

1. Decision Tree

1.1. 决策树分为根节点(**root node**), 决策节点(**decision node/internal node**)和叶子结点(**leaf node**)

1.2. 决策树分裂的终止条件? (<https://www.jianshu.com/p/d153130b813f>)

1.2.1. 节点中所有目标变量的类别相同(all datapoints relevant to that branch have the same label, 此时**entropy=0**)。已经是相同的值了自然没有必要再分裂了，直接返回这个值就好了。

1.2.2. 树的深度达到了预先指定的最大值。

1.2.3. 不纯度的减小量小于预先定好的阈值(even with some impurity in the class), 也就是指进一步的分割数据并不能更好的降低数据不纯度的时候就可以停止树分裂了。

1.2.4. 没有特征再用来分类了(run out of attributes, **use majority label**, 回归树里用**average label**就是该叶子结点里的所有值的平均)。

1.3. 决策树剪枝:预剪枝 (**Pre-Pruning**)和后剪枝 (**Post-Pruning**)

1.3.1. 预剪枝: 其中的核心思想就是，在每一次实际对结点进行进一步划分之前先采用验证集的数据来验证如果划分是否能提高划分的准确性。如果不能，就把结点标记为叶结点并退出进一步划分；如果可以就继续递归生成节点。

1.3.2. 后剪枝:后剪枝则是先从训练集生成一颗完整的决策树，然后自底向上地对非叶结点进行考察，若将该结点对应的子树替换为叶结点能带来泛化性能提升，则将该子树替换为叶结点。

1.4. **ID3 algorithm:**

1.4.1. 思想:以信息增益来度量特征选择，选择信息增益**最大的**的特征进行分裂。算法采用**自顶向下的贪婪搜索**。

1.4.2. 步骤:

1.4.2.1. 初始化特征集合和数据集合；

1.4.2.2. 计算数据集合信息熵和所有特征的条件熵，选择信息增益**最大的特征作为当前决策节点**；

1.4.2.3. 更新数据集合和特征集合(**删除上一步使用的特征，并按照特征值来划分不同分支的数据集合**)；

1.4.2.4. 重复 2, 3 两步，若子集值包含单一特征，则为分支叶子节点。

1.4.3. 计算公式:

1.4.3.1. 熵的公式: $Entropy = - \sum_{i=1}^n p(x_i) * \log_2 p(x_i)$, 其中, $p(x_i)$ 是分是 x_i 出现的概率, n 是分类的数目。

1.4.3.2. 对于在 X 的条件下 Y 的条件熵, 是指在 X 的信息之后, Y 这个变量的不确定性
的大小, 计算公式: $Entropy(Y|X) = - \sum_{i=1}^n p(x_i) * Entropy(Y|x_i)$

1.4.3.3. 信息增益公式: $infoGain(D|A) = Entropy(D) - Entropy(D|A)$, 定义属性 A 对数
据集 D 的信息增益为 $infoGain(D|A)$, 它等于 D 本身的信息熵, 减去给定 A 的条件下 D 的条件熵。
获得信息增益最高的特征就是最好的选择。

1.4.3.4. 信息增益的意义: 引入属性 A 后, 原来数据集 D 的**不确定性减少了多少**。

1.4.4. 优点:

1.4.4.1. 计算复杂度不高;

1.4.4.2. 输出结果易于理解;

1.4.4.3. 对中间值的缺失不敏感, 可以处理不相关特征数据

1.4.5. 缺点:

1.4.5.1. 没有剪枝策略, 容易过拟合;

1.4.5.2. 信息增益准则对可取值数目较多的特征有所偏好, 类似“编号”的特征其信
息增益接近于 1;

1.4.5.3. 没有考虑缺失值。

1.5. C4.5 algorithm:

1.5.1. 相较 ID3 的改进:

1.5.1.1. 引入悲观剪枝策略进行后剪枝(参考1.2.2)。

1.5.1.2. 引入信息增益率作为划分标准。先从候选划分特征中找到信息增益高于平
均值的特征, 再从中选择增益率最高的。

1.5.1.3. 将连续特征离散化, 假设 n 个样本的连续特征 A 有 m 个取值, C4.5 将其排
序并取相邻两样本值的平均数共 $m-1$ 个划分点, 分别计算以该划分点作为
二元分类点时的信息增益, 并选择信息增益最大的点作为该连续特征的二
元离散分类点。

1.5.1.4. 对于缺失值的处理可以分为两个子问题:

- 问题一: 在特征值缺失的情况下进行划分特征的选择? (即如何计算特征的
信息增益率)

- 问题二：选定该划分特征，对于缺失该特征值的样本如何处理？（即到底把这个样本划分到哪个结点里）
- 针对问题一，C4.5 的做法是：对于具有缺失值特征用没有缺失的样本子集所占比重来折算；
- 针对问题二，C4.5 的做法是：将样本同时划分到所有子节点，不过要调整样本的权重值，其实也就是以不同概率划分到不同节点中。

1.6. 回归树(Regression Tree)处理连续值

1.6.1. 总结：回归树总体流程类似于分类树，分枝时穷举每一个特征的每一个阈值，来寻找最优切分特征和最优切分点，衡量的方法是平方误差最小化(用方差代替entropy)。分枝直达到到预设的终止条件为止。

1.6.2. 回归树的**decision node**可能是离散的值，结果一定是连续型的。

1.7. 随机森林(Random Forest)

1.7.1. 总结：随机森林是由很多决策树构成的，不同决策树之间没有关联。当我们进行分类任务时，新的输入样本进入，就让森林中的每一棵决策树分别进行判断和分类，每个决策树会得到一个自己的分类结果，决策树的分类结果中哪一个分类最多，那么随机森林就会把这个结果当做最终的结果。

1.7.2. 优点：

- 1.7.2.1. 可以出来很高维度（特征很多）的数据，并且不用降维，无需做特征选择
- 1.7.2.2. 可以判断特征的重要程度
- 1.7.2.3. 可以判断出不同特征之间的相互影响
- 1.7.2.4. 不容易过拟合
- 1.7.2.5. 训练速度比较快，容易做成并行方法
- 1.7.2.6. 对于不平衡的数据集来说，它可以平衡误差。
- 1.7.2.7. 如果有很大一部分的特征遗失，仍可以维持准确度。

1.7.3. 缺点：

- 1.7.3.1. 取样(sampling)的个数越多(及分成更多的可重复数据子集训练更多的决策树)，准确度越高但是效率越低。
- 1.7.3.2. 在某些噪音较大的分类或回归问题上会过拟合。
- 1.7.3.3. 对于有不同取值的属性的数据，取值划分较多的属性会对随机森林产生更大的影响，所以随机森林在这种数据上产出的属性权值是不可信的。

2. Clustering

2.1. K-means algorithm

2.1.1. 步骤:

2.1.1.1. 选择初始化的 k 个样本作为初始聚类中心 $a=a_1, a_2, \dots, a_k$;

2.1.1.2. 针对数据集中每个样本 x_i 计算它到 k 个聚类中心的距离并将其分到距离最小的聚类中心所对应的类中;

2.1.1.3. 针对每个类别 a_j , 重新计算它的聚类中心 $a_j = \frac{1}{|c_i|} \sum_{x \in c_i} x$ (即属于该类的所有样本的质心);

2.1.1.4. 重复上面两步操作, 直到达某中止条件(迭代次数、最小误差变化等)。

2.1.2. 复杂度:

2.1.2.1. 时间复杂度: $O(tknm)$, t 为迭代次数, k 为簇的数目, n 为样本点数, m 为样本点维度。

2.1.2.2. 空间复杂度: $O(m(n+k))$, k 为簇的数目, m 为样本点维度, n 为样本点数。

2.1.3. 优点:

2.1.3.1. 容易理解, 聚类效果不错, 虽然是局部最优, 但往往局部最优就够了;

2.1.3.2. 处理大数据集的时候, 该算法可以保证较好的伸缩性;

2.1.3.3. 算法复杂度低。

2.1.4. 缺点:

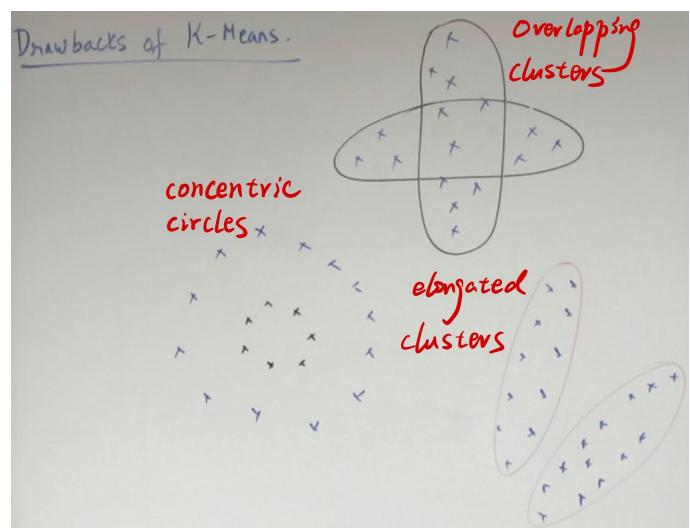
2.1.4.1. **K** 值需要人为设定, 不同 **K** 值得到的结果不一样;

2.1.4.2. 对初始的簇中心敏感, 不同选取方式会得到不同结果;

2.1.4.3. 对异常值敏感;

2.1.4.4. 样本只能归为一类, 不适合多分类任务(一个样本一定只会被分给一个簇 (**hard membership**), 但实际上有可能有两个簇 **overlap** 的情况(**需要 soft membership algorithm**));

2.1.4.5. 不适合太离散的分类、样本类别不平衡的分类、非凸形状的分类。



2.2. GMM algorithm(高斯混合模型)

2.2.1. 概述:有时候数据的分布用单高斯分布不能很好的表示,因此引入GMM。它的本质就是**融合几个单高斯模型**,来使得模型更加复杂,从而产生更复杂的样本。理论上,如果某个混合高斯模型融合的高斯模型个数足够多,它们之间的权重设定得足够合理,这个混合模型可以拟合任意分布的样本。

2.2.2. 多维高斯分布的协方差矩阵 Σ 的性质:

2.2.2.1. Σ 是对称矩阵(**symmetric matrix**)。

2.2.2.2. Σ 是满秩(**non-singular**)的,也就是 $|\Sigma|$ (行列式)不为0。

2.2.2.3. Σ 是半正定的(**positive semidefinite**),所以 $|\Sigma|$ 大于0。

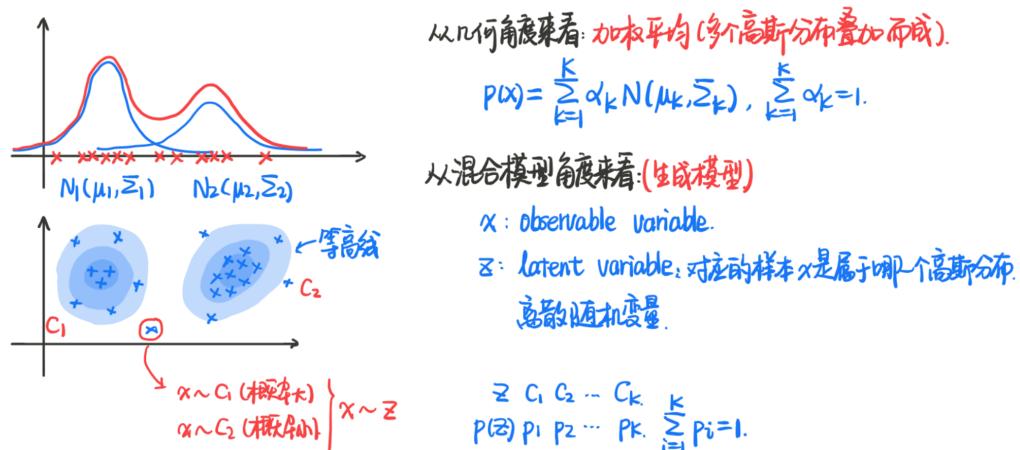
2.2.2.4. Σ 所有的特征值是大于0的**实数**。

2.2.2.5. Σ 的逆矩阵存在。

2.2.2.6. Σ 可相似对角化(**similar diagonalizable**)。

2.2.2.7. Σ **主对角线元素均非负**,其他位置元素可以有负数。

2.2.3. 基本公式及符号:



2.2.4. 用极大似然估计(MLE)求解GMM(无解):

高斯分布(极大似然).

$$\begin{aligned} p(x) &= \sum_z p(x, z) = \sum_{k=1}^K p(x, z=c_k) = \sum_{k=1}^K p(z=c_k) \cdot p(x|z=c_k) \\ &= \sum_{k=1}^K p_k \cdot N(x|\mu_k, \Sigma_k). \end{aligned}$$

概率值.

