高级人工智能

ch0 课堂小测 & 2024秋期末题

- 通用人工智能是什么？和人工智能的差别？
- 什么是搜索问题，请给出形式化定义
- 选择下述任何一个例子，将其描述为搜索问题，计算其状态数量，并说明解是什么
  a. 排序问题：给定n个不同的整数，将其排序；
  b. 学习问题：机器学习的训练，找最优模型参数；
  c. 下围棋问题；
  d. 自动驾驶问题；
  e. 快递无人机的设计问题；
  f. 代码自动生成问题（算法生成代码）
  g. 大语言模型生成你提问的答案
- 写一个广度/宽度优先搜索算法，支持搜索策略定制（调整fringe集合中节点的次序）
- 搜索算法评估哪三个方面？各自的含义是什么？
- 广度优先搜索的时空复杂性？
- 写出深度优先搜索、回溯、深度受限深度优先搜索、迭代深入搜索、代价一致这些算法和广度优先搜索算法的区别。以广度优先搜索算法（或其中某个算法）为出发点，指出另一个算法与之的改动之处
- A*算法的四个性质是什么？
- 约束满足问题的可交换性是指的什么？
- 将约束满足问题求解的回溯算法（page 15）画成流程图
- 指出优化问题、CSP 问题和搜索问题的异同
- 画出梯度下降算法的流程图，说明梯度下降算法的后继函数是什么，即两个状态之间连边的条件
- 优化问题 ppt 17-28 页，写出三种算法对应的后继函数设计方法
- 将 “将给定的一个中文句子翻译成英文” 建模成 MDP 问题，以 ppt 11 页的形式
- 解释 Q 和 V 值函数的区别，写出两个值函数的 bellman 递推式，V 值函数中不应出现动作 a
- 写出强化学习中model-based的方法和model-free的方法的思想上的差异
- 强化学习和MDP的差异
- 长期回报 u_t 和单步奖励 r_t 之间的关系（公式）。强化学习是追求上述二者哪一个最大化？
- DQN 论文阅读：
  1、论文的贡献是什么？三句话之内
  2、技术动机是什么？（为什么要研究或提出论文中创新的技术点）
  3、详细说明训练过程中引入的创新做法，5句话之内。
- 学习是什么？从解方程组的角度理解机器学习。谈自己的看法
- 一个数据 (x_i, y_i) 的损失定义为 $loss_i = err(h(x_i) - y_i)$，有哪些常见的损失定义方法？
- 计算真实 f 和机器学习结果 h 之间的误差，存在什么样的工程困难？解决的方法是什么？
- 分类和预测的区别是什么？
- 线性回归和线性分类的区别？从定义损失函数的角度比较
- 线性回归和线性分类几何意义的区别？
- 残差和间隔（margin）的相似和不同，它们的作用是什么？
- 支持向量机为什么不用梯度下降法来求解最小化 hinge 损失？
- 线性可分问题的 SVM 求解，其训练损失是多少？
- 代数间隔和几何间隔的区别？
- 形式化描述 SVM 求解的问题
- 非线性可分是什么意思？用二分类来举例说明
- SVM解决非线性可分问题的方法是什么？说出两种
- 核函数的作用是什么？典型的核函数举两个例子

- 核矩阵与核函数的关系是什么?
- 谈谈你对数据预处理（主要是体征提取/特征工程）的理解，该降维还是升维?
- 用单项式来设计通用的特征提取函数，做法是怎样的？存在的问题是什么？时空的额外开销是多少？用例子说明
- 画出给定神经网络的导数计算图：三层全连接神经网络，每层节点数为2，3，1，权参数为w1，w2, ..., w6，v1，v2，v3.最后输出层用平方损失，隐层激活函数\sigma，输出输入层无激活。给出w1, v1的梯度计算公式
- 卷积神经网络有哪些典型层？作用是什么?
- 卷积神经网络的四个关键技术思想是是什么?
- 机器学习中的注意力是什么？发展过程
- 列出各种不同注意力及其出现的先后次序
- 注意力、自注意力、交叉注意力，用图形来解释这三个注意力的区别；用数学公式来解释这三个注意力的区别
- 什么是离散 hopfield 网络（画图说明）？hopfield 网络的联想记忆工作原理是什么?
- 什么是玻尔兹曼机？和受限玻尔兹曼机的区别是什么？画出二者的差异
- Hinton 是如何从受限玻尔兹曼机构造深度信念网络（DBN）的？画图说明
- 给出 bayes 网络的联合概率计算乘积公式，画出每个边缘分布和条件分布的表头
- 给出 bayes 网络存储空间减少的行数

- **2024秋期末回忆**:
- 基本以小测题目为主
    - 填空:
        - 什么搜索算法是以栈为结构
        - SVM 通过什么计算高维的内积
        - 搜索问题的五要素
        - 评价搜索算法的三个属性
        - 写出三个激活函数
        - 写出 CSP 中 X、D、C 的含义
        - Q learning 更新 Q 的公式
        - 贝叶斯网络用什么代替概率密度函数
        - 代数间隔和几何间隔的公式
    - 判断:
        - 太多了，都不难，偏概念
    - 简答:
        - 写出深度优先搜索、回溯、深度受限深度优先搜索、迭代深入搜索、代价一致这些算法和广度优先搜索算法的区别
        - 如何评估 h 学习 f 的好坏程度
        - 什么是可采纳的启发式算法，举个例子
        - 写三个《Deep learning》paper 中你记忆深刻的点
        - 注意力机制：解释自注意力和交叉注意力机制，写出注意力机制公式并给出Q、K、V的维度，解释公式中的 d_k
        - 马尔可夫&强化学习：给了一个马尔可夫模型，计算最优策略与所有节点的 V_π 值
        - 贝叶斯网络：给了一个具体问题与三个该问题的贝叶斯网络模型，问哪个贝叶斯网络最好，并计算条件概率分布表

ch3 搜索问题

- 五要素：
  - **状态空间**
  - **后继函数**
  - **初始状态**
  - **目标测试**
  - **路径耗散**

- 搜索问题转化为状态图：
  - 每个节点表示一个状态
  - 弧及两端点表示后继函数
  - 状态图可能包括多个不连通的分量
  - 状态图上标有初态和终态
  - **无解**： 初态和终态不在同一个连通分量中

- 三国华容道 $\Longrightarrow$ 搜索问题
  - 棋盘的任意一种布局就是一个状态，是状态图中的一个节点，所有可能的的布局构成状态空间/状态图的顶点集；
  - 从棋盘的一种布局“合法地”移动一个旗子，得到另一种布局，即所谓的“后继函数”；
  - 任何一种初始布局可为初态，曹操跑出来为终态；
  - 每移动一个棋子，路径耗散为 1；
  - 问题的解：从初态到终态的一个移动序列；最优解：路径耗散/移动次数最少的移动序列。
- 8 皇后问题的形式化 1：现实问题建模为搜索问题
  - 状态，任何 0，1，2，……,8 个皇后在棋盘上时的布局代表一个状态
  - 初始状态：0 个皇后在棋盘上
  - 目标测试/目标状态：8 个皇后在棋盘上，彼此间不互吃
  - 路径耗散：放置一个皇后，代价为 1
  - 后继函数：在一个空的格子上放置一个皇后，得到一个新状态
  - 状态空间/状态图：高度为 9 的 64-叉树
- 8 皇后问题的形式化 2：现实问题建模为搜索问题
  - 状态，任何 0，1，2，……,8 个皇后在棋盘左侧，且互不攻击时代表一个状态；所谓棋盘左侧互不攻击，就是前 k 列每列一个皇后，互不攻击；
  - 初始状态：0 个皇后在棋盘上；
  - 目标测试/目标状态：8 个皇后在棋盘上；
  - 路径耗散：放置一个皇后，代价为 1；
  - 后继函数：在 k+1 列放置第 k+1 个皇后到一个不受攻击的位置；
  - 状态数目急剧减少！

- 宽度优先遍历为求解搜索问题的**基准算法**，简称为**宽度优先搜索算法**
- 同样的状态，在树中可能有多个节点表示；若允许状态被重复访问，则搜索树可能是无限深的
- **节点信息**包括：
  - **深度**：节点的（从根开始）路径长度，通常初始状态为搜索树的根，其深度为0
  - **路径代价**：从根开始的路径耗散
  - **是否已扩展**：

    - 节点扩展是对节点的一个操作，若完成扩展操作，则标注为yes
    - **节点扩展**直观理解就是**把一个节点用它的后继节点（集合）来替换**
    - 未扩展状态的集合（`FRINGE`），可理解为：在排队等待扩展的那些状态
- 搜索策略：定义 `FRINGE` 中节点优先顺序的策略

- 影响/设计实现搜索策略的核心操作：（类似队列操作）
  - INSERT(node,FRINGE)：向优先数组 FRINGE 中插入优先级最低的节点 node;
  - REMOVE(FRINGE): 从优先级数组中删除优先级最高的节点。

