DS4001-25SP-HW3：贝叶斯网络

常征　PB23030850

2025 年 5 月 30 日

1　马尔可夫链 [24%]

1.1　回答问题 [20%]

1.1.1　手动计算 [4%]

1. $P\{$ 阴天、下雨、晴天、晴天、晴天、阴天、下雨、下雨 $\} = P\{\pi_2 q_{23} q_{31} q_{11} q_{11} q_{12} q_{23} q_{33} = 1.536 \times 10^{-4}\}$
2. $P\{$ 下雨、下雨、下雨、下雨 $\} = P\{\pi_3 q_{33} q_{33} q_{33} = 1.92 \times 10^{-2}\}$
3. **预计**我就理解为期望了，不妨设连续下雨的天数为 T, 则 $P(T = k) = p_{33}^{k-1} = (0.4)^{k-1}$
   那么 $\mathrm{ET} = \sum_{k=1}^{\infty} kP(T = k) = \sum_{k=1}^{\infty} k * (0.4)^{k-1} = 1.67$

1.1.2　代码填空与参数估计 [16%]

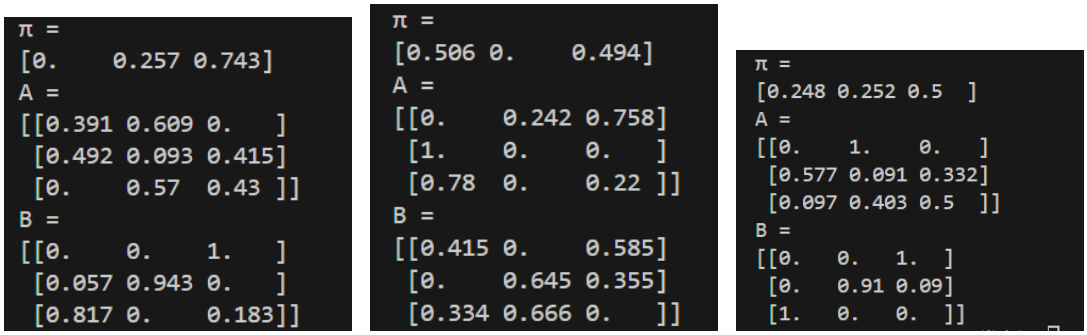
这一题的难点在于我之前没有怎么接触过 numpy，以至于不知道怎么进行切片（虽然另一门课要用，但是不用进行数据切片）。

```
#alpha初始化
alpha[0] = self.pi * self.B[:, seq[0]]
```

```
#alpha递推
alpha[t] = [np.sum(alpha[t-1] * self.A[:, j]) * self.B[j][seq[t]] for j in range(self.n_states)]
```

```
#beta更新
beta[t] = [np.sum(self.A[i] * self.B[:, seq[t+1]] * beta[t+1]) for i in range(self.n_states)]
```

运行得到了三种结果



(a) 0.25, 0.75, 0 (b) 0.5, 0.5 (c) 0.25, 0.25, 0.5

图 1: 运行结果

1.1.3 运行结果对比 [4%]

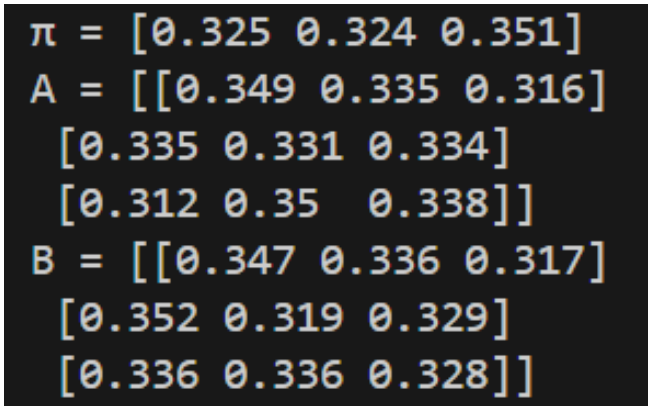


图 2: 吉布斯采样运行结果

我观察到，在本题给定的情境下，吉布斯采样的稳定性和准确度显著优于前向-后向算法。我想原因主要来源于我们对 Baum-Welch 算法的参数估计进行初始化时，随机分配了每一个参数，没有正确的建模。