X : observed data $X = (x_1, x_2, \dots, x_N)$.

(X, Z) : complete data $(X, Z) = \{(x_1, z_1), (x_2, z_2), \dots, (x_N, z_N)\}$.

$$\begin{aligned}
 \text{MLE: } \hat{\theta}_{\text{MLE}} &= \underset{\theta}{\operatorname{argmax}} \log P(\mathbf{x}) = \underset{\theta}{\operatorname{argmax}} \log \prod_{i=1}^N P(x_i) = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^N \log P(x_i) \\
 &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^N \log \sum_{k=1}^K p_k \cdot N(x_i | \mu_k, \Sigma_k) \quad \text{单-Gaussian时,} \\
 &\quad \text{无法求出解析解.} \quad \log P(x_i) = \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)
 \end{aligned}$$

直接用MLE求解Gauss, 无法得出解析解.

2.2.5. EM算法估计GMM参数:

2.2.5.1. 初始化: 随机赋值 μ_0, Σ_0, π_0 (这里 π 表示加权平均的权值)

2.2.5.2. E-step: 根据当前的 μ_k, Σ_k, π_k 计算后验概率 $\gamma(z_{nk})$, 其中 γ_{ij} 表示第 i 个 datapoint 是由第 j 个单高斯分布生成的概率。

2.2.5.3. M-step: 根据 E-step 中的 $\gamma(z_{nk})$ 重新计算新的 $\mu_{k+1}, \Sigma_{k+1}, \pi_{k+1}$

$$\begin{aligned}
 \boldsymbol{\mu}_k^{new} &= \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n \\
 \boldsymbol{\Sigma}_k^{new} &= \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{new})(\mathbf{x}_n - \boldsymbol{\mu}_k^{new})^T \\
 \pi_k^{new} &= \frac{N_k}{N} \quad , \text{ 其中}
 \end{aligned}$$

$$N_k = \sum_{n=1}^N \gamma(z_{nk})$$

2.2.5.4. 计算对数似然函数 $\ln p(\mathbf{x} | \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$, 检查结果是否收敛, 如果没有重复 E-M 步骤。

2.2.6. K-means 和 GMM 都是给定了划分的簇的个数的, 没有 optimal number of cluster。

3. Dimensionality Reduction(属于 Unsupervised Learning)

3.1. 降维的目的:

3.1.1. 直观地好处是维度降低了, 便于计算和可视化, 其更深层次的意义在于有效信息的提取综合及无用信息的摈弃。

3.2. PCA

3.2.1. 步骤(假设有 m 条 n 维数据, 需要降维为 k 维):

3.2.1.1. 将原始数据按列组成 n 行 m 列矩阵 X ;

3.2.1.2. 将 X 的每一行进行零均值化, 即减去这一行的均值;

3.2.1.3. 求出协方差矩阵 $C = \frac{1}{m} XX^T$;

3.2.1.4. 求出协方差矩阵的特征值及对应的特征向量;

3.2.1.5. 将特征向量按对应特征值大小从上到下按行排列成矩阵, 取前 k 行组成矩阵 P (此 P 即为我们需要的基变换矩阵)

3.2.1.6. $Y = PX$ 即为降维到 k 维后的数据。

3.2.2. 优点:

3.2.2.1. 缓解维度灾难:PCA 算法通过舍去一部分信息之后能使得样本的采样密度增大(因为维数降低了), 这是缓解维度灾难的重要手段;

3.2.2.2. 降噪:当数据受到噪声影响时, 最小特征值对应的特征向量往往与噪声有关, 将它们舍弃能在一定程度上起到降噪的效果;

3.2.2.3. 特征独立:PCA 不仅将数据压缩到低维, 它也使得降维之后的数据各特征相互独立;

3.2.3. 缺点:

3.2.3.1. 过拟合:PCA 保留了主要信息, 但这个主要信息只是针对训练集的, 而且这个主要信息未必是重要信息。有可能舍弃了一些看似无用的信息, 但是这些看似无用的信息恰好是重要信息, 只是在训练集上没有很大的表现, 所以 PCA 也可能加剧了过拟合;

3.3. MDS(参考:<https://zhuanlan.zhihu.com/p/433336369>)

3.3.1. 概述:PCA 只考虑了数据整体的分布情况(方差), 没有考虑点和点之间的关系, MDS 在降维的同时尽可能保持点和点的距离不变(欧氏距离), 即

$\min \sum_{i,j} (\|z_i - z_j\| - d_{ij})^2$ 。其中 d_{ij} 是原始样本 i 和原始样本 j 之间的距离, 它的度量选取可以比较随意, 只要满足 $d_{ii} = 0$, $d_{ij} = d_{ji}$, $d_{ik} + d_{jk} \geq d_{ij}$ 。这个优化问题的解不唯一(原因去看参考链接)。

3.3.2. 步骤:

3.3.2.1. 我们要寻找的是降维后的数据阵 Z , 令 $B = Z^T Z \in R^{m*m}$, 求出 B 就能求出 Z ;

3.3.2.2. 先求解出 B , 然后利用特征值分解(注意到 B 是实对称的, 特征分解一定成立) $B = U \Lambda U^T = (\Lambda^{1/2} U^T)^T (\Lambda^{1/2} U^T) = Z^T Z$, 此处 $Z = \Lambda^{1/2} U^T$, 其中 Λ 是将特征值从大到小依次排列得到的对角阵, U 是这些特征值对应的特征向量构成的矩阵。

3.3.2.3. 令 $\|z_i - z_j\| = d_{ij}$ (这时候肯定最小)。可以得出以下四个等式:

$$\left\{ \begin{array}{l} d_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij} \\ \sum_{i=1}^m d_{ij}^2 = \text{tr}(B) + mb_{jj} \\ \sum_{j=1}^m d_{ij}^2 = \text{tr}(B) + mb_{ii} \\ \sum_{i=1}^m \sum_{j=1}^m d_{ij}^2 = 2m\text{tr}(B) \end{array} \right. , \text{ 其中 } b_{ij} \text{ 表示 } B \text{ 矩阵中的 } i, j \text{ 这个元素, } \text{tr}(B) \text{ 是 } B \text{ 的迹。}$$

四个未知数四个等式, 求解这个方程组就能得到B矩阵, 进而求出Z。

3.3.3. 优点:

3.3.3.1. 不需要先验知识, 计算简单;

3.3.3.2. 保留了数据在原始空间的相对关系, 可视化效果比较好。

3.3.4. 缺点:

3.3.4.1. 各个维度的地位相同, 无法区分不同维度的重要性。

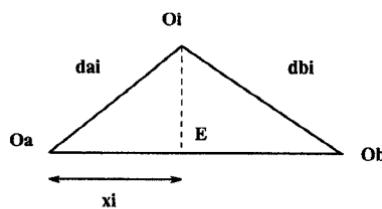
3.4. FastMap

3.4.1. 概述: 利用启发式思想加速MDS到线性复杂度(MDS是 n^2)。

3.4.2. 步骤:

3.4.2.1. 随机选择一个点 \mathbf{O}_b , 找到离它最远的点 \mathbf{O}_a , 然后再找到离 \mathbf{O}_a 最远的点代替 \mathbf{O}_b 。他们就是标志点(pivot points)。我们假设它们就是当前所有点里距离最大的两个点(启发式)。**之所以这么做是因为在3.4.2.2中我们要确保每个 \mathbf{O}_i 在 \mathbf{ab} 上的映射都落在 \mathbf{ab} 线段内。**

3.4.2.2. 对于其余每个点 \mathbf{O}_i , 求出 x_i (余弦定理)。



3.4.2.3. 更新各点对 $(\mathbf{O}_i, \mathbf{O}_j)$ 的距离 $(D'(\mathbf{O}_i, \mathbf{O}_j'))^2 = (D(\mathbf{O}_i, \mathbf{O}_j))^2 - (x_i - x_j)^2$, **此时维度已经比之前减少了1。**

3.4.2.4. 重复3.4.2.1~3.4.2.3直至维度减小了k维。

3.4.3. 优点:

3.4.3.1. 速度快, 复杂度为 $O(nk)$ 。分析: 3.4.2.1的复杂度为 $O(n)$, 3.4.2.2复杂度也是 $O(n)$, 3.4.2.3复杂度还是 $O(n)$ 因为我们已经求出了 x_i 。整个过程重复 k 次。总复杂度就为 $O(nk)$ 。

3.4.3.2. K是用户决定的, 可以根据不同任务对精度的需求改变(K is user-specified, representing a tradeoff between memory and accuracy)。

3.4.3.3. 可以通过画出每次迭代的时候的失真图来确定k在取什么值的时候效果最好(Optimal value of K can be determined externally by plotting average distortion vs K and identifying the point of diminishing returns)。

3.4.4. 缺点:结果不一定最优。很明显在3.4.2.1中不是每次都能保证ab是整个数据集里最远的两个点,因为我们是随机取的Q。

3.4.5. 应用:

3.4.5.1. **FastMap for Shortest Path Computations:** 目的是设计一个算法计算最短路比Dijkstra更快。方法是修改FastMap中计算距离的公式使其维护的 d_{ij} 是两个点的最短路,这里用的是Dijkstra构造最短路径树的想法,所以复杂度是 $O(|E|+|V|\log|V|)$,称为近线性(near-linear time)。这样总体模型的输入是一个无向图,输出是该图在欧几里得空间里的映射(Euclidean embedding)。其中点到点的距离就是最短路。

- 亮点:

- 用欧氏距离替换了曼哈顿距离,因为曼哈顿距离通用性不强(is not defined for general graphs)且在A*算法中的效果很差;
- 通过修改距离公式和权值更新公式保证了启发式的一致性(Consistency)与可接受性(Admissibility)详见下面伪代码;
- 速度比Dijkstra快很多因为不用遍历每个点,并且可以用于动态图(dynamic graphs)
- 将带障碍的无向图映射到欧几里得空间以后不需要考虑障碍了,减少了复杂性。

- 缺点:不能保证最优性(毕竟用的是启发式)。

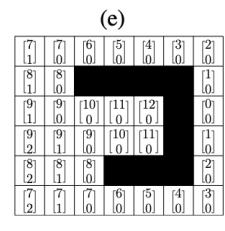
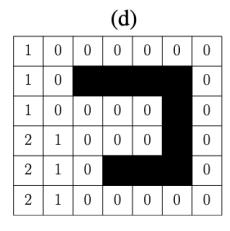
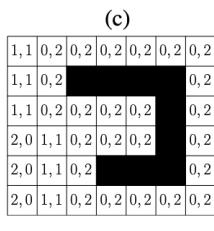
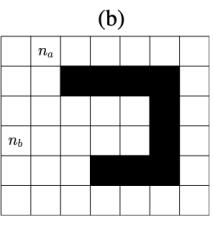
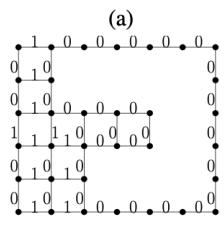
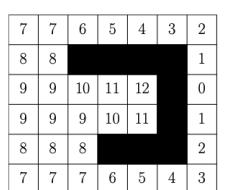
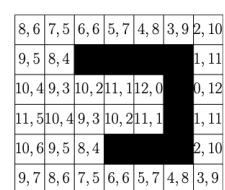
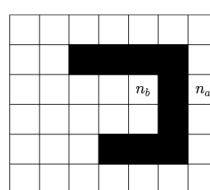
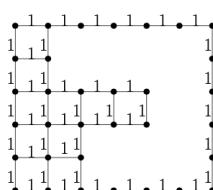
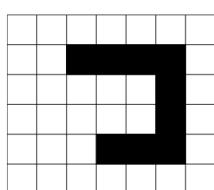
```

for each  $v \in V$  do
     $[p_v]_K = (d_{av} + d_{ab} - d_{vb})/2$ 
for each edge  $(u, v) \in E$  do
     $w'(u, v) = w'(u, v) - |[p_u]_K - [p_v]_K|$ ;