- **搜索算法框架: 基准算法**

```
Input:  𝔾: 状态图; s0: 初态;
Output:  path: 代表解的路径
path ⟵ (s0), FRINGE ⟵ ϕ /* 初始化 */;
if (GOAL(s0) = T) then
    return path = (s0);
end
INSERT(s0, FRINGE);
while T do
    if empty(FRINGE) == T then
        return failure /* 返回 failure, 表示无解 */;
    end
    N ⟵REMOVE(FRINGE) /* 将未扩展节点中队头节点从队列移除到 N*/;
    s ⟵STATE(N) /* 从节点 N 恢复为状态 s*/;
    update path;
    foreach s' in successors(s) do
        为 s' 创建 N 的新子节点 N';
        if GOAL(s') = T then
            return (path, s')/* 找到目标节点，返回解 */;
        end
        INSERT(s',FRINGE);
    end
end
```

节点扩展

- 评估搜索算法性能
  - **完备性**
  - **最优性**
  - **复杂性**:
    - 搜索树的大小：
      - 由初态、后继函数和搜索策略决定
      - 影响因素：分支因子（后继的最大个数），最浅目标状态的深度，路径的最大长度
    - 时间复杂度：访问过的节点数目
    - 空间复杂度：同时保存在内存中节点数目的最大值

ch4 盲搜索

- Uninformed / Blind（盲搜索）：`FRINGE` 是无序的
- Informed / Heuristic（启发式搜索）：将更有希望的节点放在未扩展节点集合 `FRINGE` 的前面，优先扩展

- 基准算法是盲搜索算法

- 基准算法的重要参数：
  - **分支因子** b：后继函数返回的最大状态数目
  - **从初态到目标状态的最小深度** d：搜索树上离根最近的目标节点的深度
- 对基准算法的评价：
  - 完备的
  - 每条边的路径耗散相等时满足最优性，否则不一定
  - 复杂度：最差时，时间复杂度 $1+b+b^2+\ldots+b^d=O(b^d)$
  - 问题无解时，若状态空间无限大或者任意状态可被任意次重复访问，则宽度优先搜索算法不会停止

- 对基准算法的改进：
  - **双向搜索**
    - 分别从初态和终态启动两个宽度优先算法，分别维护两个 `FRINGE` 集合，当两个集合有交集时算法结束
    - 完备性和最优性和基准算法一致，时间与空间复杂度为 $O(b^{d/2})$
    - 缺点：不方便定义终态的前驱函数
  - **深度优先搜索**：
    - 相对于宽度优先搜索，深度优先搜索使用栈结构作为 `FRINCE`
    - 在节点扩展时，新节点总是插入到 `FRINGE` 的头部（新节点的优先级最高）
    - 评价：
      - 若搜索树有限，则是完备的
      - 不一定是最优的
      - 复杂度：最差时
        - 时间复杂度 $1+b+b^2+\ldots+b^m=O(b^m)$
        - 空间复杂度 $O(bm)$
        - 其中 m 是**叶子节点的最大深度**
  - **回溯算法**：
    - 相对于深度优先搜索，回溯算法在每次扩展节点的时候，只扩展一个新节点，即 `FRINCE` 的大小为1
    - 评价：
      - 若深度优先搜索树是无限的，则回溯搜索可能是不完备的
      - 可能无法得到最优解
      - 复杂度：最差时，时间复杂度 $O(b^m)$，空间复杂度 $O(m)$
  - **深度受限深度优先搜索**：
    - 相对于深度优先搜索，深度受限深度优先搜索限制了搜索树的深度
    - 设置扩展节点的深度阈值为 k，当节点深度大于 k 时结束扩展，并返回结果
    - 结果为有解、无解、深度阈值 k 内无解三种
  - **迭代深入搜索**：
    - 相对于深度受限深度优先搜索，迭代深入搜索的目标是寻找合适的深度阈值 k 值
    - 从小到大使用不同的深度阈值 $k=0,1,2,\ldots$ 不断重复进行深度受限深度优先搜索
    - 评价：
      - 当 $k=d$ 时，是完备的
      - 当单步路径耗散相同时，是最优的；否则不一定
      - 复杂度：最差时，时间复杂度 $O(b^d)$，空间复杂度 $O(bd)$

        - 对未知问题，解的深度未知，付出较小的代价，**迭代深入搜索是首选的盲搜索算法**
    - **代价一致搜索**：
        - 相对于宽度优先搜索，代价一致搜索每次在 `FRINGE` 中**优先选择让目前获得的路径耗散最小的节点进行扩展**
        - 即 `FRINGE` 中节点将根据路径耗散从小到大进行排序，路径耗散小的节点优先级更高
        - 适用于单步路径耗散不同时的情形，**当单步路径耗散相同时，代价一致搜索等价于基准算法**
        - 能保证最优性和完备性，一般来说时空复杂性较基准算法大

	基准算法	双向搜索	深度优先搜索	回溯算法	深度受限搜索	迭代深入搜索	代价一致搜索
完备	b 有限时	b 有限时	搜索树有限时	搜索树有限时	不一定	b 有限，$k=d$ 时	b 有限时
最优	代价一样时	代价一样时	不一定	不一定	不一定	代价一样时	最优
时间	$O(b^d)$	$O(b^{d/2})$	$O(b^m)$	$O(b^m)$	$O(b^l)$	$O(b^d)$	/
空间	$O(b^d)$	$O(b^{d/2})$	$O(bm)$	$O(m)$	$O(bl)$	$O(bd)$	/

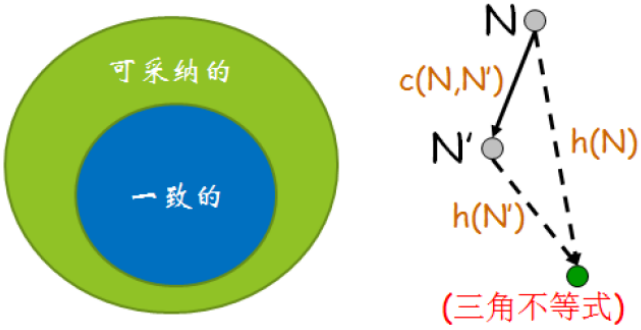
- 避免重复访问
    - 设置标识数组，标记状态是否被访问过
    - 用 Open / Closed 表，Closed 表存储所有访问过的状态，未拓展的状态用 Open 来表示
    - 若当前待扩展的转台在 Closed 中，将其丢弃，否则扩展
- 图搜索和树搜索
    - 将采用 Open / Closed 表的算法框架称为图搜索
    - 在树搜索中，不同的节点可能表示相同的状态，但是图搜索中将相同状态都合并到同一个节点上
    - 图的宽度有限搜索类似于树的层次遍历
- 完备性
    - 状态空间无限，一般不完备
    - 状态空间有限，但是每个状态允许被无限次访问，搜索不完备
    - 状态空间有限，访问重复的节点被丢弃，则搜索完备，但一般不是最优

ch5 启发式搜索

- 评估函数 f 将搜索树的节点 N 映射到一个非负实数 $f(N)$，表示从初始状态到达某个节点的路径耗散

设计节点评估函数 f 的两种方法：

- $f(N)=g(N)+h(N)$，完整解的路径耗散，对应 $A*$ 算法
- $f(N)=h(N)$，从当前节点到目标节点的路径耗散，对应 贪婪算法

- 符号解释说明
  - N：当前待判决/估计/评价的节点
  - $g(N)$：从初始节点到 N 的路径耗散
  - $h(N)$：从 N 到目标节点的路径耗散，就是所谓的 启发式函数，估计值
- 节点评估函数 f 的设计，核心在于启发式函数 h 的设计
  - 一般情形下，启发式函数值由当前节点和目标节点二者所确定
  - $h(N)$ 应保证的性质：$h(N)>=0, h(Goal)=0, h(N)$ 越小，离目标越近
- 可采纳的/admissiable 启发式函数
  - 假设 $h^*(N)$ 是节点 N 到目标节点的实际最优路径耗散
  - 启发式函数 $h(N)$ 是“可采纳的”，当且仅当 $0 \le h(N) \le h^*(N)$
  - 总是“乐观地”估计路径耗散!!!
- 

对一个可采纳的启发式函数 h

- 若它对所有节点都满足三角不等式，即 $h(N) \le c(N,N')+h(N')$，其中 N' 是 N 的后继节点，则 h 是一致的或单调的。
- $c(N,N')$ 是节点 N 到 N' 的单步路径耗散。

- **A* 算法**
  - 有解时：
    - 具有完备性、最优性
      - 对于可采纳的 $h(N)$，具有完备性（不丢弃重复访问节点，而是保存 $g(N)$ 更小的节点）与（树搜索上）最优性
    - 对于一致的 $h(N)$，A* 算法拓展一个状态节点时到该状态的路径一定是最优的
  - 无解时：
    - 若状态空间无限 / 允许重复访问，A* 算法的搜索不会停止
    - 求解实际问题时通常给一个停止时间，到达时间时自动停止
  - 图搜索和树搜索：
    - 树搜索允许状态重复访问 -> 保证获得最优解 -> 无解时可能永远停不下来

        - 图搜索避免状态重复访问 -> 状态有限时完备 -> 不保证最优
    - A* 算法状态重复访问改进：
        - 当且仅当该节点的新路径的耗散 $g(N)$ 比以前访问该状态的路径耗散更大时，丢弃重复访问的该状态节点
    - 特殊的一致启发函数 $h=0$：此时 A* 退化为宽度优先搜索（代价一致搜索）
    - 设计启发函数时，对于两个可采纳的启发式函数 h_1, h_2，若对任意节点有 $h_1 \leq h_2$，则称 h_2 比 h_1 准确
        - 更准确的启发式函数拓展的节点会更少，不精确的启发式函数会访问更多的节点
    - 有效分支因子 b^* 可以度量启发式函数的有效性