从稳定性的角度出发，吉布斯采样的稳定性取决于我们的采样量，而 Baum-Welch 算法则很依赖于对模型的正确初始化假设，因此在本题的情境下是吉布斯采样更稳定；从精度上来讲，Baum-Welch 算法是基于极大似然估计，迭代逼近，理论精度更高，但是可能会落在局部最优解，这也说明了为什么几乎总是会收敛到图一的三种情况，而不是任意值。

2 贝叶斯网络 [64%]

2.1 贝叶斯网络：推理 [54%]

2.1.1 精确推理 [16%]

Problem a

```
1    def observe(self, agentX: int, agentY: int, observedDist: float):
2        # BEGIN_YOUR_CODE (our solution is 7 lines of code, but don't worry if you deviate from this)
```

```
        for i in range(self.belief.getNumRows()):
            for j in range(self.belief.getNumCols()):
                x = util.colToX(j)
                y = util.rowToY(i)
                distance = pow(pow(x-agentX, 2) + pow(y-agentY, 2), 0.5)
                prop = util.pdf(distance, Const.SONAR_STD, observedDist)
                self.belief.setProb(i, j, self.belief.getProb(i,j)*prop)
        self.belief.normalize()
        # END_YOUR_CODE
```

~~其实我现在也不太懂为什么原来的概率乘以概率密度函数就变成了条件概率。~~我们可以证明原来的概率乘以概率密度函数就变成了条件概率，证明略去。

总之，对于网格的每一点，我们先利用 `colToX` 或者 `rowToY` 把坐标转换为实际的距离，然后计算该点到小车的距离。进而带入正态分布的概率密度函数，就可以和先验概率作用，得到后验概率。不过要记得正则化。

对于正态分布的概率密度函数，注释里提到 `observedDist: true distance plus a mean-zero Gaussian with standard deviation Const.SONAR_STD` 。并且由正态分布的性质，我们就可以知道 observeDist 实际服从均值为 distance, 方差为 Const.SONAR_STD 的正态分布。

Problem b

```
    def elapseTime(self) -> None:
        # BEGIN_YOUR_CODE (our solution is 5 lines of code, but don't worry if you deviate from this)
        new_blief = util.Belief(self.belief.getNumRows(), self.belief.getNumCols(), 0)
        for ((old, new), trans) in self.transProb.items():
            old_prop = self.belief.getProb(old[0], old[1])
            new_blief.addProb(new[0], new[1], old_prop*trans)
        new_blief.normalize()
        self.belief = new_blief
        # END_YOUR_CODE
```

这实际上就是利用原来的 belief 和概率转移矩阵更新得到新的 belief 的过程。代码第 3 行首先是实例化一个新的 Belief 对象，第 4-6 行就是对于概率转移矩阵的每一个元素，利用其 t 时刻概率和转移概率计算 t+1 概率，最后我们得把新的 Belief 对象替代给原来的 belief。

写报告的时候发现这一步忘记了，居然也能成功到终点，可能这就是 ai 的魅力吧。

2.1.2 模糊推理 [16%]

Problem c

```
    def updateBelief(self) -> None:
        newBelief = util.Belief(self.belief.getNumRows(), self.belief.getNumCols(), 0)
        # BEGIN_YOUR_CODE (our solution is 5 lines of code, but don't worry if you deviate from this)
        for i, (r, c) in enumerate(self.samples):
            newBelief.addProb(r, c, self.weights[i])
        newBelief.normalize()
        self.belief = newBelief
        # END_YOUR_CODE
```

这一个和 Problem b 类似，都是更新 belief。不过不同的是 b 是用转移概率矩阵，而 c 是要求我们使用样本的权重进行更新。

~~因为我懒得看 samples 的定义格式~~我们从已经写好的 observe 方法观察到，循环大概是这样:

`for i, (r, c) in enumerate(self.samples):`

然后访问对应的 weight 就可以了。

Problem d

```
def elapseTime(self) -> None:
    # BEGIN_YOUR_CODE (our solution is 8 lines of code, but don't worry if you deviate from this)
    newSamples = []
    for i, (r, c) in enumerate(self.samples):
        if (r, c) not in self.transProbDict:
            newSamples.append((r, c))
        else:
            newSamples.append(util.weightedRandomChoice(self.transProbDict[(r, c)]))
    self.samples = newSamples
    self.updateBelief()
    # END_YOUR_CODE
```