```

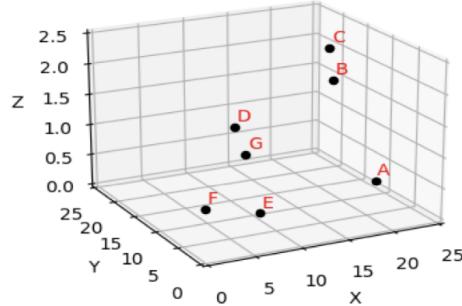
- 算法重点伪代码 $K = K + 1; K_{max} = K_{max} - 1;$

- 例子(2D-Embedding):



3.4.5.2. **FastMap for Combinatorial Problems on Graphs**: 相当于是一个上面那个应用的推广，推广到其他和图相关的问题例如mult-agent meeting problem。总体思路就是用**FastMap**将原问题映射到欧几里得空间里，然后用欧氏距离替换掉原本的距离，在损失一定精度的前提下大大降低复杂度。

Euclidean Embedding



- 欧几里得映射：

3.4.5.3. **Efficiently Computing Measures of Centrality**: 问题是在一个非常庞大的关系网络图里找出前k个最重要的一个点(例如找一个人，他和其他所有人的距离的和最小)。老样子，适当更改**FastMap**离计算距离的公式，将原问映射到欧几里得空间，用**极少的正确率换取极快的速度**。

3.4.5.4. **Embedding Directed Graphs in Potential Fields**: 欧几里得空间里不能表示有向边，所以FastMap不能直接用在有向图里。这里假设有向图是强连通的，那么方法就是将前k-1维距离表示为无向边形式，其大小为 $(d_i + d_{ji})/2$ 即平均值，然后对前k-1维使用FastMap映射到欧几里得空间，对于它们实际上的距离差，用一个因素(factor) ψ 通过机器学习推测出。即 $P_{k+1} = \psi(P_1, \dots, P_k)$ 。

3.4.5.5. **Community Detection and Block Modeling**: 问题是在一个非常庞大的关系网络图中得想办法分成若干个簇。方法：In near-linear time, convert the problem to a regular GMM clustering problem in Euclidean space.

4. Classification & Regression

4.1. Perceptron Learning Algorithm

4.1.1. 目的：将数据进行**二分类**。

4.1.2. 前提：数据**线性可分**。

4.1.3. 定义：

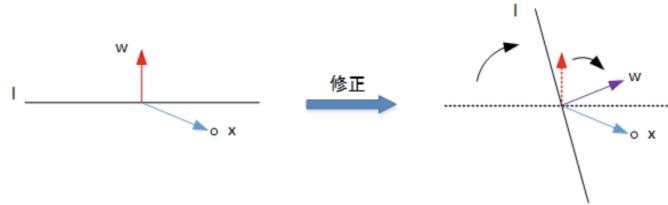
4.1.3.1. 目标函数： $\sum_{i=1}^N w_i x_i + b \Rightarrow \sum_{i=1}^{N+1} w_i x_i$ (这里把b看作 $w_0 x_0$ ，其中 $x_0=1$)

4.1.3.2. 损失函数：

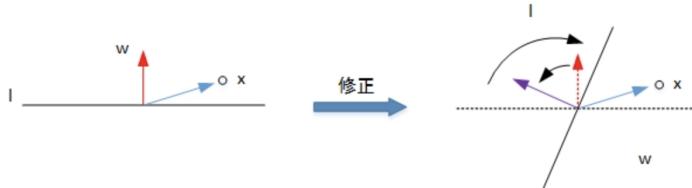
- 通俗解释：误分类点到超平面S的总距离
- 推导过程：一个点被误分类有两种情况①当 $y_i=1$ 时应该是正样本，但是 $wx_i+b<0$; ②当 $y_i=-1$ 时应该是负样本，但是 $wx_i+b>0$ 。合并两种情况得到误分类的点到S的距离为 $-\frac{y_i^*(w^*x_i+b)}{\|w\|}$ 。由于 $\|w\|$ 不仅不影响w的方向也不影响正负值的判断，所以不关注它，最终损失函数为 $\sum_{x_i \in M} y_i^* (w^* x_i + b)$

4.1.4. 步骤：

- 随机选择一个初始权重 w_0 。
- 统计根据这个权重得到的超平面S进行分类后被误分类的点。
- 随机选择一个点进行修正，如果这个点是①的情况，修改 $w'=w+\alpha x_i$ ，几何意义就是原来 $wx_i < 0$, w 与 x_i 的夹角 $> 90^\circ$ 。修正就是让夹角变小，如下图：



如果这个点是②的情况，修改 $w'=w-\alpha x_i$ ，几何意义就是原来 $wx_i > 0$ ，修正要让夹角变大，如下图：



- 重复4.1.4.2~4.1.4.3直至没有被误分类的点，如果死循环了说明数据线性不可分。(每次循环被误分类的点可能增多或减少，但最终会收敛)

4.1.5. 优点：简单粗暴

4.1.6. 缺点：

- 无法处理非线性数据分类。
- 迭代次数受结果超平面以及训练集的数据影响很大，可能导致复杂度很高。

4.2. Pocket PLA

4.2.1. 目的: 针对PLA无法处理非线性二分类的情况进行优化。

4.2.2. 方法: 增加一个容忍阈值, 如果更新w以后的超平面的误分类点数比更新前要少, 就更新最优解, 否则不更新。相当于在有限的迭代次数里尽可能找到误分类点最少的超平面但是又不一定要求没有误分类点。

4.3. Linear Regression

4.3.1. 目的: 用一个线性超平面尽可能拟合数据分布。

4.3.2. 方法: 定义一个超平面 $f(w) = w^T x + b$, 通过给定的 x 和 y 用 **最小二乘法(Ordinary Least Squares)** 求解 w 。大致过程就是定义一个损失函数 $L(w)$ 表示为 $f(w)$ 预测出的 $w^T x_i$ 和真实值 y_i 的差值的平方的和。现在要使 $L(w)$ 最小, 就是对 w 求导取 0 得到极值。

最小二乘法.

$$\begin{aligned} L(w) &= \sum_{i=1}^N \|w^T x_i - y_i\|^2 = \sum_{i=1}^N (w^T x_i - y_i)^2 \\ &= (\underbrace{w^T x_1 - y_1}_{\parallel}, \underbrace{w^T x_2 - y_2}_{\parallel}, \dots, \underbrace{w^T x_N - y_N}_{\parallel}) \begin{pmatrix} w^T x_1 - y_1 \\ w^T x_2 - y_2 \\ \vdots \\ w^T x_N - y_N \end{pmatrix} = Xw - Y \\ &= w^T (x_1 x_2 \dots x_N) - (y_1 y_2 \dots y_N) \\ &= w^T X^T - Y^T \end{aligned}$$

矩阵等式!

$$\begin{aligned} L(w) &= (w^T X^T - Y^T)(Xw - Y) \\ &= w^T X^T X w - Y^T X w - W^T X^T Y + Y^T Y \\ &= w^T X^T X w - 2w^T X^T Y + Y^T Y \end{aligned}$$

$$\begin{array}{l} X \in \mathbb{R}^{N \times P} \\ Y \in \mathbb{R}^{N \times 1} \\ W \in \mathbb{R}^{P \times 1} \end{array}$$

$$\hat{W} = \arg \min L(w)$$

X^T (伪逆)

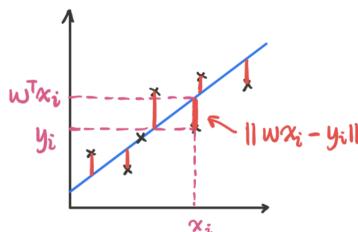
$$\frac{\partial L(w)}{\partial w} = 2X^T X w - 2X^T Y \triangleq 0 \Rightarrow X^T X w = X^T Y \Rightarrow W = (X^T X)^{-1} X^T Y$$

4.3.3. 几何解释:

4.3.3.1. 最直观的, 红色距离的平方求和。

几何解释1:

误差与所有红色距离有关。

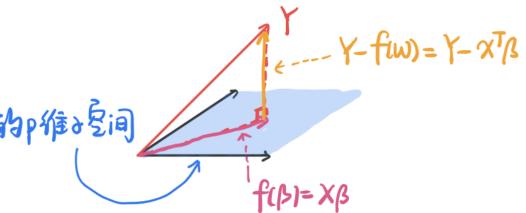


4.3.3.2. 可以把X矩阵想像成p个N维向量，那么X就构成了一个p维空间，Y不在p维空间里(除非所有的y都完美被f(w)拟合了)。那么问题就变成了找向量Y到p维空间最近的点，也就是Y在p维空间里的投影。

几何解释2：

$$f(w) = \frac{X \cdot W}{N \times p} = X \cdot \beta.$$

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Np} \end{bmatrix}_{N \times p}$$



在p维空间中找一向量 $f(\beta)$ 使得 $f(\beta)$ 与 Y 的距离最小。

$f(\beta)$ 是 Y 在p维空间的投影

则满足 $Y - f(w)$ 与 p维空间的基向量垂直

$$\text{即 } X^T(Y - f(w)) = 0$$

$$X^T(Y - X\beta) = 0$$

$$X^T Y = X^T X \beta$$

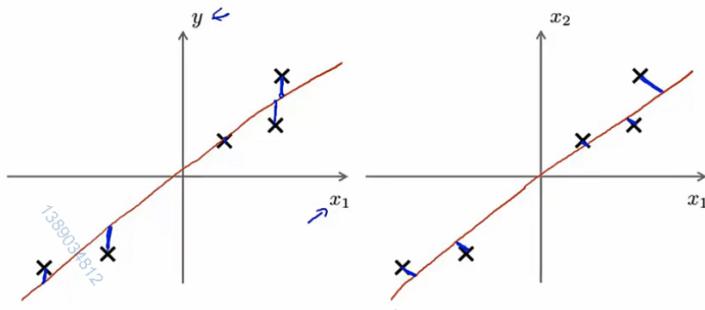
$$\beta = (X^T X)^{-1} X^T Y$$

4.3.4. PCA和Linear Regression的区别：

4.3.4.1. PCA是无监督学习(unsupervised learning)，回归是监督学习。

4.3.4.2. LR中有自变量(independent)和因变量(dependent)的区分，PCA中所有变量是一视同仁的。

4.3.4.3. LR中最小化的是点到直线的垂直距离的平方，PCA中最小化的是点到直线的最短的正交距离，如下图左边是LR右边是PCA。



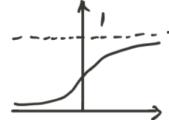
4.4. Logistic Regression

4.4.1. 目的: 虽然名字是回归, 但其实做的是分类问题。

4.4.2. 方法: 引入**sigmoid函数**, 把所有的输入映射到(0,1)区间内表示概率, 用**极大似然估计(MLE)**作为损失函数, 目标是让MLE最大化。

Data: $\{(x_i, y_i)\}_{i=1}^N, x_i \in \mathbb{R}^P, y_i \in [0,1]$

sigmoid function: $\sigma(z) = \frac{1}{1 + e^{-z}}$ $\begin{cases} z \rightarrow +\infty, \lim \sigma(z) = 1 \\ z \rightarrow -\infty, \lim \sigma(z) = 0. \end{cases}$



$$p_1 = P(y=1|x) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}, \quad y=1 \quad \varphi(x_i; w)$$

$$p_0 = P(y=0|x) = 1 - P(y=1|x) = \frac{e^{-w^T x}}{1 + e^{-w^T x}}, \quad y=0 \quad 1 - \varphi(x_i; w).$$

$$\text{MLE: } \hat{w} = \underset{w}{\operatorname{argmax}} \log P(Y|x) = \underset{w}{\operatorname{argmax}} \log \prod_{i=1}^N P(y_i|x_i) = \underset{w}{\operatorname{argmax}} \sum_{i=1}^N \log P(y_i|x_i)$$

$$= \underset{w}{\operatorname{argmax}} \sum_{i=1}^N (y_i \log p_1 + (1-y_i) \log p_0) = \underset{w}{\operatorname{argmax}} \sum_{i=1}^N y_i \log \varphi(x_i; w)$$

$$+ (1-y_i) \log (1 - \varphi(x_i; w))$$

- cross entropy

MLE(max) \Rightarrow loss function (min cross entropy)