- **IDA* 算法**
    - 迭代深入 A* 算法：设置 f 的阈值，超过 f 的阈值时节点就不再扩展
    - 要求一致的启发式函数
    - 算法思想
        - 初始化 f 的阈值 t 为 $f(N0)$
        - 重复执行下述两步：
            - 执行深度优先搜索，扩展 $f(N) <= t$ 的节点 N
            - 重新设置 t 为未扩展节点中 f 的最小值
    - 优点：
        - 完备、最优
        - 内存要求比 A* 少
        - 避免了未拓展节点集合的排序开销
    - 不足：
        - 无法充分利用内存
        - 无法避免重复访问不在路径上的点

ch6 优化问题

- 优化问题是指 $minf(x)\ s.t.h(x) \leq 0$，其中 $f(x)$ 是目标函数，$h(x)$ 是约束条件
- 优化问题转化为搜索问题：
    - 初态是一个随机解
    - 状态空间由任意解构成
    - 目标状态是最小值的解
    - 后继函数是由不同算法给定的
    - 一个后继函数将把一个无序的状态空间建立某种序关系，在这一关系上每个解对应的函数值构成了优化问题的地貌图

- 后继状态是状态空间的子集
  状态图不是完全图
  边权值为概率，表示状态间转移概率

  后继状态为整个状态空间
  状态图为完全图
  边权值为 1

典型代表算法

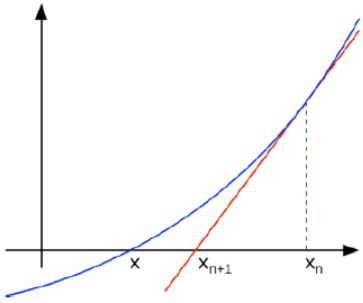
局部搜索算法
- 爬山法、梯度下降法、禁忌搜索、局部剪枝搜索、模拟退火

全局搜索算法
- 模拟退火、遗传算法、蚁群算法、粒子群算法、差分算法、EDA(估计分布算法)、Memetic 算法

- **梯度下降算法**
  - 第一步：初始化，设置算法相关参数/超参数 主要包括两个参数（参数具体使用参看第三步）：
    - 算法停止准则 T，当执行 T 次循环时，算法终止；
    - 下降步长/步幅序列，$\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_T]$。该序列一般为递减。

    第二步：初始化，初态 X_0
    - 初态 X_0 可设置为某个特殊值
    - 也可以用随机值

    第三步：循环迭代
    - $for\ s = 1, 2, ..., T$
      - $X_{s+1} = X_s - \lambda_s \nabla_X g(X_s)$, 其中
      - $\nabla_X g = (\frac{\partial g}{\partial x_1}, \frac{\partial g}{\partial x_2}, \ldots, \frac{\partial g}{\partial x_n})^t$ and $X = (x_1, x_2, \ldots, x_n)^t$
    - $endfor$
  - 缺点：
    - 不能保证获得全局最优
    - 停止条件和学习率需要人工经验
  - 改进：
    - 不知道步长就多试几个步长来尝试，选择其中使函数下降最快的步长
    - 随机梯度下降：计算梯度方向时，只用一个随机样本的信息，而不是计算所有训练数据集对梯度方向的贡献
- **牛顿法**

- 

  寻找函数 $g(X)$ 的根 X^*，使得 $g(X^*)=0$
  - 找 $g(X)$ 在 X_n 处的切线，以切线和 X 轴的交点为新点 X_{n+1}
  - $\nabla g(X_n)=\frac{g(X_n)-0}{X_{n+1}-X_n} \Rightarrow X_{n+1}=X_n-\frac{g(X_n)}{\nabla g(X_n)}$

- **爬山法**
  - 每一轮在 X_n 的邻域的表现更好的子集中找到策略 s 确定的最优点作为下一轮的 X_{n+1}
  - 策略 s：
    - **best-found**：选择使目标函数最小的邻点 -> 梯度下降的离散化版本
    - **first-found**：选遇到的**第一个更好的**邻点 -> 随机梯度下降
    - **Random Walk**：**无论是否更好**，随机选择一个邻点
  - 改进：
    - 用不同随机初始值重启算法
    - 在局部最优点增加扰动调整邻居算子
    - 模拟退火（爬山+随机行走）
    - **LBS / 局部剪枝搜索**：
      - 爬山法的并行改进：在初始时准备 k 个初态
      - 迭代中可能有两种策略：
        - 假设每个解有 l 个后继，每步都在所有 lk 个后继中选择最优的 k 个
        - 所有的 lk 个后继对应一个被选择的概率分布
- **模拟退火**
  - 以概率 p 允许“坏”的移动，而不是像爬山法一样，每次都选最好的移动
  - 每一轮在 X_n 的邻域中随机选择一个，如果表现更好则选中，否则概率地选中这个新点
  - 超参数 p 决定了移动到坏解的概率，$p=e^{-\Delta g/T}$，Δg 是初解和它邻居的目标函数之差，T 是温度
  - 在整个搜索过程中温度 T 会不断下降，T 下降的足够慢，那么算法找到最优解的概率逼近 1
- **Tabu 搜索**
  - 保留一个长度有限的**历史解列表**，如果发现的新解在列表中则退而求其次的选择其他解，可以一定程度上防止局部最优
  - 列表越长搜索域越广，列表短时的局域性比较好
- **(1 + 1) - EA**
  - 全局邻居算子的爬山法：每一轮在**全状态空间**中随机选择一个解，如果这个新解更好则转移
  - 并行化：
    - 初始时刻有 k 个解，每一轮在各自产生的 l 个后继共 kl 个候选者中根据策略选择 k 个，称为 $(k,kl)-ES$
    - 或者在 kl 个候选者和 k 个种子共 $k+kl$ 个解中选择 k 个，称为 $(k+kl)-ES$
- 蚁群算法
  - 构造性算法，解的各个部分被逐步构造出来

- 用分布式的策略来估计，让 n 只蚂蚁在有限状态机上爬行，并留下信息素，来标记爬行历史
- 在评价好的状态转移边上，会有更多的蚂蚁在上面爬行，留下信息素

ch7 约束满足问题

- **CSP 问题**
  - 形式化定义 $CSP=(X,D,C)$，是一个三元组，其中：
    - X 是一组变量，不妨假设为一个 n 维向量 $X=(X_1,X_2,\ldots,X_n)$
    - D 是值域，定义了每一个变量的值域，$D=(D_1,D_2,\ldots,D_n)$
    - C 是一个约束组，$C=(C_1,C_2,\ldots,C_m)$，是 m 个计算公式，约束 X 分量之间取值的关系；
    - $\forall i, X_i$ 从 D_i 中取值，所有变量都给一个取值，形成一个“赋值” $v=(x_1,x_2,\ldots,x_n)$，或者说是一个“解”，可以参与 $C_1,C_2,\ldots,C_m$ 表示的约束计算公式的计算，判断约束是否被满足。
  - 有限和无限CSP：解的数目是否是有限的
    - 判断的方法：变量的取值域是有限 or 无限
      - 若存在变量的取值范围是无限的，则为无限
      - 若每个变量的取值个数是有限的，则为有限
  - 将 CSP 问题形式化为搜索问题

    符号和概念定义
    - 有效赋值：对 n 个变量 $x_1,x_2,\ldots,x_n$，一个一个地赋值，不妨设前 k 个赋值为：$x_1\leftarrow v_1,x_2\leftarrow v_2,\ldots,x_k\leftarrow v_k$，此时的赋值，会使前 k 变量相关的约束条件都满足；
    - 完全赋值：$k=n$，也就是 n 个变量都被赋值，使得所有的约束条件都得到了满足。

    系统化方法
    - 状态/状态空间：有效赋值/所有可能有效赋值
    - 初始状态：$\{\},k=0$
    - 后继函数：$\{x_1\leftarrow v_1,x_2\leftarrow v_2,\ldots,x_k\leftarrow v_k\}\rightarrow\{x_1\leftarrow v_1,x_2\leftarrow v_2,\ldots,x_k\leftarrow v_k,x_{k+1}\leftarrow v_{k+1}\}$
    - 目标测试：$k=n$
    - 路径耗散：假设单步路径耗散为 1
  - CSP 问题特性：**可交换性**
    - 变量赋值的次序和完全赋值的可达性无关
    - 扩展节点时可以任意选择一个未赋值的变量来扩展节点
    - 不需要存储到达当前节点的路径，因此可以用回溯算法

- 回溯算法
  -

算法框架：CSP-BACKTRACKING(A)

```
Input: 约束条件; 有效赋值 A
Output: A: 完全有效赋值
if 有效赋值 A 是完全的 then
    return A
end
X ← 选择一个未赋值变量;
D ← 选择一个 X 所有可能取值的次序，例如随机次序;
foreach v in D do
    Add(X ← v) to A;
    if A 是有效的 then
        result ←CSP-BACKTRACKING(A) /* 递归调用 */;
        if result ≠ failure then
            return result;
        end
    end
    Remove(X ← v) from A;
end
return failure;
```

