(均值)收缩
    2. 算法
        1. 选取最近的两个簇进行合并

           要点:
            - 代表点在迭代的过程中会逐渐的收缩
            - 选簇间距离最小的两个点进行合并
        2. 更新最近临簇
        3. 合并
            1. 直到簇的个数达到要求
    3. 进一步优化
        1. 随机抽样 / 数据分组 :
        2. 总的算法思路:
            - 抽样出样本集
            - 将样本集分配 p 组
            - 每一组进行一次CURE聚类，假设每一个组的簇数接近 q

              在每组的凝聚过程中，当出现簇凝聚缓慢的时候或者算法结束的时候，删除局外的孤立点，减少噪声的影响
            - 在生成的所有的簇上再次进行CURE算法凝聚聚类
            - 将数据集上的所有的点都分配到最近的代表点所在的簇中完成算法

3. CHAMELEON算法(利用动态建模的层次聚类方法)
    1. 特点

       层次聚类(自底向上 + 自顶向下) + 图聚类算法的思想的融合
    2. 算法层次
        1. 图建模(动态)

           使用KNN构建稀疏图(减少噪声)，只有最K近的点之间存在边(边的权重代表了我们的数据之间的相似度)
        2. 图割(自顶向下)

           将稀疏图进行分割，分离成互不相连的子图

           $P.S.$: 分割的结果不唯一，选取最优的分割结果(被切边的权重之和最小化(目标函数))

           解决方法:

           多层图割技术 (挑选当前数目最多的簇进行分割，直到数目小于我们的限定的阈值为止)

3. 层次聚类(自底向上)
    - 引入概念:

      考虑了两个簇之间的连接边的权重，两个簇内部的连接权重

        1. 相对相互连接性 :
        2. 相对紧密度 :
    - 合并标准
        1. 连个度量都大于阈值合并
        2. 连个度量的乘积最大的合并

3.密度聚类

1. 思想
    - 聚类基于密度(数据点的数目)
    - 认为簇就是密集的区域
    - 数据密度 : 数据空间一定范围内的数据点的数目，梵音了数据的紧致程度
    - 需要定义数据密度的计算方式
2. 优点

   相对于基于距离计算的聚类算法来说(分类聚类，部分的层次聚类算法)
    1. 可以更强的剔除局外点(噪声不敏感)
    2. 可以更准确的适应聚类形状
    3. 一次扫描(计算高效)
3. 优化算法
    1. DBSCAN(基于密度空间聚类算法的在有噪声下的应用)
        1. 数据密度定义
            - 邻域最大半径 ϵ
            - 邻域内最小数据点数 $MinPts$
        2. 概念参数
            - ϵ-邻域 : 一某一个点为中心，半径为ϵ的圆
            - 核心点 : ϵ-邻域内存在至少$MinPts$个点的点
            - 直接密度可达

              某一个点是核心点并且另一个点在核心点的邻域内
            - 密度可达
            - 密度连接
        3. 簇的定义
            1. 密度可达的最大非空数据子集(自己内任意两点密度连接)
            2. 当所有的簇定义后我们可以删除孤立点，去除噪声的影响
    2. DENCLU(基于密度的聚类)
        1. 数据密度定义
            - 影响函数 : 反应两点之间的影响程度的函数，值随着两点的距离变远而变小
                - 高斯影响函数 : 连续
                - 平方波影响函数 : 阈值离散
            - 密度函数 : 数据集中的其他点对该点的影响函数之和(综合所有的点构成一张图)
            - 密度吸引子 : 密度函数(图)的局部极大值点
            - 密度吸引 : 对于某一个密度吸引子，其他的点认为是按密度吸引到该点上，存在一个密度吸引梯度
        2. 簇的分类

1. 按中心定义的簇 : 阈值限定密度吸引子，所有可以密度吸引到该店的数据点构成一个簇
2. 任意形状的簇 : 合并后的中心定义簇(在阈值上存在一个可以连接到两个密度吸引子的点序列即可合并)

3. 算法步骤
    - 网格预聚类 : 加快计算

      每个超空间中记录信息
        - 数据点的个数
        - 数据点指针
        - 数据点的值和(均值)

      只保留超过阈值的网格进行之后的计算
    - 实际聚类 :
        1. 求密度吸引子，得到中心定义簇，今儿合并得到任意形状簇
        2. 只使用超过阈值的网格及其相邻网格
4. 特点
    - 优点 :
        1. 适合应对大量噪声
        2. 高维数据表示
        3. 速度甚至快于DBSCAN
    - 缺点
        1. 需要大量参数 : 相当于是稠密图

4.网格聚类

1. 思想
    - 不同于基于数据密度的算法，基于网格的算法是从数据空间(值域，解空间)角度着眼问题
    - 将数据控件划分成网格，对网格使用聚类的操作实现聚类
    - 为了调高效果，使用多分辨率技术
2. 特点
    - 优点
        1. 计算复杂度和数据量无关，只和值域有关，算法的可伸缩性强
3. 算法
    1. STING(统计信息网格方法)
        1. 算法思想
            - 分层处理网格，高层次的网格在下一层会被再次分解
            - STING扫描底层网格的统计信息(数据类型分布等等)
            - 高层的信息可以很简单的从底层信息计算得到
            - 每个层,块的计算完成意味着聚类完成
        2. 查询(自顶向下)

           STING算法比较容易进行数据的依赖查询操作

           我们选取一个层来进行分析，看看待查对象在该层上是否存在有关的分布，今儿不断的深入或者切换(数据相关性挖掘)
        3. 特点
            1. 易于查询关联性
            2. 聚类边界单一，不能很好的发现真实的聚类形状
    2. CLIQUE(查询中聚类算法)
        1. 思想
            - 在数据空间的子空间中查找对应的搞数据密度区域(子空间聚类方法)
            - 有稍微的降维的作用在里面
            - 密度 + 网格思想融合

2. 算法
    - 将超空间中每一维等长的分割
    - 稠密 : 如果每一个网格中的数据栈总数据的比例超过了阈值认为是稠密的网格
    - 簇 : 相互连接的稠密的网格(这个网格是针对我们的子空间来说的)最大集合构成簇
    - 在合适的子空间中发现稠密网格组成的最大连通集合，得到各各簇实现聚类
    - 聚类结果是很多的超网格(子空间)
3. 算法流程
    1. 确定含簇的最大子空间 :
        - 难点，很那发现合适的子空间(使用先验经验)
        - 使用自底向上的方式来发现子空间
    2. 确定簇
        - 构建图找簇
        - 深度有限搜搜
    3. 簇的最小描述(使用MDL):
        - 简介表达聚类的结果，提高聚类结果的可解释性
        - 凝聚网格降低描述复杂性(奥卡姆剃刀原则)
4. 特点
    - 优点
        1. 对输入数据的顺序不敏感并且可以指定数据分布
        2. 数据维度增加的可应对性
        3. 自动发现存在簇的高密度子空间
    - 缺点
        1. 精度降低