4.4.3. 优点: 输出值落在(0,1)之间, 并且有概率意义。

4.4.4. 缺点: 模型简单容易欠拟合(**underfitting**), 精度不高。

5. Neural Network(<https://www.youtube.com/watch?v=WPYNpUw6MhE>)

5.1. 方向导数(directional derivative)和梯度(gradient):

偏导数的含义是某个点在x正方向上的变化率, 方向导数的含义是某个点在任意方向上的变化率。梯度的含义是某个点沿着哪个方向的变化率最大, 它的方向与取得最大方向导数的方向一致, 而它的模为方向导数的最大值

5.2. 目的: 解决生活中的非线性问题。

5.3. 方法:

在**前向传播(forward-propagation)**时引入了**激活函数(activation function)**给神经元增加了非线性, 通过**后向传播(backward-propagation)**从错误中学习通过**梯度下降(gradient descent)**更新权值。

5.4. 梯度下降法:

5.4.1. 方法:每次迭代通过求当前点的最大梯度,即变化最快的方向,往该方向变化的性价比是最高的,移动的步长就是向量乘学习率。直到某个点的梯度变化小于阈值。批量梯度下降法(**Batch Gradient Descent**)用所有样本的均值来更新梯度;随机梯度下降法(**Stochastic Gradient Descent**)随机选择一个样本来更新梯度。

5.4.2. 随机梯度下降法的优点:

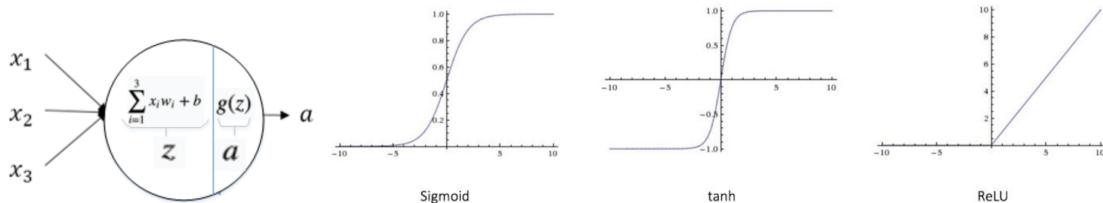
5.4.2.1. 计算简单(computationally easy)

5.4.2.2. 随机在大多数情况是好的(randomness is good in general)

5.4.2.3. 经验常量容易被确定(empirical constants are easier to determine)

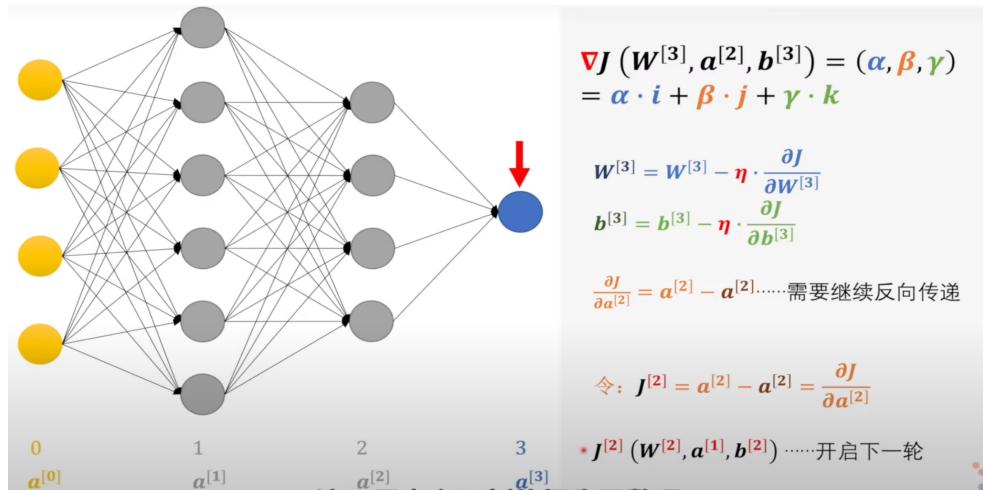
5.5. 神经网络里的感知机(**interconnected perceptrons**)

5.5.1. 单个感知机:神经网络里单个神经元由求和部分,偏置和激活函数组成,如下图所示:偏置就是 $y=wx+b$ 里的b,而激活函数主要有**Sigmoid**, **tanh**, **ReLU**。



5.5.2. 多个感知机:多个单独的感知机之间可以用逻辑运算相连(如xor)从而增加了整个网络可以表达更复杂的运算。

5.6. BPNN步骤(<https://www.zhihu.com/question/27239198>)



5.6.1. 前向传播：当前层的感知机以前面一层的输出作为输入，经过自己的运算以后又把自己的输出传给下一层感知机们作为输入。最后在输出层得到一个值。

5.6.2. 后向传播：得到输出层的值之后，通过损失函数 $J(w, a, b)$ 计算它和预期值之间的差距，然后更新权重 w 和偏置 b 。方法就是随机梯度下降，在损失函数下降最快的方向上移动学习率 $\eta * \text{偏导数}$ 的步长。至于 a 是上一层的输出，所以把对 a 求偏导的结果传给上一层作为它的损失值。重复这个过程直到输入层，输入层的 a 是常量所以只需要更新 w 和 b 。这样就完成了一轮迭代。(注意中间层的反向传播可能来自多个感知机，所以它的损失函数应该是所有和它相连的后一层的对 a 偏导的和求平均)

5.6.3. 重复5.6.1~5.6.2直至输出层的值和期望值的误差小于设定阈值。

6. Support Vector Machine(SVM)

6.1. 目的：在分类问题中找出一个最大间隔超平面(Hyperplane separation theorem)。

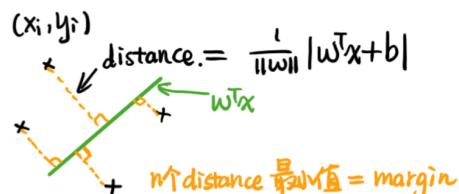
6.2. 硬间隔(Hard-margin)SVM

6.2.1. 将分类问题转化为优化问题：转化以后成为了一个二阶凸优化问题(Quadratic Programming)，其特点是一定存在唯一的一个极值(对后面求偏导特别重要)。

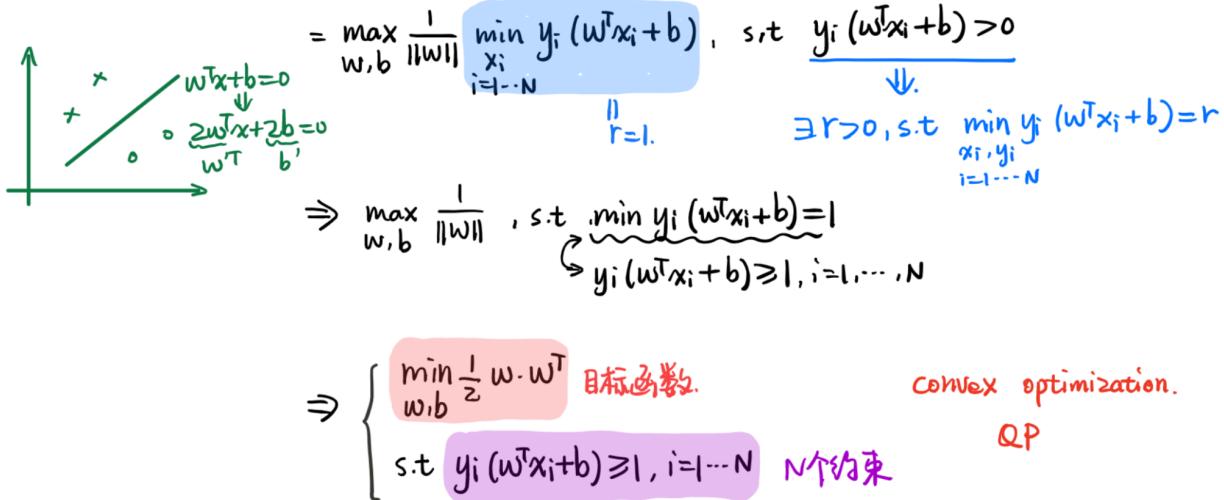
最大间隔分类器 模本: $\{(x_i, y_i)\}_{i=1}^N \quad x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}$

$$\begin{aligned} & \max \text{margin}(w, b) \\ \text{s.t. } & \begin{cases} w^T x_i + b > 0, & y_i = +1 \\ w^T x_i + b < 0, & y_i = -1 \end{cases} \\ & y_i (w^T x_i + b) > 0 \quad \text{for } \forall i = 1, \dots, N \end{aligned}$$

$$\begin{aligned} \text{margin}(w, b) &= \min_{w, b} \text{distance}(w, b, x_i) \\ &\quad x_i, i=1 \dots N \\ &= \min_{w, b} \frac{1}{\|w\|} |w^T x_i + b| \\ &\quad x_i, i=1 \dots N \end{aligned}$$



$$\begin{aligned} \Rightarrow \max \text{margin}(w, b) &= \max_{w, b} \min_{x_i} \frac{1}{\|w\|} |w^T x_i + b|, \quad \text{s.t. } y_i (w^T x_i + b) > 0 \\ &= \max_{w, b} \min_{x_i} \frac{1}{\|w\|} y_i (w^T x_i + b), \quad \text{s.t. } y_i (w^T x_i + b) > 0 \end{aligned}$$



6.2.2. 将优化问题转化为无约束对偶问题：用拉格朗日乘子法(Lagrange multiplier)将原问题转化为对偶问题。由于SVM的优化问题是一个二阶凸优化问题(Quadratic Programming), 所以其一定是强对偶关系。

P2 Hard-Margin SVM

$$\text{Data} = \{(x_i, y_i)\}_{i=1}^N, x_i \in \mathbb{R}^p, y_i \in \{+1, -1\}$$

$$\begin{cases} \min_{w,b} \frac{1}{2} w^T w \\ \text{s.t. } y_i (w^T x_i + b) \geq 1 \quad i=1, \dots, N \end{cases} \rightarrow \text{primal problem.}$$

$$L(w, b, \lambda) = \frac{1}{2} w^T w + \sum_{i=1}^N \lambda_i (1 - y_i (w^T x_i + b))$$

$$\begin{cases} \min_{w,b} \max_{\lambda} L(w, b, \lambda) \\ \text{s.t. } \lambda_i \geq 0. \end{cases}$$

$$\begin{cases} \max_{\lambda} \min_{w,b} L(w, b, \lambda) \\ \text{s.t. } \lambda_i \geq 0. \end{cases}$$

$$\begin{cases} \max_{\lambda} -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j x_i^T x_j + \sum_{i=1}^N \lambda_i \\ \text{s.t. } \lambda_i \geq 0, \sum_{i=1}^N \lambda_i y_i = 0. \end{cases}$$

$$\left. \begin{array}{l} (x_i, y_i) \\ \text{①若 } 1 - y_i (w^T x_i + b) > 0 \\ \max_{\lambda} L(\lambda, w, b) = \frac{1}{2} w^T w + \infty = \infty \\ \text{②若 } 1 - y_i (w^T x_i + b) \leq 0 \\ \max_{\lambda} L(\lambda, w, b) = \frac{1}{2} w^T w + 0 = \frac{1}{2} w^T w \\ \min_{w,b} \max_{\lambda} L = \min_{w,b} \frac{1}{2} w^T w \\ \text{③} \rightarrow \min_{w,b} \max_{\lambda} L = \min_{w,b} (\infty, \frac{1}{2} w^T w) \\ = \min_{w,b} \frac{1}{2} w^T w \end{array} \right\}$$

$$\min_{w,b} \max_{\lambda} L \geq \max_{w,b} \min_{\lambda} L \text{ 弱对偶关系.}$$

$$\min_{w,b} \max_{\lambda} L = \max_{w,b} \min_{\lambda} L \text{ 强对偶关系}$$

6.2.3. 利用KKT条件的slackness complementary确定SVM的支持向量:由slackness complementary可知对于每一个点 (x_i, y_i) , 要么 $\lambda_i=0$ 要么 $1-y_i(w^T x_i + b)=0$ 。对于支持向量, 也就是 $w^T x_i + b = \pm 1$ 的点, $1-y_i(w^T x_i + b)=0$, 所以 λ_i 可以不等于0; 对于非支持向量的点, $w^T x_i + b \neq \pm 1$, λ_i 必须等于0。所以所有不是支持向量的点都有 $\lambda_i=0$, 只需要找那些 $\lambda_i \neq 0$ 的点, 它们就是支持向量。

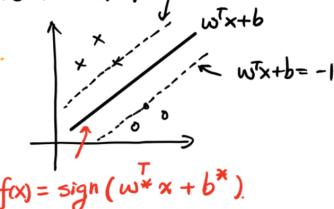
P3 KKT条件:

$$\left\{ \begin{array}{l} \frac{\partial L}{\partial w} = 0, \frac{\partial L}{\partial b} = 0, \frac{\partial L}{\partial \lambda} = 0 \\ \lambda_i (1 - y_i (w^T x_i + b)) = 0 \\ \lambda_i \geq 0 \\ 1 - y_i (w^T x_i + b) \leq 0 \end{array} \right.$$

\$\Rightarrow\$ slackness complementary.