这一问我们要实现样本转移，并不复杂。不过题目有这样一个要求：if a sample is in a location with no outgoing transitions defined, keep the sample in place (identity transition). 所以就有了我们的 if-else 判断。

2.1.3 粒子滤波 [16%]

Problem e

```
    # BEGIN_YOUR_CODE (our solution is 5 lines of code, but don't worry if you deviate from this)
    self.particles = collections.defaultdict(int)
    all_pos = [(x,y) for x in range(numRows) for y in range(numCols) if (x, y) in self.transProbDict]
    for p in range(self.NUM_PARTICLES):
        pos = random.choice(all_pos)
        self.particles[pos] = self.particles.get(pos, 0) + 1
    # END_YOUR_CODE
```

我觉得 Problem e 没有表述的很清楚（或者说文档和代码注释表述的要求存在矛盾）。文档里的描述是：根据已建立的转移概率字典来从**所有可转移**的位置均匀采样。但是注释里给出的要求是：**randomly distributed across the grid**。从结果来看前者是对的，因为后者会出 Bug(雾)。

为了找到所有可转移的位置，我使用了一个 list：遍历网格的每一个点，判断其在不在 self.transProbDict 的 key 中，实际就是代码的第 3 行。

在找到所有可转移位置之后，按照均匀采样的原则，对于每一个粒子，我们使用 random.choice 来随机选择其位置，并把结果存在 self.particles 中。实际上我们要保存的不是每个粒子的位置，而是每个位置的粒子数。就是第 6-7 行所做的事。

顺带一提，我最开始用的是普通的字典，但是看到后面其他方法助教用的是 collections.defaultdict(int)，就保持一致了。不过我试了以下用普通的字典也不会报错。

Problem f

```
    weight = collections.defaultdict(int)
    for pos in self.particles:
        x = util.colToX(pos[0])
        y = util.rowToY(pos[1])
        dis = math.sqrt((agentX - x) ** 2 + (agentY - y) ** 2)
        prop = util.pdf(dis, Const.SONAR_STD, observedDist)
        weight[pos] = self.particles[pos] * prop
    self.particles = weight
```

Problem f 就是对粒子滤波的重加权过程。大部分操作和 problem a 类似，不过不同的是我们计算出一个新的权重之后不能直接赋给 self.belief。因为后面的代码时利用 self.particles 来完成这件事的，所以我们

要更新 self.particles 的值,变成一个 {pos:weight} 的字典。这样后面就可以调用 `weightedRandomChoice` 方法了。

2.1.4 思考题 [6%]

对于 LikelihoodWeighting,在 init, observe, elapseTime 三个方法的最后都调用了 `self.updateBelief()`,而对于 ParticleFilter，只在在 init, observe, 两个方法的最后都调用了 `self.updateBelief()`。

我们可以看到主要的区别就是在 elapseTime 方法中是否调用 `self.updateBelief()`。

从算法思想的角度来看，LikelihoodWeighting 使用样本的位置和权重来计算和更新 belief，ParticleFilter 直接使用粒子的数量和位置来作为 belief。

对于前者，当样本的位置更新的时候，belief 也需要对应的改变；对于后者，粒子位置的变动天然的就包含了 belief 的改变，无需再次更新

2.2 贝叶斯网络：学习 [10%]

文档和群聊天记录弄得乱七八糟的，没太看懂助教老师的意思。我就按照我的想法算了。

记第 i 轮的结果为 o_i，取硬币 A,B 分别为 x_A, x_B，则有

$$P(o_1|x_A) = \theta_A^{old} \cdot \theta_A^{old} \cdot (1-\theta_A^{old}) \cdot \theta_A^{old} \cdot \theta_A^{old} = 5.184 \times 10^{-2} \tag{1}$$

$$P(o_1|x_B) = \theta_B^{old} \cdot \theta_B^{old} \cdot (1-\theta_B^{old}) \cdot \theta_B^{old} \cdot \theta_B^{old} = 1.536 \times 10^{-2} \tag{2}$$