算法启动：CSP-BACKTRACKING({})

- 改进：
  - **前向检验**技术：提前检测出冲突，每进行一次赋值，就把未赋值的变量中现在不再能取的值从其候选值域中去掉
  - 优先选择约束最强的变量，即**残留值域最小**的变量赋值
  - 优先选择在未满足约束中出现次数最多的变量，即**最多被约束着**的变量赋值
  - 选择好变量后，优先赋值为：**使其他未赋值变量的残留值域缩减最小**的值
    - 做法：对于每个值，都进行向前检验，看哪个值对其他未赋值变量的残留值域大小的影响最小
  - 约束传播技术

- 进一步加速 CSP 求解算法：**约束图**
  - 每个变量是一个节点，变量间连边，构成图
  - 连边规则：两个变量参与某个约束就连边。边意味着边两端的变量在赋值时需要考虑另一个变量的取值
  - 某些变量赋值后，这些顶点间的连边参与的约束条件被满足，可以移除这些边，图有可能变成不连通的子图；对子图递归求解

ch8 马尔可夫决策

- 马尔可夫决策过程 & 经典搜索：
经典搜索	MDP
行动是**确定**地让搜索沿一条边前进	行动**随机**地让搜索沿多条边前进
解是一条路径或是状态序列	解不是路径而是**策略**：一组从状态到行动的映射关系
  - 经典搜索问题是 MDP 在概率只能取值为 0 或 1 时的特例

- **MDP 问题形式化**：

MDP/马尔科夫决策过程：一种搜索问题的变型

- S：状态空间
- 初态：s_0
- 行动：$Action(s)$，给定状态 $s \in \mathbf{S}$，合法行动集合
- 状态转移概率：$T(s, a, s')$，从状态 s 出发，采用行动 a，导致结果状态 s' 的概率；
- 奖励：$Reward(s, a, s')$，状态转移 (s, a, s') 得到的收益
- 目标测试：$isEnd(s)$

- 行动 + 状态转移概率 = 后继函数
- 奖励 = 路径耗散（最大化奖励 & 最小化路径耗散）
- **马尔科夫性**：行动的确定只和当前的状态有关

- **概念定义**：

状态转移：$s \to s'$

- 任意给定一个状态 s 和任意一个行动 a，其状态转移到一个可能的“后继状态”集合，而转移到这些可能后继状态的概率形成一个分布，即概率和为 1
- 用公式描述，即$\sum_{s' \in \mathbf{S}} T(s, a, s') = 1$

行动收益

- 对每个行动 s，定义其收益为它导致状态转移带来的奖励：$U(s, a, s') = Reward(s, a, s')$
- 该收益获得的概率是 $T(s, a, s')$
- 该行动的期望收益：$\sum_{s' \in \mathbf{S}} T(s, a, s')\, U(s, a, s')$

路径收益

- 路径 $s_0, a_0, s_1, a_1, \ldots, s_e$ 上所有行动带来的收益之和；

策略收益

- 一个策略 (记为 π) 会造成多条从初态到终态的路径，每条路径的出现概率可能不一样，收益可能也不一样；
- 所有从初态到终态的路径的收益期望，我们将之定义为 策略的价值，即策略收益，记为 V_π。

策略 $\pi : s \in \mathbf{S} \to a \in Action(s)$

- 即 $a = \pi(s)$
- 策略评估算法能计算一个策略的价值 $V_\pi(s)$

策略空间的大小

- 假设有 n 个状态 $s_1, s_2, \ldots, s_n$
- 策略空间的大小：$\Pi_{s_i \in \mathbf{S}} |Action(s_i)|$，状态数目 $|\mathbf{S}|$ 的指数函数

- 使用递推式计算 MDP 的策略价值：
  - 计算策略价值即为**策略评估**

推广：计算策略价值的方法

- 思路：找到递推公式，后解之；
- 通用地推公式：$V_\pi(s) = \sum_{s' \in \mathbf{S}} T(s,a,s')[U(s,a,s') + V_\pi(s')]$
- 上述公式中 s 表示初态/当前状态；每个状态都可以列出一个上述递推式，联立这些递推式，得到方程组（n 个状态，n 个未知数 $V_\pi(s)$,n 个线性方程），解之。

策略评估算法思想：解递推方程组的方法

- 初始化所有状态的策略价值 $V_\pi(s) = 0$
- Repeat T 次:
  - 对每个状态 $s \in \mathbf{S}$，执行:
    - 利用递推方程循环更新 $V_\pi(s) = \sum_{s' \in \mathbf{S}} T(s,a,s')[U(s,a,s') + V_\pi(s')]$

- Q-value
  - Q-value: $Q_\pi(s,a)$ 定义为从状态 s 出发，采用行动 a 后继续采用策略 π 的价值/收益；
  - Q-value 与 V_π 的区别：在状态 s 时，采用了不同的行动，故 $V_\pi(s)$ 仅仅是 π, s 的函数，而 $Q_\pi(s,a)$ 是 π, s, a 的函数
  - V 函数：在状态 s 处采取策略 π 的期望收益，只与当前状态和策略有关
    - $V_\pi(s) = \sum_{s' \in \mathbf{S}} T(s,a,s')[U(s,a,s') + V_\pi(s')] = \sum_{s' \in \mathbf{S}} T(s,\pi(s),s')[U(s,\pi(s),s') + V_\pi(s')]$
  - Q 函数：在状态 s 处采取行动 a，后续仍采用策略 π 的期望收益，与当前状态、当前采取的行动和策略有关
    - $Q_\pi(s,a) = \sum_{s' \in \mathbf{S}} T(s,a,s')[U(s,a,s') + V_\pi(s')]$

- **策略改进**算法:
  - 把状态 s 所有可能的行动都尝试一遍，找到期望奖励最大的行动 a，用来更新策略 π
  - 更新后的策略就是在 s 处选取收益最大的行动 a

  策略改进算法

  - 输入一个策略 π，输出一个更新的改进策略 π_{new}，充分利用马尔科夫性
  - 对任意状态 s，执行下述操作:
    - 对不同的行动 $a \in Action(s)$，计算 $Q_\pi(s,a)$;
    - 更新 $\pi_{new} = \arg\max_{a \in Action(s)} Q_\pi(s,a)$

- **策略迭代** & **值迭代**:

策略迭代算法

计算最优的策略

- $\pi \leftarrow$ 任意初始化值
- Repeat T_1 次（或者 π 不再变化为止）:
  - 评估策略，计算 V_π;
  - 执行策略改进算法，得到 π_{new}
  - $\pi \leftarrow \pi_{new}$

值迭代算法

计算最优的策略

- 对所有状态 s 初始化 $V^0_{opt}(s) \leftarrow 0$;
- for $t = 1, 2, \ldots, T_0$ 次:
  - 对每一个状态 s，执行:
    - $V^t_{opt}(s) \leftarrow \max_{a \in Action(s)} \sum_{s'} T(s, a, s')[Reward(s, a, s') + V^{t-1}_{opt}(s')]$

- 折扣因子:

MDP：引入折扣因子的变化

- 递推方程：$V_\pi = \sum_{s'} T(s, a, s')[Reward(s, a, s') + \lambda V_\pi(s')]$
- $\lambda = 1$ 的特殊情形

ch9 强化学习

- MDP & RL

MDP	强化学习
转移概率和奖励已知，离线	转移概率和奖励未知，在线
决策判断可以仿真	需要花费代价去搜索未知的转移概率和奖励
f 已知	f 未知

- RL
  - 问题描述
    - 已知：序列 $s_0, a_1, r_1, s_1, a_2, r_2, \ldots, a_n, r_n, s_n$，其中 $s_i, i = 0, 1, \ldots, n$ 表示状态，$a_i, i = 1, 2, \ldots, n$ 表示行动，$r_i, i = 1, 2, \ldots, n$ 表示奖励
    - 求解：给每个状态确定一个“最佳行动”，即找到最优策略
  - 基于模型的**蒙特卡洛算法**，对于数据量要求大，要求样本数据满足独立性假设

蒙特卡洛方法：出现频率代替概率

- 从已知数据中任何状态 s 开始，(s,a,r,s') 视为“一个/一段/一组数据”，原数据序列被分割成 n 段；
- 计数，并计算 $\hat{T}(s,a,s')=\frac{\#(s,a,s')}{\#(s,a)}$，$\hat{U}(s,a,s')=\frac{\sum_{r\ in\ (s,a,r,s')} r}{\#(s,a,s')}$
- 用 $\hat{T}(s,a,s')$ 近似估计 $T(s,a,s')$，$\hat{U}(s,a,s')$ 近似估计 $U(s,a,s')$