原问题与对偶问题是强对偶关系

\$\Leftrightarrow\$ 满足KKT条件



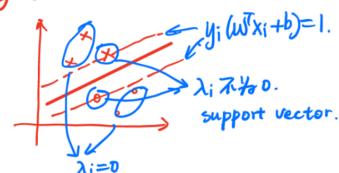
$$f(x) = \text{sign}(w^T x + b^*)$$

$$\exists (x_k, y_k) \text{ s.t. } 1 - y_k (w^T x_k + b) = 0$$

$$y_k (w^T x_k + b) = 1$$

$$y_k^2 (w^T x_k + b) = y_k.$$

$$b^* = y_k - w^T x_k = y_k - \sum_{i=0}^N \lambda_i y_i x_i^T x_k$$



6.3. 核方法(Kernel Method)

6.3.1. 基础: 基于定理Cover's Theorem: 高维空间比低位空间更容易线性可分。

6.3.2. 方法: 给原本non-linear的空间进行非线性转换 $\phi(x)$ 升维, 在新的空间里就很容易线性可分了。以硬间隔SVM为例:

Hard-Margin SVM

$$\text{Primal Problem: } \min_{w, b} \frac{1}{2} \|w\|^2$$

$$\text{Dual Problem: } \min \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j \underbrace{x_i^T x_j}_{\phi(x_i^T) \cdot \phi(x_j)} - \sum_{i=1}^N \lambda_i$$

现在的问题就变成了如何求 $\phi(x_i^T) \cdot \phi(x_j)$ (这里表示 $\phi(x_i^T)$ 和 $\phi(x_j)$ 的内积。由于 $\phi(x)$ 维度可能非常高, 所以求解 $\phi(x)$ 几乎是不可能的。

因此我们希望能跳过求 $\phi(x)$ 直接定义一个函数求 $\phi(x_i^T) \cdot \phi(x_j)$, 这个函数称为核函数(kernel function)。

Kernel Function:

$$K(x, x') = \phi(x)^T \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle$$

$$\forall x, x' \in X, \exists \phi: x \mapsto z, \text{ s.t. } K(x, x') = \phi(x)^T \cdot \phi(x')$$

则称 $K(x, x')$ 是一个核函数。

6.3.3. 参考(包括常用的核函数)

https://chen-feiyang.github.io/posts/%E6%A0%B8%E6%96%B9%E6%B3%95kernel_method/

6.4. 软间隔(Soft-margin)SVM

6.4.1. 特点: 允许一点错误, 不一定要完全把所有数据分隔开(处理噪声的情况)。

6.4.2. 方法: 添加一个 loss 表示容忍度, loss 的具体表示为下图所示, 其余的计算方法和

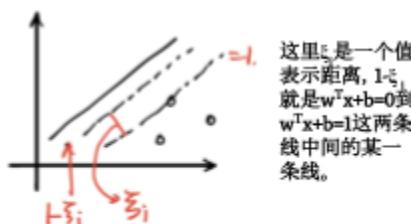
硬间隔一模一样, $\sum_{i=1}^N \xi_i$ 也称作 **slack**。

$$\begin{cases} \min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^N \max\{0, 1 - y_i(w^T x_i + b)\} \\ \text{s.t. } y_i(w^T x_i + b) \geq 1, i=1 \dots N \\ \xi_i = 1 - y_i(w^T x_i + b), \xi_i \geq 0 \end{cases}$$

相当于我们添加了一个 loss, C 是人为设定的一个常数

$$\begin{cases} \min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^N \xi_i \\ \text{s.t. } y_i(w^T x_i + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases}$$

最终形式



值得注意的是, 在这里 $\xi_i > 0$ 不一定 support vector 了。 $0 < \xi_i < C$, 此时第 i 个样本是 support vector; $\xi_i = C$, 此时第 i 个样本是 outlier 即噪声。

7. Graphical Models(图论&概率)

7.1. 贝叶斯规则(Bayes Rule)

$$\text{加法法则: } P(x_1) = \int P(x_1, x_2) dx_2$$

$$\text{乘法法则: } P(x_1, x_2) = P(x_1) \cdot P(x_2 | x_1) = P(x_2) \cdot P(x_1 | x_2)$$

$$\text{链式法则: } P(x_1, x_2, \dots, x_p) = \prod_{i=1}^p P(x_i | x_1, x_2, \dots, x_{i-1})$$

$$\text{Bayesian 法则: } P(x_2 | x_1) = \frac{P(x_1, x_2)}{P(x_1)} = \frac{P(x_1, x_2)}{\int P(x_1, x_2) dx_2} = \frac{P(x_2) \cdot P(x_1 | x_2)}{\int P(x_2) \cdot P(x_1 | x_2) dx_2}$$

$\xrightarrow[\text{独立}]{\text{相互独立}}$ $P(x_1, \dots, x_p) = \prod_{i=1}^p P(x_i)$

7.2. 联合分布(Joint Distribution)

7.2.1. 联合分布表示列出所有变量所有可能组合的概率。其某一列的和表示为某个变量 x 等于某个值 a 的边缘概率。其总和必为1。

| $y \backslash x$ | x_1 | x_2 | x_3 | x_4 | $p_{Y X}(y)$ |
|----------------------|-------|-------|-------|-------|--------------|
| y_1 | 4/32 | 2/32 | 1/32 | 1/32 | 8/32 |
| y_2 | 3/32 | 6/32 | 3/32 | 3/32 | 15/32 |
| y_3 | 9/32 | 0 | 0 | 0 | 9/32 |
| $p_X(x) \rightarrow$ | 16/32 | 8/32 | 4/32 | 4/32 | 32/32 |

7.2.2. 给出一个联合分布表，通常有以下几种询问：

7.2.2.1. Distribution Inference Query: 例如 $P(x=a, y=b) = ?$ 或者 $P(x=a, y=b | z=c) = ?$ 都可以结合表格以及贝叶斯法则得出结果。复杂度最多为 $O(2^N)$ (这里 N 是变量个数)。

7.2.2.2. MAP(maximum a posterior) Query: 例如 $(v_1, \dots, v_n) \operatorname{argmax}_{x_1=v_1, x_2=v_2, \dots, x_n=v_n} P(x_1, \dots, x_n | v_1, \dots, v_n) = ?$ 相当于找整个表格的最大值(不考虑边缘概率)。又如 $(v_1, \dots, v_k) \operatorname{argmax}_{x_1=v_1, x_2=v_2, \dots, x_n=v_n | x_{k+1}=v_{k+1}, \dots, x_n=v_n} P(x_1, \dots, x_n | v_1, \dots, v_n) = ?$ 仍然是结合贝叶斯法则得结果。特点是必须遍历整个表格的所有变量(involve all variables)。复杂度也为 $O(2^N)$ 。

7.2.2.3. MMAP(marginal MAP) Query: 和MAP的区别是MMAP考虑的是变量的一个子集合(subset of variables)。例如 $(v_1, v_2) \operatorname{argmax}_{x_1=v_1, x_2=v_2} P(x_1, x_2 | v_1, v_2) = ?$ 注意这里就不是一定要到 v_n 了因为是subset。但是仍然需要考虑表格中的所有情况，所

以复杂度也为 $O(2^N)$ 。一个更大的区别是 **MAP** 求的是上图橙色框里的最大值，而 **MMAP** 求的是某些块的和的最大值(概率学中 **marginal=summation**)。

7.2.2.4. MAP和MLE的区别: **MAP**要多考虑一个先验概率(**prior**)。也可以说在所有变量的先验概率相同时, 就变成了**MLE**问题。

7.3. 贝叶斯网络(Bayesian Belief Network || Belief Network || Bayesian Network)

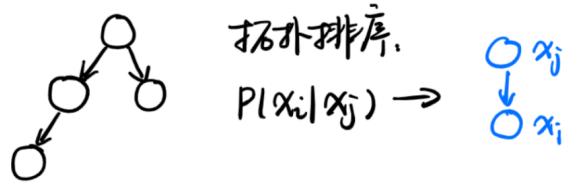
7.3.1. 特点: 是一个离散的有向无环图(DAG)。

7.3.2. 方法: 通过拓扑排序将变量之间的关系转化为一个有向无环图(DAG)。

$$P(x_1, x_2, \dots, x_p) = P(x_1) \prod_{i=2}^p P(x_i | x_{1:i-1})$$

条件独立性: $x_A \perp x_C | x_B$

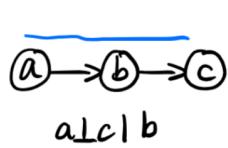
因分母: $P(x_1, x_2, \dots, x_p) = \prod_{i=1}^p P(x_i | \text{pa}_{\text{list}})$ x_i 的父集



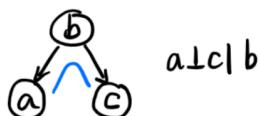
拓扑排序
 $P(x_i | x_j) \rightarrow$



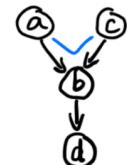
7.3.3. **D-Separation**: 下图表示的三种情况。其中如果一条路没有被阻塞(D-connected), 我们称为active path; 一条active path上的所有变量都是active的。在第一、二种情况下, b不被观测到时b是active的; 在第三种情况下, 或者它的子孙被观测到时b是active的。如果路径不active, 那a,c就是独立(independent)的。



a $\perp\!\!\! \perp$ c | b
若b被观测到, 路径被阻塞。

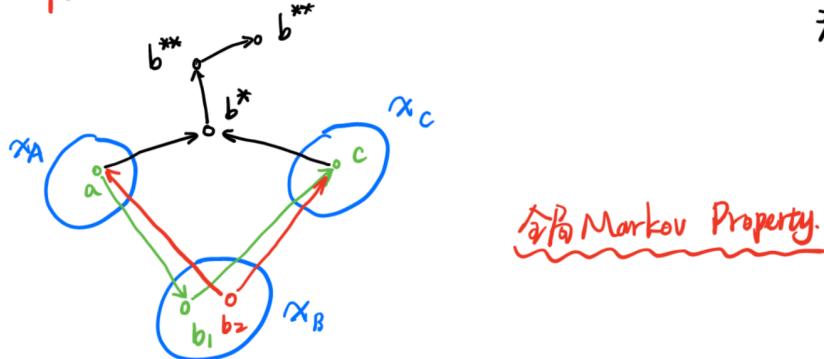


若b被观测到, 路径被阻塞。



观测情况下, a $\perp\!\!\! \perp$ c, 即路径是阻塞的。
若b被观测到, 则路径连通。
若d被观测到, 则路径连通
 $a \rightarrow c$.