于是经过归一化，就有：

$$\gamma_{1,A} = \frac{P(o_1|x_A) \cdot \pi^{old}}{P(o_1|x_A) \cdot \pi^{old} + P(o_1|x_B) \cdot (1-\pi^{old})} = 0.771 \tag{3}$$

$$\gamma_{1,B} = \frac{P(o_1|x_B) \cdot (1-\pi^{old})}{P(o_1|x_A) \cdot \pi^{old} + P(o_1|x_B) \cdot (1-\pi^{old})} = 0.229 \tag{4}$$

对于第 2 轮，类似的有：

$$\gamma_{2,A} = \frac{P(o_2|x_A) \cdot \pi^{old}}{P(o_2|x_A) \cdot \pi^{old} + P(o_2|x_B) \cdot (1-\pi^{old})} = 0.229 \tag{5}$$

$$\gamma_{2,B} = \frac{P(o_2|x_B) \cdot (1-\pi^{old})}{P(o_2|x_A) \cdot \pi^{old} + P(o_2|x_B) \cdot (1-\pi^{old})} = 0.771 \tag{6}$$

对于第 3 轮，类似的有：

$$\gamma_{3,A} = \frac{P(o_3|x_A) \cdot \pi^{old}}{P(o_3|x_A) \cdot \pi^{old} + P(o_3|x_B) \cdot (1-\pi^{old})} = 0.6 \tag{7}$$

$$\gamma_{3,B} = \frac{P(o_3|x_B) \cdot (1-\pi^{old})}{P(o_3|x_A) \cdot \pi^{old} + P(o_3|x_B) \cdot (1-\pi^{old})} = 0.4 \tag{8}$$

对于第 4 轮，类似的有：

$$\gamma_{4,A} = \frac{P(o_4|x_A) \cdot \pi^{old}}{P(o_4|x_A) \cdot \pi^{old} + P(o_4|x_B) \cdot (1-\pi^{old})} = 0.229 \tag{9}$$

$$\gamma_{4,B} = \frac{P(o_4|x_B) \cdot (1-\pi^{old})}{P(o_4|x_A) \cdot \pi^{old} + P(o_4|x_B) \cdot (1-\pi^{old})} = 0.771 \tag{10}$$

M 步：

$$\pi^{new} = \frac{1}{4}\sum_{i=1}^{4}\gamma_{i,A} = 0.59275 \tag{11}$$

$$\theta_A^{new} = \frac{1}{\sum_{i=1}^{4}\gamma_{i,A}}(\frac{4}{5}\gamma_{1,A} + \frac{1}{5}\gamma_{2,A} + \frac{3}{5}\gamma_{3,A} + \frac{1}{5}\gamma_{4,A}) = 0.584 \tag{12}$$

$$\theta_B^{new} = \frac{1}{\sum_{i=1}^{4}\gamma_{i,B}}(\frac{4}{5}\gamma_{1,B} + \frac{1}{5}\gamma_{2,B} + \frac{3}{5}\gamma_{3,B} + \frac{1}{5}\gamma_{4,B}) = 0.337 \tag{13}$$

3 贝叶斯深度学习 [6%]

过高置信预测导致危险：比起固定的参数，BDL 通过将参数建模为服从特定分布的随机变量，从而能更精确的量化不确定性 (置信度)。

对分布外数据预测很差：通过贝叶斯推理计算预测变量的边际分布，而非传统模型的单点估计。

易受对抗性操纵：通过参数后验分布的熵约束，限制模型对输入微小扰动的敏感性。

举例：NeurIPS 2020 的一篇名为https://arxiv.org/abs/2006.04591的论文提出了一种基于证据理论的贝叶斯深度学习框架，能够显式建模认知不确定性，从而显著提升模型对分布外数据的检测能力。

其他地方我看不太懂，总之最后的优化目标是：

$$\mathcal{L} = E_{x,y}[-\log p(y|x) + \gamma E] \tag{14}$$

, 其中 E 是不确定度。

体验反馈 [6%]

(a) [**必做**] 10 小时

(b) [**选做**] 这次的框架不太好，写完了都不知道写的对不对。(我知道不是助教的问题，但是就是想吐槽)。比如精确推理我写的是错的也能比较顺利到终点，后面两个很难到终点，也不知道对不对。