- Model-free 方法：
  - 假设采用该策略得到一个历史轨迹，用轨迹中的 (s,a) 组带来的收益均值来估计 $Q_\pi(s,a)$

    执行策略 π，得到一条随机路径

    - 已知数据：$s_0,a_1,r_1,s_1,a_2,r_2,\ldots,a_n,r_n,s_n$
    - 定义引入折扣因子的时刻 t 的收益：$u_t=r_t+\lambda r_{t+1}+\lambda^2 r_{t+2}+\ldots$
    - 用 u_t 的均值估计/当成 $Q_\pi(s,a)$：即将 u_t 按 (s,a) 不同取值分组，然后求组内均值。
  - 对任意时刻 t，对应数据段 (s,a,r)：
    - 计算出 (s,a,u_t)
    - 令：$\xi=\frac{1}{(s,a)\text{更新次数}+1}$
    - 更新：$Q_\pi(s,a)=(1-\xi)Q_\pi(s,a)+\xi u_t$

ch11 机器学习概念

- 定义：对于现实世界的一个过程 / 机制 / 方法的抽象 f，从训练数据中归纳出一个学习器，该学习器实现了函数 f 的一个近似 h，这就是模型 / 假设
- 建模过程分为两步：选择模型、训练
- 评价 h 的优劣时有以下指标：
  - **准确性**：h,g 之间的近似度
  - **复杂性**：h 的描述长度，应保证在合理有效范围内以得到可满足的时空代价
  - **完备性**：h 是否对 f 的每个输入都定义了一个响应

ch12 线性回归

$h(x)=\mathbf{W}\cdot\mathbf{x}+b\Rightarrow h(x)=\mathbf{W}\cdot\phi(x)$

- 完整的线性函数应该包括常数项 b
- 但是，我们把输入变量 $\mathbf{x}$ 做一个简单的变换，$\mathbf{x}=(x_1,x_2,\ldots,x_n)^t\longrightarrow\mathbf{x}=(x_1,x_2,\ldots,x_n,1)^t$，就可以消去线性函数中的常数项 b
- 更一般地，我们将输入变量 x，做一次变换，即 $x\longrightarrow\phi(x)=(\phi_1(x),\phi_2(x),\ldots,\phi_n(x))^t$，不管 x 是几维变量，经过一次变换，成为 n 维向量，我们称变换后的结果为“输入特征向量”，该变换称为“特征提取函数”。

h 的获得

从评价标准：准确性出发

- 从分析事情要达到的目标出发，反向思考，我们应怎么去找 h
- 我们称 $h(x)$ 和 $f(x)$ 在某个输入 x 上值不一样，为“误差”或“损失”
  - 绝对损失：$loss_1(h,f,x)=|f(x)-h(x)|$，适用于连续 y 值情形
  - 平方损失：$loss_2(h,f,x)=(f(x)-h(x))^2$，适用于连续 y 值情形
  - 计数损失：$loss_0(h,f,x)=1[f(x)\neq h(x)]$，适用于离散 y 值情形

线性 h 的“一行”损失函数

- 所谓“一行”是指用表格描述 h，一行代表 x 的某一个取值；有时也称一个数据的损失。
- h 为线性函数时，损失函数如下：
  - 绝对损失：$loss_1(\mathbf{W},f,x)=|f(x)-h(x)|=|\mathbf{W}\cdot\phi(x)-f(x)|$
  - 平方损失：$loss_2(\mathbf{W},f,x)=(f(x)-h(x))^2=(\mathbf{W}\cdot\phi(x)-f(x))^2$
  - 计数损失：$loss_0(\mathbf{W},f,x)=1[f(x)\neq\mathbf{W}\cdot\phi(x)]$

训练集上的所有损失之和

- 实用的评价标准
  - $TrainLoss(D_{train},\mathbf{W})=\frac{\sum_{(x,y)\in D_{train}}loss(x,y,\mathbf{W})}{|D_{train}|}$
  - 也称为 “训练误差”

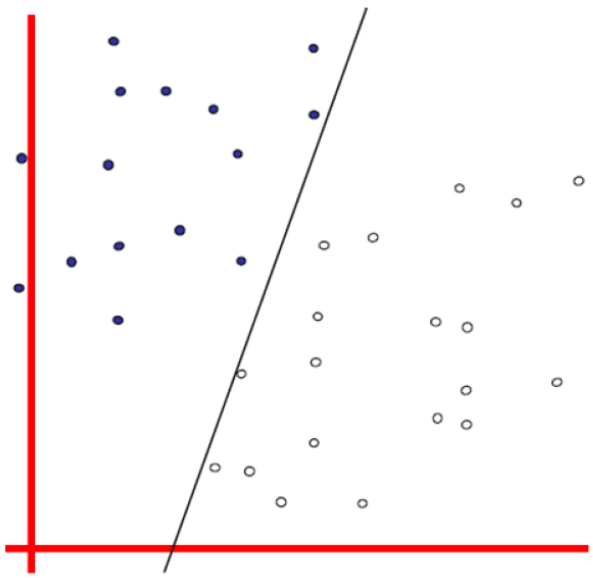
测试集上的所有损失之和

- 也称“经验风险”
  - $TestLoss(D_{test},\mathbf{W})=\frac{\sum_{(x,y)\in D_{test}}loss(x,y,\mathbf{W})}{|D_{test}|}$
  - 通常用测试集上的所有损失来作为 h 的评价标准。

ch13 线性分类

- 在分类问题中，线性分类器给出一个评分 score，需要一个离散化过程将 score 转换为类别标签
-

分类的几何意义



解释说明

- 考虑图示二维的情形：$\mathbf{x} = (x_1, x_2)$ $y \in$ {实心点，空心点}
- 直线方程 $ax_1 + bx_2 + c = 0$，可扩展到 d 维空间 $\mathbf{W} \cdot \mathbf{x} + b = w_1 x_1 + w_2 x_2 + \ldots + w_d x_d + b$
- 三维时，是平面，$d(> 0)$ 维时称为“超平面”
- 直线上的点满足 $ax_1 + bx_2 + c = 0$，实心点满足 $ax_1 + bx_2 + c > 0$，空心点满足 $ax_1 + bx_2 + c < 0$
- h 的线性表达式为 0 时，就是该超平面，也称“决策边界/分类器”：$h(x) = sign(W \cdot \phi(x)) =$

$$\begin{cases} +1 & , \text{if } W \cdot \phi(x) > 0 \\ -1 & , \text{if } W \cdot \phi(x) < 0 \\ * & , \text{if } W \cdot \phi(x) = 0 \end{cases}$$

- 以二分类问题为例
  - **分数** $score = W \cdot \phi(x)$ 描述了模型对预测结果的自信程度，其绝对值越大表示模型越自信
  - **间隔** $margin = W \cdot \phi(x) y$ 表示预测结果的正确性有多好，负值表示预测错误
    - 这一定义比符号函数保留了更多信息，它可以视作平面外点到平面距离的 $\| W \|$ 倍

- 基于间隔 margin 定义损失函数，问题变成最大化 margin 值
  - 硬判决：符号函数，无法使用梯度下降
  - 软判决：可以使用梯度下降
    - logistic 回归：$loss_{logistic}(x, y, W) = log(1 + e^{-W \cdot \phi(x) y})$
    - hinge 损失：$loss_{hinge}(x, y, W) = max(1 - W \cdot \phi(x) y, 0)$

- **SVM**
  - 代数间隔
    - $a - margin \triangleq \mathbf{W} \cdot \phi(x) y$，或
    - $a - margin \triangleq (\mathbf{W} \cdot x + b) y$
  - 几何间隔：数据点到分类平面的距离
    - $g - margin \triangleq \frac{\mathbf{W} \cdot \phi(x) y}{\|\mathbf{W}\|}$
    - $g - margin \triangleq \frac{(\mathbf{W} \cdot x + b) y}{\|\mathbf{W}\|}$
    - 从 $hinge$ 损失函数的定义看，我们需要让代数间隔大于等于 1，使得 $hinge$ 损失为最小值 0；
    - 即 $loss_{hinge}(x, y, \mathbf{W}) = max\{1 - \mathbf{W} \cdot \phi(x) y, 0\} = 0, \Rightarrow \mathbf{W} \cdot \phi(x) y \geq 1$
    - 用几何间隔来描述，即 $\frac{\mathbf{W} \cdot \phi(x) y}{\|\mathbf{W}\|} \geq \frac{1}{\|\mathbf{W}\|}$，即任何一个数据点，它到分类面的距离都大于等 $\frac{1}{\|\mathbf{W}\|}$
  - **支持向量**：在所有数据点中，有些点离决策边界 / 超平面距离最短，超平面可以由这些点来确定，这些点称为支持向量（SV）

- 随着分类面发生变化，支持向量也会发生改变。所以非支持向量实际是一种约束，它使得模型在试图改变分类面时，面对新的支持向量考虑新的最大间隔距离

  线性 SVM 定义

  线性 SVM 优化问题：适用于线性可分的数据集/问题

$$\min\|\mathbf{W}\| \Longleftrightarrow \min \frac{1}{2}\mathbf{W}\cdot\mathbf{W}$$
$$s.t.$$
$$1-(\mathbf{W}\cdot\mathbf{x}+b)y\leq 0, \forall(\mathbf{x},y)\in D_{train}$$

  解释说明

  - 每个样本/训练数据带来一个约束条件
  - 约束条件表明所有的训练数据都被正确分开 $(\mathbf{W}\cdot\mathbf{x}+b)y\geq 1>0$
  - 且分开得足够远，即代数间隔 >1