D-Separation



7.3.4. **Conditional Probability Table(CPT)**: 条件概率表可以理解为只考虑联合概率的一部分, 所有的CPT的乘积就是**Joint Distribution**。一个CPT的例子如下: 其中 $\frac{4}{9} \div \frac{6}{9} = \frac{4}{6}$, $\frac{2}{9} \div \frac{6}{9} = \frac{2}{6}$, $\frac{1}{9} \div \frac{3}{9} = \frac{1}{3}$, $\frac{2}{9} \div \frac{3}{9} = \frac{2}{3}$ 。可以看出CPT的特点是列的和为1。

| | x=0 | x=1 | P(y) |
|------|-----|-----|------|
| y=0 | 4/9 | 1/9 | 5/9 |
| y=1 | 2/9 | 2/9 | 4/9 |
| P(x) | 6/9 | 3/9 | 1 |

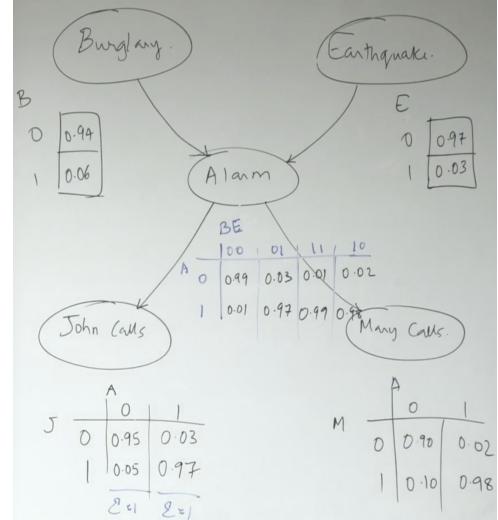
Joint Distribution: ;

| | x=0 | x=1 |
|----------------|-----|-----|
| P(y=0 given x) | 4/6 | 1/3 |
| P(y=1 given x) | 2/6 | 2/3 |
| Sum | 1 | 1 |

CPT:

7.3.5. 贝叶斯网络&CPTs例子:

每个variable都有自己的CPT, 并且可以利用Bayes Rule和D-Separation证明在这个图中有 $P(B)*P(E)*P(A|B,E)*P(J|A)*P(M|A) = p(A,B,E,J,M)$, 即**CPTs的乘积等于联合分布**。这说明了贝叶斯网络和直接联合分布能获得一样的结果, 但是贝叶斯网络需要考虑的情况少得多(在这个例子中只需要计算 $1+1+4+2+2=10$ 个概率, 因为对于每一列只要知道了一个概率p另一个就等于1-p)。而直接联合分布需要计算 $2^5=32$ 个概率。



7.3.6. 贝叶斯网络的复杂度: 这里定义parent(x)

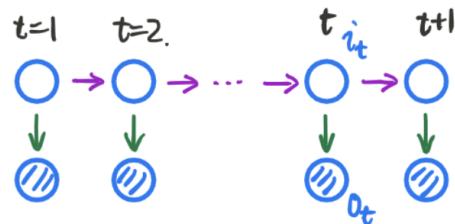
表示直接指向x的节点的集合; family(x)表示 $\{x\} \cup \text{parent}(x)$ 。假设k表示一个贝叶斯网络中最大的家族大小(the largest family size), 则贝叶斯网络的复杂度为 **O(N*2^k)**。

| Comparison | Representation | Computation |
|--|--|--|
| Joint Distribution(brute force tabulate) | Size 2^N ; Unintuitive numbers to specify. | $O(2^N)$ (Inference, MAP, MMAP) |
| BN | Size $N*2^K$; Intuitive numbers to specify. | Would be great if $O(N*2^K)$ but actually it is $O(N*2^t)$. $K \leq t \leq N$. Here $t = \text{treewidth}$. |

8. HMM(隐马尔可夫模型)

8.1. 基本概念

HMM是一个贝叶斯网络(因为是一个有向无环图)。由三个部分组成： $P(i^{(t)})$ 即初始状态； $P(O^{(t)} | i^{(t)})$ 即隐变量和观测变量之间的关系； $P(i^{(t)} | i^{(t-1)})$ 即隐变量在时序上的转移。其特点是能将两个不同domain的数据结合在一起(由观测变量推测隐变量)。



8.2. 变量消除(Variable Elimination)

8.2.1. 每个CPT都是一个表格(Table)，但是变量消除的表格不一定是CPT。

8.2.2. 表格之间的两种操作：

8.2.2.1. 合并(multiplication):

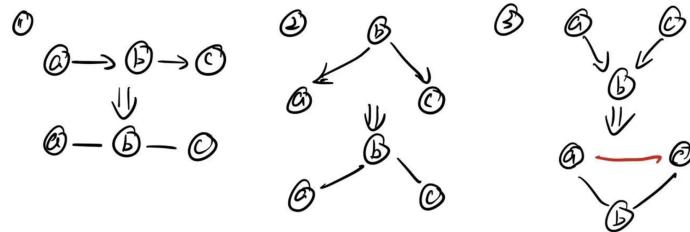
| T ₁ (x ₁ , x ₂ , x ₃) | | | | T ₂ (x ₂ , x ₄) | | | | T ₃ (x ₁ , x ₂ , x ₃ , x ₄) | | | |
|--|----------------|----------------|-------|---|----------------|-------|----------------|---|----------------|----------------|-----------|
| x ₁ | x ₂ | x ₃ | Value | x ₂ | x ₄ | Value | x ₁ | x ₂ | x ₃ | x ₄ | Value |
| 0 | 0 | 0 | 0.5 | 0 | 0 | 0.5 | 0 | 0 | 0 | 0 | 0.5 × 0.5 |
| 0 | 0 | 1 | 0.5 | 0 | 1 | 0.2 | 0 | 0 | 0 | 1 | 0.5 × 0.2 |
| ⋮ | | | | ⋮ | | ⋮ | ⋮ | | | ⋮ | ⋮ |
| 1 | 1 | 1 | 0.9 | 1 | 1 | 0.1 | 1 | 1 | 1 | 1 | 0.9 × 0.1 |

8.2.2.2. 映射(projection):

| T ₁ (x ₁ , x ₂ , x ₃ , x ₄) | | | | T ₂ (x ₁ , x ₂ , x ₃) | | |
|---|----------------|----------------|----------------|--|----------------|----------------|
| x ₁ | x ₂ | x ₃ | x ₄ | x ₁ | x ₂ | x ₃ |
| 0 | 0 | 0 | 0 | 0.7 | | |
| 0 | 0 | 0 | 1 | 0.2 | | |
| ⋮ | | | | ⋮ | | |
| 0 | 1 | 0 | 0 | 0.3 | proj | |
| 0 | 1 | 0 | 1 | ab | | |
| ⋮ | | | | ⋮ | | |
| 1 | 1 | 1 | 1 | 0.9 | | |

用什么规则取决于
在解决什么问题

8.2.3. 变量关系图(variable interaction graph): 其实就是有向无环图转无向图。对应D-Separation的三种结构有三种转移方式。转换以后的无向图也叫做道德图Moralized Graph。



8.2.4. 变量消除算法: 在原有的variable interaction graph中消除部分变量的同时维持原有的限制(constraint)不变。过程例子如下:

example: variables: $x_1, x_2, x_3, x_4 \in \{0, 1\}$

interaction graph: $x_1 - x_2 - x_3 - x_4 \Rightarrow$ solution: $\begin{cases} x_1 = 1 \\ x_2 = 0 \\ x_3 = 1 \\ x_4 = 0 \end{cases}$
constraints: $\{x_1 > x_2, x_2 + x_3, x_3 \neq x_4\}$

Now eliminate x_3 :

① multiplication:

| x_2 | x_3 | Value | x_3 | x_4 |
|-------|-------|-------|-------|-------|
| 0 | 0 | X | (0) | 0 |
| 0 | 1 | ✓ | (1) | 0 |
| 1 | 0 | ✓ | (1) | 0 |
| 1 | 1 | X | (1) | 0 |

② projection

| x_2 | x_3 | x_4 | Value |
|-------|-------|-------|----------|
| 0 | 0 | 0 | X (0, 0) |
| 0 | 0 | 1 | X (0, 1) |
| 0 | 1 | 0 | ✓ |
| 1 | 0 | 0 | X (1, 0) |
| 1 | 0 | 1 | X (1, 1) |
| 1 | 1 | 0 | X (1, 0) |
| 1 | 1 | 1 | X (1, 1) |

record

③ new interaction graph: $x_1 - x_2 - x_4$

new constraints: $x_1 > x_2, x_2 = x_4 \Rightarrow$ solution $\begin{cases} x_1 = 1 \\ x_2 = 0 \\ x_4 = 0 \end{cases}$ ★消除一个variable后要互相连接它的所有邻居。

④ 如果想找回 x_3 :

找 projection 表: $x_2 = 0, x_4 = 0 \Rightarrow x_3 = 1$ (和原solution一致)

8.2.5. 变量消除对MAP和MMAP的不同:

8.2.5.1. 应用在MAP时, 由于考虑的是所有单独变量, 所以在projection时用的function时“max”。

8.2.5.2. 应用在MMAP时, 由于只考虑变量的一个子集, 所以在projection时先对non-MAP变量用“sum”projection消除, 再对真正关心的 x_1, \dots, x_k 用“max”projection消除。

8.2.6. 变量消除的顺序(order)与复杂度：假设有N个变量需要消除。**不同的变量消除顺序会导致不同的复杂度**(如果消除一个有4个邻居(neighbor)的变量，表的大小就是 $5*5=25$, 如果先消除它的邻居，表的最大大小就是 $2*2=4$, 相差很多)。

8.2.6.1. 所有的消除顺序的方案数: **N!** (NP-Hard)

8.2.6.2. 定义**k = the size of largest family**, 即在消除所有变量的过程中表格的最大包含变量数;

8.2.6.3. 定义**w = induced width = k -1**。

8.2.6.4. 定义**t = the minimum induced width of all possible order**。也叫**Treewidth**。

8.2.6.5. 总复杂度为**O(N*2^t)**。

8.2.7. 变量消除与**HMM**:当变量消除算法应用于HMM时，它是一个动态规划并且复杂度是线性的(因为HMM本身是一棵树的形式, $t=1 \Rightarrow O(N*2^t)=O(2N)$)。这个算法叫做**Viterbi algorithm**。

8.3. 树宽 (Treewidth)

8.3.1. 定义：如8.2.6.4, 树宽就是**minimum induced width of all possible variable elimination order**; 也可以说是一个无向图离成为一棵树的距离。

8.3.2. 特性：

8.3.2.1. 通常(**in general**)是NP-Hard的(在某些特殊情况下很容易证明是线性，例如HMM)。

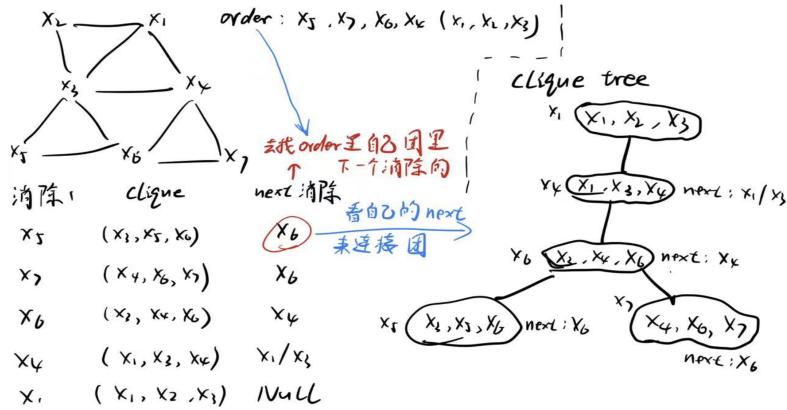
8.3.2.2. 存在近似算法(approximation algorithm)。

8.3.2.3. 可以用启发式(heuristics)方法解决(每次消除点的度数(degree)最小的变量)。

⚠ 注意:不能排序以后按点的度数从小到大消除(每次消除一个变量需要将其所有邻居互相连接，因此很多点的度数是在动态变化的)。

8.3.2.4. 度数都很小的图不一定树宽很小。例如n*n的网状图，每个点的度数只有4但是其树宽treewidth= $\sqrt{n^2} = n$ 。

8.3.3. Clique Tree (Junction-tree)



9. 过拟合, 正则化和交叉验证

9.1. 过拟合(Overfitting)

9.1.1. 概念: 训练出的拟合函数过于复杂导致在训练集拟合的非常好但是在测试集的表现不佳(拟合了过多噪声)。

9.1.2. 解决方法:

9.1.2.1. 减少特征(降低任务复杂度);

9.1.2.2. 增加训练数据(让模型接触到更广泛的情况);

9.1.2.3. 正则化

9.2. 正则化(Regularization)

9.2.1. 概念:可以理解为增加了一个限制条件让模型不要学的太好。例如模型原函数为

$w_0 + w_1x + w_2x^2 + \dots + w_nx^n$, 增加一个限制 $\sum_{i=1}^n w_i \leq C$ 就可以控制模型学习的程度, 这

里**C越大说明限制越宽松**, 模型就容易**过拟合**; **C越小说明限制越严格**, 模型就容易**欠拟合**。

9.2.2. 形式:为了把这个限制作为惩罚加在训练过程中, 我们修改代价函数(cost

function)为 $J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (w_i x^{(i)}) - y^{(i)} \right]^2 + \lambda \sum_{j=1}^n w_j^2 \text{ or } |w_j| \right]$, 目标就是最小化

$J(\theta)$ 。这里的 **λ 称为正则化参数**。可以看出 **λ 越大则惩罚的越多, λ 越小则惩罚的越少**, 与**C相关联就是C越大 λ 越小, 拟合出来的曲线越简单; C越小 λ 越大, 拟合出来的曲线越复杂**。

9.2.3. 增加了正则化的 $J(\theta)$ 被称为 E_{AUG} 即Augmented Error。顺便说下 E_{IN} 表示模型假设对样本(已知)的错误率; E_{OUT} 表示对真实情况(未知)的错误率。

9.2.4. 问题: 如何确定最佳的 λ 呢? →交叉验证

9.3. 交叉验证(Cross-Validation)

9.3.1. 目的:确定在给定数据集中表现最好的**假设空间(hypothesis space)**。

9.3.2. 适用情况(找最佳值):

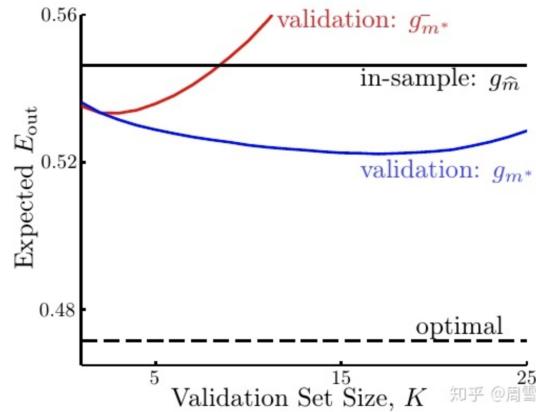
9.3.2.1. 确定**Regularization**中的 λ ;

9.3.2.2. 确定**soft-margin SVM**中的C;

9.3.2.3. 确定**SVM**中的**kernel function**;

9.3.2.4. 确定**NN**中的**epoch**次数。

9.3.3. 验证集(validation set): 属于数据集的一部分, 从数据集中分出一部分用来测试模型的效果。设数据集为 D , 去掉验证集以后的训练集为 D^- , 用训练集训练出各个模型后, 用验证集选出其中最好的模型(我们把此模型称为 $g_{m^*}^-$), **记录最好模型的假设空间**(比如说使用哪个算法, 迭代次数是几次, 学习速率是多少, 特征转换的方式是什么, 正则化方式是哪种, 正则化系数是多少等等), 然后整个数据集再训练出一个新模型, 作为最终的模型(我们把此模型称为 g_{m^*}), 这样得出的模型效果会更好, 其测试误差会更接近于泛化误差。从下图可以看出 $g_{m^*}^-$ 随着验证集的增加效果先变好后变差, 因为训练集的数据越来越少了。**那么如何解决数据分配的问题呢?** →交叉验证



9.3.4. 交叉验证:

9.3.4.1. **LOOCV(Leave-one-out cross-validation)**: 只用一个数据作为测试集, 其他的数据都作为训练集, 并将此步骤重复N次(N为数据集的数据数量), 最后将误差取平均得到整体误差。这么做的好处是既保证了每次的训练集样本都尽可能大, 又考虑了每一个样本对总体误差的影响。缺点是复杂度太高。



9.3.4.2. **K-fold Cross Validation**: 把数据集分成K份, 每次使用其中一份作为验证集而将其他都作为训练集。相当于一定程度上松弛了LOOCV的条件从而牺牲很小的误差来大幅度降低复杂度。

10. 强化学习(Reinforcement Learning)

10.1. Bellman Update

其实Bellman Update就是Bellman–Ford algorithm求最短路。简单来说就是利用迭代从终点一圈一圈向外扩散到终点的最短路。每次转移都找邻居的最小值。转移方程为 $d_G(s) = \min_{a \in A(s)} [cost(a) + d_G(s')]$ 。这里A(s)表示action的集合, $d_G(s)$ 表示离终点的距离。其优点是可以处理负边; 可以简单将理念用在stochastic shortest path

10.2. Stochastic Shortest Path

还是最短路问题, 但是在每次进行一个action时有一定的概率转移到其他状态而不是完美地得到计划的结果。 $P(s'|s, a)$ 表示在s这个状态执行a这个操作以后转移到s'这个状态的概率。这样应用Bellman Update就可以把转移方程修改成:

$d_G(s) = \min_{a \in A(s)} [cost(a) + \sum_{s'} P(s'|s, a) * d_G(s')]$, 表示考虑了所有可能的转移以后的状态, 停止条件是 $d_G(s)$ 收敛即两次迭代的结果之差小于设定阈值。

10.3. 马尔可夫决策过程(MDP)

10.3.1. 符号定义:

| 符号 | 定义 |
|---------------------------|---|
| $s \in S'$ | state即状态 |
| $a \in A(s)$ | actions即可执行的动作 |
| s_0 | Initial state初始状态 |
| $P(s' s, a)$ | Transiton probability转移概率, 即在s状态执行动作a转移到s'状态的概率 |
| $R(s, a, s')$ | Reward即从s状态执行a动作转移到s'状态能获得的奖励 |
| $\pi: s \rightarrow A(s)$ | policy策略, 即在s状态下执行A(s)中的某个动作 |
| γ | Discount factor折扣, 即后续的奖励在经过时间的沉淀以后会对当前的奖励显得不那么重要。举例: 假设时刻的总奖励为G 则 $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T$ 注意这里考虑的是后续时间因为在强化学习中奖励是延迟(delay)的(只有玩到最后才知道自己赢没赢)。 |

| | |
|------------|---|
| $V^\pi(s)$ | Value of state s under policy π 。价值函数，即采用 π 这个策略在 s 这个状态能得到的价值。 |
| π^* | Optimal policy最优策略，即从初始状态出发能获得最大价值的策略。 $\pi^* = \operatorname{argmax}_{\pi} V^\pi(s_0)$ |

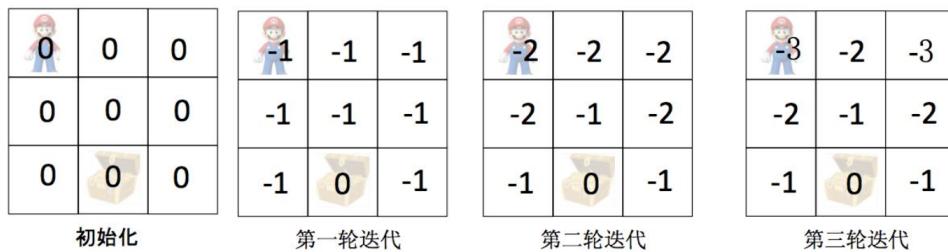
如何获得最优策略呢？→①Value Iteration; ②Policy Iteration

10.3.2. 价值迭代(Value Iteration): <https://zhuanlan.zhihu.com/p/33229439>。

10.3.2.1. 例子:假设我们有一个3 x 3的棋盘:

- 有一个单元格是超级玛丽，每回合可以往上、下、左、右四个方向移动。
 - 有一个单元格是宝藏，超级玛丽找到宝藏则游戏结束，目标是让超级玛丽以最快的速度找到宝藏。
 - 假设游戏开始时，宝藏的位置一定是(1, 2)。
- 这是一个标准的马尔科夫决策过程(**MDP**)，其中：
- 状态空间**S'**:超级玛丽当前的坐标
 - 决策空间**A(s)**:上、下、左、右四个动作
 - Action对State的影响和回报
 - $P(s' | s, a)$:这里假设所有关系都是已知的
 - 超级玛丽每移动一步，reward = -1
 - 超级玛丽得到宝箱，**reward = 0**并且游戏结束

现在利用**Value Iteration**求解该问题的**MDP**:



初始化

第一轮迭代

第二轮迭代

第三轮迭代

1. 初始化:所有state的价值 $V(s) = 0$
2. 第一轮迭代:对于每个state，逐一尝试上、下、左、右四个Action
 - 记录Action带来的Reward、以及新状态 $V(s')$
 - 选择最优的Action，更新 $V(s) = \max(\text{Reward} + V(s')) = -1 + 0$
 - 第一轮结束后，所有状态都有 $V(s) = -1$ ，即从当前位置出发走一步获得Reward=-1
 - ★ 这里特别注意宝箱是目标，因此宝箱永远为0
3. 第二轮迭代:对于每个state，逐一尝试上、下、左、右四个Action
 - 记录Action带来的Reward、以及新状态 $V(s')$

- 选择最优的Action, 更新 $V(s) = \text{Reward} + V(s')$
- 对于宝箱周围的State, 最优的Action是一步到达宝箱, $V(s) = \max(\text{Reward} + V(s')) = -1 + 0$ ★(注意宝箱的Reward是0)
- 对于其他State, 所有的Action相同, $V(s) = \max(\text{Reward} + V(s')) = -1 + -1$
- 第二轮结束后, 宝箱周围的State的价值保持不变 $V(s) = -1$, 其他State的价值 $V(s) = -2$

4. 第三轮迭代: 对于每个state, 逐一尝试上、下、左、右四个Action

- 记录Action带来的Reward、以及新状态 $V(s')$
- 选择最优的Action, 更新 $V(s) = \text{Reward} + V(s')$
- 对于宝箱周围的State, 最优的Action是一步到达宝箱, $V(s) = \text{Reward} + V(s') = -1 + 0$
- 对于宝箱两步距离的State, 最优的Action是先一步到达宝箱周边的State, $V(s) = \text{Reward} + V(s') = -1 + -1$
- 对于宝箱三步距离的State, 所有Action相同, $V(s) = \text{Reward} + V(s') = -1 + -2$

5. 第四轮迭代: 对于每个state, 逐一尝试上、下、左、右四个Action

- 在第四轮迭代中, 所有 $V(s)$ 更新前后都没有任何变化, 价值迭代已经找到了最优策略。

10.3.2.2. 内核: 其实这里用的就是Bellman Update, 也叫**贝尔曼方程 (Bellman**

Equation: $V_*(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V_*(s')]$ 。上个例子中默认 $\gamma = 1$ 而已。

10.3.2.3. **如何求最优策略?** 我们得到的只是每个位置的最优价值, 求最优策略其实和求得到最短路的路径是相同的, 只需要在每次迭代的时候记录是哪个邻居给了自己最优的价值 $V_*(s)$ 。

10.3.3. 策略迭代(Policy Iteration): <https://zhuanlan.zhihu.com/p/34006925>。

★ **什么叫策略? policy: recommended action at every state.** 人话: 策略就是根据当前状态决定该采取什么Action。以超级玛丽寻找宝箱为例, 超级玛丽需要不断朝着宝箱的方向前进

- 当前状态在宝箱左侧, 策略应该是朝右走
- 当前状态在宝箱上方, 策略应该是朝下走

★ **如何衡量策略的好坏?** —— 策略评估 (Policy Evaluation)

- 给定一个策略, 我们可以根据价值函数计算出每个状态的期望价值 $V(s)$ 。

★ **如何找到更好的策略?** —— 策略迭代 (Policy Iteration) **特别像EM算法**

- 初始化:随机选择一个策略作为初始值,比如说不管什么状态,一律朝下走,即 $P(\text{Action} = \text{朝下走} | \text{State}) = 1$, $P(\text{Action} = \text{其他Action} | \text{State}) = 0$
- 第一步 策略评估 (**Policy Evaluation**):根据当前的策略计算 $V(s)$
- 第二步 策略提升 (**Policy Improvement**):计算当前状态的最好Action,更新策略, $\pi(s) = \operatorname{argmax}_a \sum_{s'} [R(s, a, s') + \gamma V(s')]$