- 解决线性 SVM 问题：拉格朗日乘子法、SMO算法，过程略

- 回归问题其他损失函数：

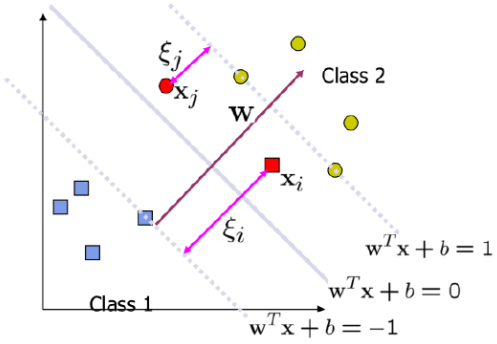
  平滑 L1 损失：降低 L1 损失函数中离群点的影响

$$loss_1'(y,\hat{y})=\begin{cases}0.5(y-\hat{y})^2 & if|y-\hat{y}|<1\\ |y-\hat{y}|-0.5 & 其它\end{cases} \tag{1}$$

  - 交叉熵：适用于分类问题
    - 函数 $softmax(\mathbf{v})$：将一个实数向量 $\boldsymbol{v}=(v_1,v_2,\ldots,v_n)$ 转化为一个概率分布 $\boldsymbol{Q}=(e^{v_1},e^{v_2},\ldots,e^{v_n})\frac{1}{\sum_{i=1}^n e^{v_i}}$，或者视为“归一化”操作；
    - 交叉熵：计算 $\boldsymbol{Q}$ 和某个真实分布 $\boldsymbol{P}$ 之间的差异，$H(\boldsymbol{P},\boldsymbol{Q})=-\sum_{i=1}^n \boldsymbol{P}_i\log(\boldsymbol{Q}_i)$，$\boldsymbol{P}$ 是训练数据集中各类数据的比例/概率，$\boldsymbol{Q}$ 是分类器算出的输入数据属于各类的“概率”（来自 $softmax(\cdot)$ 对算出的各种分数值进行的“归一化”）。

ch14 非线性处理

- 非线性 SVM
  - **松弛变量法**：



- 约束条件：$\forall(x_i, y_i) \in D_{train}$
  $1-(W\cdot x+b)y \le 0 \Longrightarrow$
  $1-[(W\cdot x_i+b)y_i+\xi_i] \le 0$
- 目标函数：
  $\Longrightarrow \min_{W,\xi}(\frac{1}{2}W\cdot W+C\sum_i \xi_i)$
- 训练误差：原来线性可分时，误差为 0$\Longrightarrow \sum_i \xi_i$

松弛变量法

- 如图所示，假设真实/我们想要的分类面如灰实线所示，x_i 错分了，“越界”了，x_j “调皮”或因为噪声，跑到“禁区”里了
- 添加 ξ_i, ξ_j 把 x_i, x_j 给“拉回”各自的“支持向量”边界。
- 我们称 ξ_i, ξ_j 为松弛变量

SVM 中训练误差的产生

- 被平移了的数据点，虽然成了 sv，但是它们带来了训练数据的误差/损失，非支持向量带来 0 误差/损失
- 两类支持向量，它们的拉格朗日乘法因子 $\alpha_i > 0$
- 一类 $\xi_i = 0$，它们是真正的 sv，损失/误差为 0
- 一类 $\xi_i > 0$，它们是平移后的 sv，损失/误差为一类 ξ_i
- 当获得最优解时，$\xi_i = 1-(W\cdot x_i+b)y_i$，这就是样本 (x_i, y_i) 的 $hinge$ 损失

正则化：引入新的优化目标

- $\min_{W,\xi}(\frac{1}{2}W\cdot W+C\sum_i \xi_i)$，也即 $\min_{W,\xi}(\frac{1}{2C}W\cdot W+\sum_i loss_{hinge})$
- 其中原线性 SVM 的优化目标 $\min_W \frac{1}{2}W\cdot W$ 被称为正则化项，训练集上的 $hinge$ 损失 $\sum_i \xi_i$ 称为训练误差最小化项
- 新优化目标既考虑了训练误差最小，又考虑控制模型复杂性（正则化项，也可以说是结构风险最小化）
- 参数 C，用于平衡两个优化目标之间的相对重要性。

- **核函数**：
  - 特征提取函数可以把在低维空间中线性不可分的数据集映射到高维空间，可能实现线性可分
  - $K(x_i, x_j)$ 为核函数，等于 x_i, x_j 两个变量升维后的特征向量的内积，是高维特征空间的内积计算公式
  - 直接获得核函数的值而不关心特征提取过程
  - 常用的，可供我们选择的核函数有：
    - $K(x, a)=(1+x\cdot a)^2$ 多项式核函数
    - $K(x, a)=x\cdot a$ 内积核函数
    - $K(x, a)=e^{-\frac{\|x-a\|^2}{2\sigma^2}}$ 高斯核函数

- KNN：

KNN: 算法描述

训练过程

- 将所有已知样本存储起来。

预测/分类过程: 在给定距离定义下，

- 计算新数据/未知类别数据的 K 个最近邻居
- 以 K 个邻居中占多数的类标号作为新数据的类标号

算法评述

- 优点:
  - 训练简单，基本是 0 代价
  - 可解释性好
- 缺点
  - 预测代价大，找最近的 K 个邻居，算法时间复杂度不低
  - 样本数目太少时，易过拟合，泛化能力不足

- 非参数方法
  - 参数的个数随样本数目增加而增加
  - 参数可以理解为决策边界的细节；样本越多，参数越多，分类器复杂性越高。
  - 想象一下，完整映射表几乎快填完整了，此时再来一个新数据，判定错误的可能性，直觉上就会觉得变小了。
  - KNN 是非参数方法

参数方法

- 相对应的参数方法，参数个数是固定的
- 如线性分类器，权向量 $\mathbf{W}$ 的维数不会因样本增加而增加。
- 通常，参数方法中，样本越多，参数设置得会更合理。

- 过拟合与泛化能力
  - 过拟合: 定义在训练数据集上，训练误差非常小的情况。可能造成的原因: 模型太复杂，太“迁就特殊的训练数据”，而训练数据本身可能因为“噪声”就存在不精确的问题。
  - 泛化能力: 无法在所有数据上计算误差，折衷利用测试集上的误差，该误差和训练误差接近即认为是泛化能力好。

模型/假设的复杂性和正则化

- 完整映射表具有最大描述复杂性，KNN 的复杂性随样本增加而增加;
- 正则化可以认为是人为“强行”在模型中添加“预定义”的知识，以此来控制描述复杂性。比如 KNN 算法扩展 2

ch15 神经网络

- 假设类

  假设类

  考虑相同 $\phi(\cdot)$ 的所有 $\mathbf{W}$ 可以屏蔽 $\mathbf{W}$ 带来的影响

  - 假设类：$\mathcal{H}_\Phi = \{h_\mathbf{W} : \mathbf{W} \in R^d\}$，所有满足下列条件的假设/模型 h 的集合：
    - $\phi(\cdot)$ 相同
    - $\mathbf{W}$ 不同，即参数不同
  - 用假设类中使得 $h(x)$ 性能最好的 $\mathbf{W}$ 来代表 $\phi(\cdot)$ 的性能

  假设类的例子

  - $$x \in \mathbb{R}, y \in \mathbb{R}$$
    $$\text{Linear function: } \Phi(x) = x$$
    $$\mathcal{H}_1 = \{x \to w_1 x : w_1 \in \mathbb{R}\}$$
    $$\text{Quadratic functions:} \Phi(x) = [x, x^2]^t$$
    $$\mathcal{H}_2 = \{x \to w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 \in \mathbb{R}\}$$

  特征提取函数设计的通用方法

  - 定义输入 x 的各个单项式构成的向量为特征 $\phi(x)$
  - 存在问题：单项式的最高次数难以确定，太高会过拟合，增加计算代价；太低可能不能很好地处理复杂的非线性决策边界

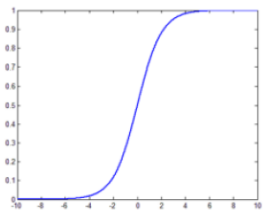
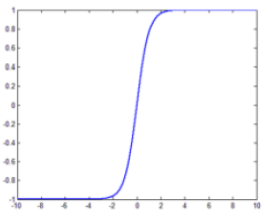
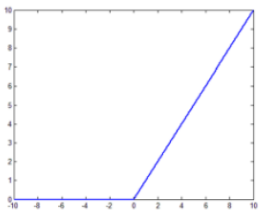
- 激活函数

  激活函数

  - 对分数/score 做一种非线性映射/变换，增加非线性处理能力，是对生物机制的模拟。

  常见的激活函数 [1]