- 不停的重复策略评估和策略提升,直到策略不再变化为止。

10.3.3.1. 例子相同,再玩一次:

现在利用**Policy Iteration**求解该问题的MDP:

| | Policy | | | Expected Value | | | | |
|-------|--------|---|---|----------------|----|----|--|--|
| 初始化 | ↓ | ↓ | ↓ | -∞ | -2 | -∞ | | |
| | ↓ | ↓ | ↓ | -∞ | -1 | -∞ | | |
| | ↓ | ↙ | ↓ | -∞ | 0 | -∞ | | |
| | | | | | | | | |
| 第一轮迭代 | → | ↓ | ← | -3 | -2 | -3 | | |
| | → | ↓ | ← | -2 | -1 | -2 | | |
| | → | ↙ | ← | -1 | 0 | -1 | | |

1. 初始化:无论超级玛丽在哪个位置,策略默认为向下走
 - 策略评估:计算 $V(s)$
 - 如果宝藏恰好在正下方,则期望价值等于到达宝藏的距离(-2或者-1)
 - 如果宝藏不在正下方,则永远也不可能找到宝藏,期望价值为负无穷
 - 策略提升:根据 $V(s)$ 找到更好的策略
 - 如果宝藏恰好在正下方,则策略已经最优,保持不变
 - 如果宝藏不在正下方,根据 $\pi(s) = \operatorname{argmax}_a \sum_{s'} [R(s, a, s') + \gamma V(s')]$ 可以得出最优策略为横向移动一步
2. 第一轮迭代:通过上一轮的策略提升,这一轮的策略变成了横向移动或者向下移动
 - 策略评估:计算 $V(s)$
 - 如果宝藏恰好在正下方,则期望价值等于到达宝藏的距离(-2或者-1)
 - 如果宝藏不在正下方,当前策略会选择横向移动,期望价值为-3, -2, -1
 - 策略提升:根据 $V(s)$ 找到更好的策略
 - 如果宝藏恰好在正下方,则策略已经最优,保持不变

- 如果宝藏不在正下方, 根据 $\pi(s) = \operatorname{argmax}_a \sum_{s'} [R(s, a, s') + \gamma V(s')]$ 可以得出当前策略已经最优, 保持不变

10.3.4. 价值迭代和策略迭代的相同点:

10.3.4.1. 都属于**Model-based**方法, 这种方法假设我们知道**Action带来的Reward**和**新状态, 即 $P(s' | s, a)$ 和 $R(s, a, s')$**

10.3.4.2. Action和state都必须是离散值。

10.3.5. 价值迭代和策略迭代的区别:

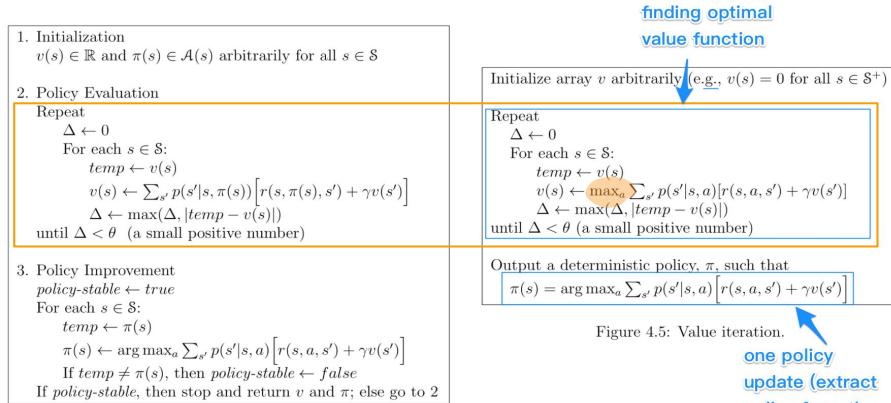


Figure 4.3: Policy iteration (using iterative policy evaluation) for v_* . This algorithm has a subtle bug, in that it may never terminate if the policy continually switches between two or more policies that are equally good. The bug can be fixed by adding additional flags, but it makes the pseudocode so ugly that it is not worth it. :-)

10.3.5.1. 可以把Value Iteration看成是简化的Policy Iteration。因此, Value Iteration在大尺度问题上速度更快(**value iteration is better for large scale problems**)。

10.3.5.2. 在Policy Iteration中:不断迭代Policy Evaluation和Policy Improvement, **逐渐优化Policy**并且获得准确的 $V_*(s)$ 。

10.3.5.3. 对比之下, 在Value Iteration中:只不断迭代Policy Evaluation获得一个不太准确的 $V_*(s)$ 。再根据这个不太准确的 $V_*(s)$ 做一次Policy Improvement。

10.4. Model-free Method (Q-learning & SARSA)

10.4.1. Model-free和Model-based的区别: Model-based已知 **$R(s, a, s')$ 和 **$P(s' | s, a)$**** , 而Model-free不知道。

10.4.2. TD(λ)(Temporal Difference): 已知策略 π 求其价值函数 $V^\pi(s)$ 。

10.4.2.1. 特点: 不需要知道 **$R(s, a, s')$ 和 **$P(s' | s, a)$**** , 添加了 α 表示trace decay, 可以理解为学习率, 表示这次的误差有多少需要被学习。

10.4.2.2. 更新方程: $V^{new}(s) = V^{old}(s) + \alpha [R(s, a, s') + \gamma V^{old}(s') - V^{old}(s)]$

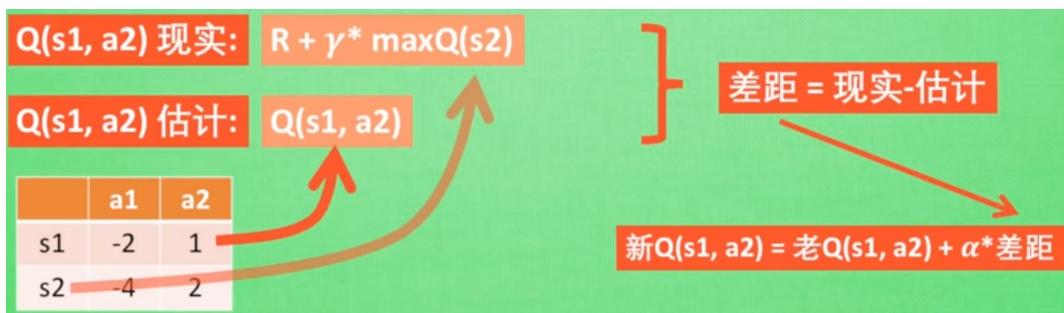
10.4.2.3. 其中 $R(s, a, s') + \gamma V^{old}(s')$ 被称作TD的目标值；

$R(s, a, s') + \gamma V^{old}(s') - V^{old}(s)$ 被称作误差。

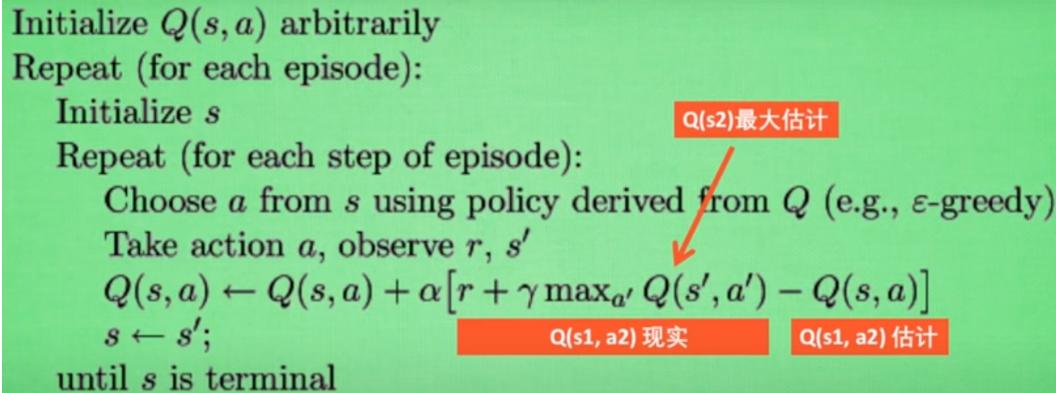
10.4.3. **Q-learning**: 哪也不知道求最优策略 π^* 。**(off-policy)**

10.4.3.1. 特点：利用了TD的想法，假设一个函数 $Q(s, a)$ 表示在某一个时刻的 s 状态下，采取动作 a 能够获得收益的期望。假设我们知道了 $Q(s, a)$ 我们就能求出 $V^\pi(s)$ ，也可以求出对应的 π 。因此有 $V^\pi(s) \sim Q(s, a) \sim \pi$ 。

10.4.3.2. 更新方程： $Q(s, a) = Q(s, a) + \alpha [R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 。这里的 $\max_{a'} Q(s', a')$ 表示选择最优的动作 a' 的收益期望。



10.4.3.3. 算法流程：



10.4.4. Exploitation-exploration trade-off

10.4.4.1. 概念：

- Exploitation: 根据当前信息，由训练的模型做出最佳的决策；
- Exploration: 探索未知的领域，比如在某个state执行之前在这个state没有执行的action。

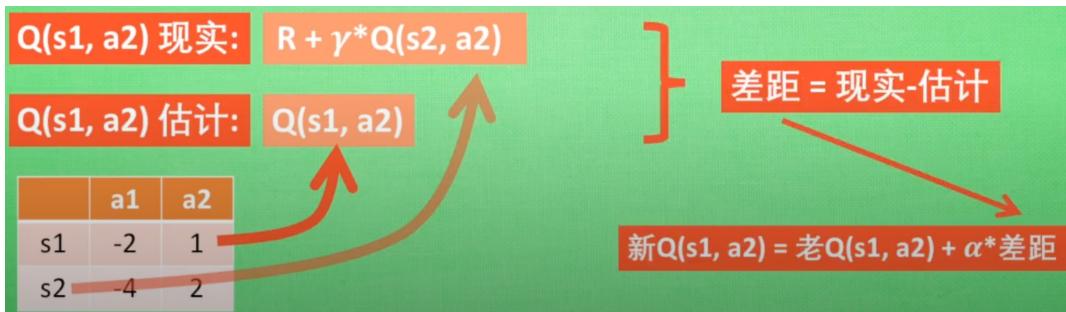
10.4.4.2. 目的：做exploitation和exploration的目的就是获得一种长期收益最高的策略，这个过程可能对short-term reward有损失。如果exploitation太多，那么模型比

较容易陷入局部最优，但是exploration太多，模型收敛速度太慢。这就是exploitation-exploration困境。

10.4.5. SARSA 和 Q-learning 解决相同问题。(on-policy)

10.4.5.1. 特点：在Q-learning中我们每一轮迭代都在寻找最优的动作 a ，也就是一直在做Exploitation。这么做容易局部最优。因此，SARSA提出了一个**贪心因子 ε** ，在每次选择动作的时候，有 ε 的概率选择一个随机的**action**执行；有 $1 - \varepsilon$ 的概率选择当前认为的最优**action**。

10.4.5.2. 更新方程： $Q(s, a) = Q(s, a) + \alpha[R(s, a, s') + \gamma Q(s', a') - Q(s, a)]$ 。大致上和Q-learning相同，区别在于Q-learning的 $\gamma Q(s', a')$ 有 ε 的概率是随机选择的一个**action**而不一定是 s' 状态下最优的**action**。



10.4.5.3. 算法流程：在根据action a 得到了新的状态 s' 后，SARSA并不是**贪婪的选择 s' 状态下的最大利益期望**(Q-learning中做的)，而是给了 ε 的概率选择别的action。这就是Exploration的思想，牺牲**short-term reward**来换取获得更大的**long-term reward**的机会。

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
    Initialize  $s$ 
    Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
    Repeat (for each step of episode):
        Take action  $a$ , observe  $r, s'$ 
        Choose  $a'$  from  $s'$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
         $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'; a \leftarrow a'$ ;
    until  $s$  is terminal
  
```