  - Sigmoid　　Tanh　　Rectified Linear

- 若将激活函数加入，得到下图

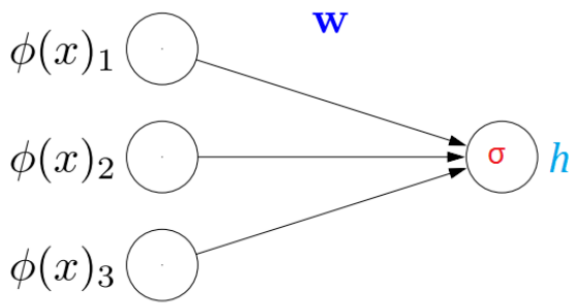
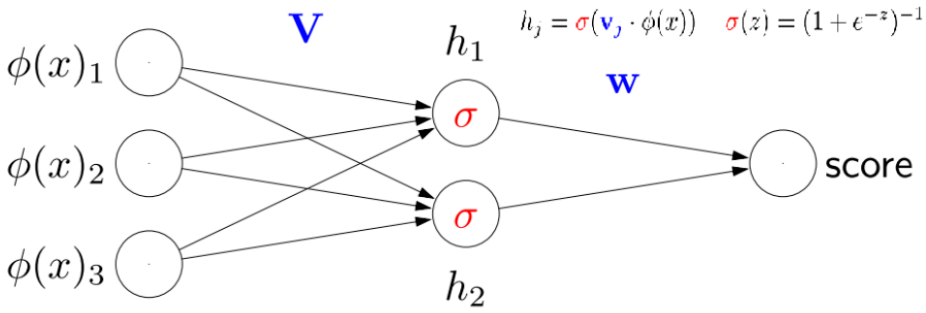


Figure: “激活的”线性函数 $h = \sigma(score) = \sigma([\phi(x_1), \phi(x_2), \phi(x_3)] \cdot W)$

- 神经网络

基本概念：NN

- 网络结构: 输入层、隐层、输出层，权值，隐层单元/节点
- 激活函数
- 训练信号/样本数据
- 隐层输出与数据特征



权值学习

导数/偏导数的意义

- 输入 in 的微小改变是如何影响输出 out 的？一种“敏感性分析”
- 当 in_1 发生改变 ϵ 时，输入 out 发生的改变为 $\frac{\partial out}{\partial in_1}\epsilon$
- 用图形来描述导数计算，如下图 $(out = function(in_1, in_2, in_3))$。边上的绿色字体标记导数，孩子节点是输入变量。

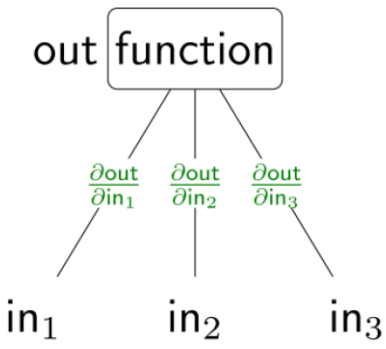
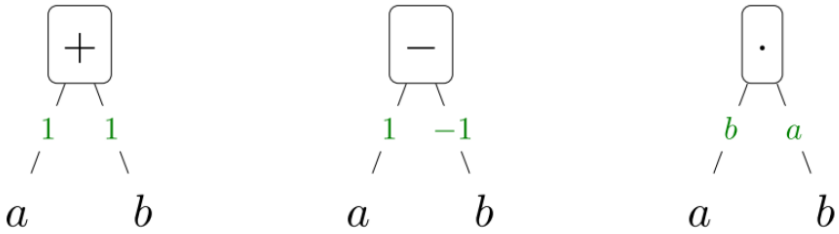
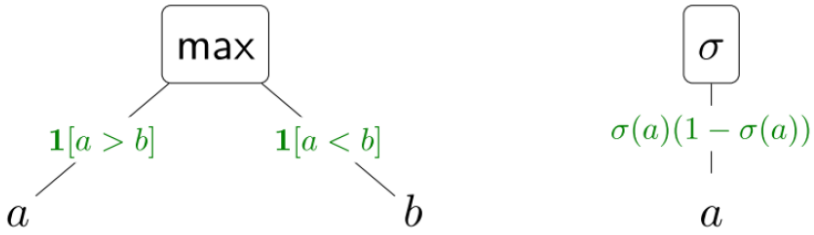


Figure: 导数计算的图形化描述

常见函数的偏导数图形表示





- 通过链式法则求其自变量的偏导，利用这个偏导信息进行梯度下降优化
- 在函数的导数关系树上，先将数据代入叶子节点向上传递获得输出，此过程称为前向过程；然后将误差值从根节点向下反馈，此过程即是后向传播

- 深度学习
  - 适用于深度神经网络的权值学习方法：
    - 神经网络用于特征提取，隐层是一个特征提取函数
    - 权值学习方法的改变：自编码器实现隐层代表的特征提取函数

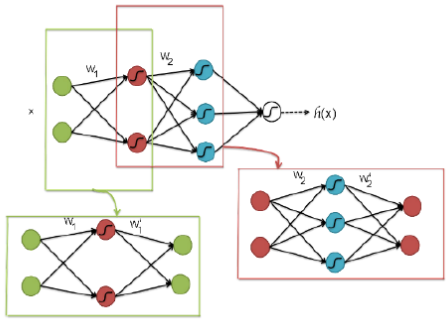


Figure: 例子：深度神经网络学习方法的改进

特征提取的核心思想：自编码器

- 每个隐层独立训练，训练一个三层 BP 网络；
- 对任意一个三层 BP 网络：输入层是前一个隐层的输出（或原始输入数据），输出层与输入层完全一样；训练该三层 BP 网络；
- 丢弃输出层及其关联边；
- 重复上述过程，进行多次，得到多个连续的隐层；这个循环过程构建了一系列的自编码器。
- 最后，利用“监督信号”训练最后一个输出层连边的权值。

新权值学习的特点

特征提取不需要“监督信号”

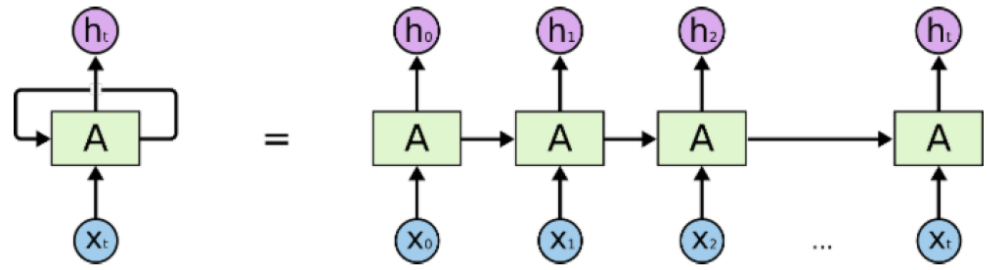
- 现实中，训练数据/类标签通常难以获得；有大量的数据没有标签（即有 x，没有 y），该方法的特征提取过程可以充分利用大量没添加标签的数据。（分类就是为了给数据自动添加标签）

训练代价相对于整体的梯度下降要小很多

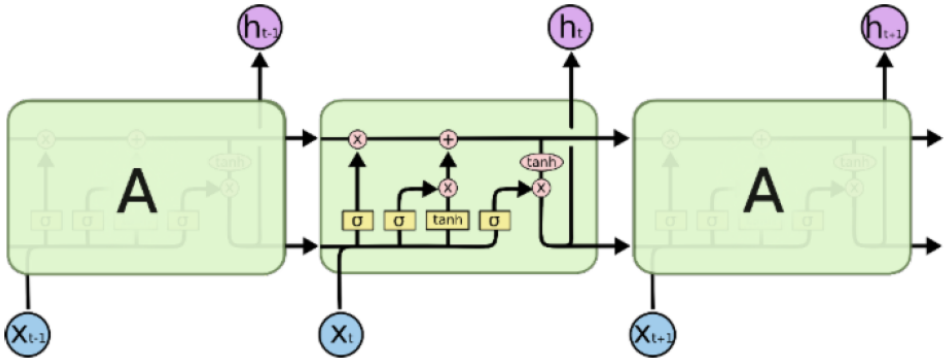
- 虽然每个三层 BP 网络的训练还是比较费时间，相对于多层的 BP，时间代价已经降低了。

隐藏的约束条件

- 假设数据来自某个具有“特点”数据生成器
- 或者说给定的数据集具有“可分类”的特点

- CNN
  - 隐层节点的局部性：每个隐层节点能感受到的信息都是输入信息的某个局部
  - 隐层节点的同一性：隐层节点（**滤波器 / filter / 卷积核**）对输入的加权方式和处理方式都是同样的，**权值共享**
  - 不同的卷积核，对应一类不同的图像特征；每个卷积核作用在输入图像上，就得到一个**特征映射**；特征映射可视为图像的一种变换
  - 参数的个数和卷积核的结构大小，卷积核的个数相关，与输入层节点数、隐层节点数无直接关系
  - 卷积层的层数越高，提取到的特征就越全局化
  - **池化 / Down-pooling / 下采样**：聚合特征、降维，减少运算量
  - 输出层：全连接层，以前面层的输出作为输入，以此构建一个经典的用于分类的神经网络
- RNN
  - 

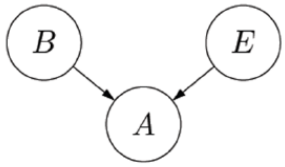
  - 对序列数据的每个数据，RNN 执行相同操作，但是输入和前面的状态或输出相关
  - 存在长期依赖问题，梯度爆炸 / 消失，很难处理时间上很长的历史信息
- LSTM

- 

  - 注意力机制

ch16 贝叶斯网络

- **Bayes 网络的例子**



网络结构

b	$p(b)$
1	ϵ
0	$1-\epsilon$

e	$p(e)$
1	ϵ
0	$1-\epsilon$

边缘分布

b	e	a	$p(a \mid b,e)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

条件概率质量函数

=1: 发生入室盗窃，否则 B=0

=1: 发生地震，否则 E=0

=1: 警铃响了，否则 A=0

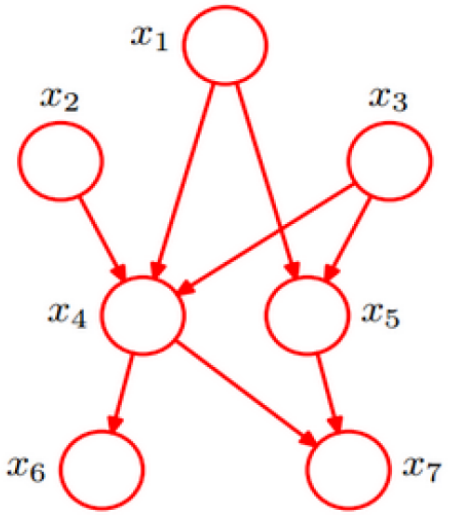
b	e	a	$\mathbb{P}(B=b,E=e,A=a)$
0	0	0	$(1-\epsilon)^2$
0	0	1	0
0	1	0	0
0	1	1	$(1-\epsilon)\epsilon$
1	0	0	0
1	0	1	$\epsilon(1-\epsilon)$
1	1	0	0
1	1	1	ϵ^2

联合概率质量函数

约简：联合概率质量函数 = 网络结构 + 若干边缘分布 + 若干条件概率质量函数

-

从网络结构到乘积公式的例子



解释说明

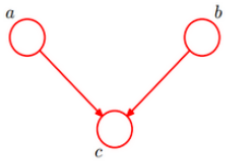
- 开始之前，先思考存储代价降低了多少。
- 看左图，写乘积公式的规律/方法是什么？
- 原理是什么？
  - （条件）概率计算的链式法则
  - （条件）独立性

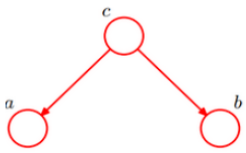
$$p(x_1,\ldots,x_7)=p(x_1)p(x_2)p(x_3)p(x_4|x_1,x_2,x_3)p(x_5|x_1,x_3)p(x_6|x_4)p(x_7|x_4,x_5)$$

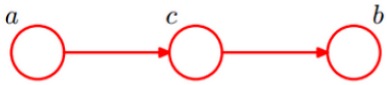
- 链式法则：用条件概率和边缘概率计算联合概率的通用方法

$$\begin{aligned}&p(x_1,\ldots,x_n)=p(x_n|x_1,\ldots,x_{n-1})\cdot p(x_1,\ldots,x_{n-1})\\&=p(x_n|x_1,\ldots,x_{n-1})\cdot p(x_{n-1}|x_1,\ldots,x_{n-2})\cdot p(x_1,\ldots,x_{n-2})\\&\ldots\\&=p(x_n|x_1,\ldots,x_{n-1})\cdot p(x_{n-1}|x_1,\ldots,x_{n-2})\cdot\ldots\cdot p(x_1)\end{aligned}$$

- **条件独立性**





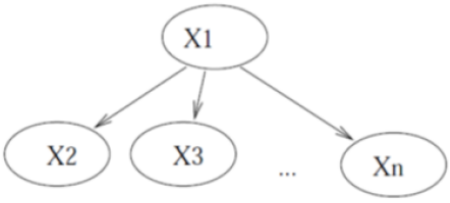


条件独立

- 先写出各个图形表示的联合概率质量函数的乘积表达式，然后用后面的条件独立公式来证明/约简链式法则的结果。
- 情形 1：$p(a,b,c)=p(a)p(b)p(c|a,b)$，因为 a,b 在 c 未知的条件下独立：$p(a,b)=p(a)p(b)$，绝对独立。
- 情形 2：$p(a,b,c)=p(c)p(a|c)p(b|c)$，因为 a,b 在 c 已知/给定的条件下独立：$p(a,b|c)=p(a|c)p(b|c)$
- 情形 3：$p(a,b,c)=p(a)p(c|a)p(b|c)$，因为 a,b 在 c 已知/给定的条件下独立：$p(a,b|c)=p(a|c)p(b|c)$
- 一句话：网络结构图中，不直接连边的节点之间条件独立

-

朴素 Bayes 模型



解释说明

- 如图所示的特殊网络结构，其中 X_1 一般称为“cause”或类标签，其他的 $X_i, i > 1$ 称为观测值，现象，效果等；
- 联合概率计算公式为:
  $p(X_1, X_2, \ldots, X_n) = p(X_1) \cdot p(X_2|X_1) \cdot \ldots \cdot p(X_n|X_1)$
- 朴素 Bayes 模型的应用：分类或推理，就是已知 $X_2, X_3, \ldots, X_n$，求 X_1；在概率论框架下，我们转换为求 $p(X_1|X_2, \ldots, X_n)$，计算概率值!

- Bayes 网络的学习
  - 网络结构的确定
    - 依靠专家建模
    - 从数据集中自动学习网络结构
      - 利用卡方 / 互信息等做相关性测试（独立性检测）
      - 基于搜索-评分的方法
  - 概率质量函数的确定
    - 蒙特卡洛逼近

求概率质量函数例子—单变量

问题描述

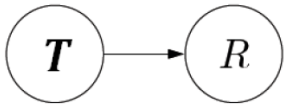
- 数据集：$D = \{1, 5, 4, 4, 3, 5, 5, 4, 4, 4\}$，表示对一个餐馆的打分；
- 随机变量 $R \in \{1, 2, 3, 4, 5\}$ 表示餐馆的分数，求 $(p_1, p_2, p_3, p_4, p_5)$

求解过程 蒙特卡罗逼近:

- 分别统计每个分数出现的次数 c_i 以及数据总数 c
- 得到 $(p_1, p_2, p_3, p_4, p_5) = (c_1/c, c_2/c, c_3/c, c_4/c, c_5/c) = (0.1, 0, 0.1, 0.5, 0.3)$，即用频度近似概率值

求概率质量函数例子—两变量

$p(t,r) = p(t)p(r|t)$



给定的网络结构

问题描述

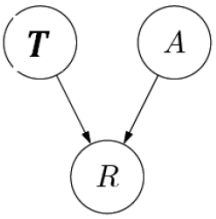
- 数据集：$D = \{(c,4),(c,4),(c,5),(s,1),(s,5)\}$
- 随机向量 $X = \{T, R\}$，随机分量 $R \in \{1,2,3,4,5\}$ 表示对一个餐馆的评分，随机分量 $T \in \{c, s\}$ 表示餐馆类型，假设类型是决定餐馆评分的唯一标准（简化）

求解过程

- 统计频度当边缘概率 $p(T) = (0.6, 0.4)$
- 令 $T = c$ 或 s 时，分别统计各个打分的频度，得到条件概率 $p(R|T) = ((0,0,0,0.67,0.33),(0.5,0,0,0,0.5))$

求概率质量函数粒子—三变量

$p(t,r,a) = p(a)p(t)p(r|t,a)$



给定的网络结构

问题描述

- 数据集：$D = \{(c,0,3),(c,1,5),(s,0,1),(s,0,5),(s,1,4)\}$
- 随机向量：$X = \{T, A, R\}$，A 表示是否获得认证，其他变量含义如前所述

求解过程

- 统计频度当边缘概率 $p(T) = (0.4, 0.6), p(A) = (0.6, 0.4)$
- 对 (T, A) 的任意给定的组合取值，统计 R 的出现频度当成条件概率 $p(R|T,A) = ((0,0,1,0,0)_{c0},(0,0,0,0,1)_{c1},(\ldots)_{s0},(\ldots)_{s1})$

拉普拉斯平滑

问题描述

- 条件概率表中存在很多不合理的估计值 0，原因是样本数量太少。如何处理？

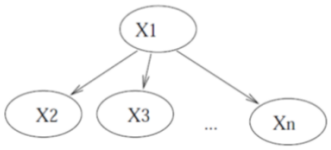
解决办法：拉普拉斯平滑

- 去掉这些 0 的方法称为平滑技术
- 若在统计每个值出现次数时，计数器的初始值设置为非零，这就是所谓的“拉普拉斯平滑”
- 通常设置计数器初始值为 1
- 如右图的例子，数据集为 $D = \{(d,4),(d,5),(c,5)\}$

x_1	x_2	$p_2(x_2 \mid x_1)$
d	1	1/7
d	2	1/7
d	3	1/7
d	4	2/7
d	5	2/7
c	1	1/6
c	2	1/6
c	3	1/6
c	4	1/6
c	5	2/6

x_1	$p_1(x_1)$
d	3/5
c	2/5

求概率质量函数例子—朴素 Bayes 网络



给定的 Bayes 网络结构

问题描述

- 给定数据集 D 和 Bayes 网络结构如图，求边缘/条件概率质量函数/表。

求解过程

- 首先在数据集中统计 X_1 各种取值出现的频度，并以之替代概率，得到 $p(X_1)$；
- 依据 X_1 的不同取值，将数据集 D 分为若干组，同组 X_1 的值相同；
- 统计每组内 $X_i, i > 1$ 的各种取值出现的频度，得到条件概率质量函数 $p(X_i|X_1)$

- 隐变量学习：EM 算法