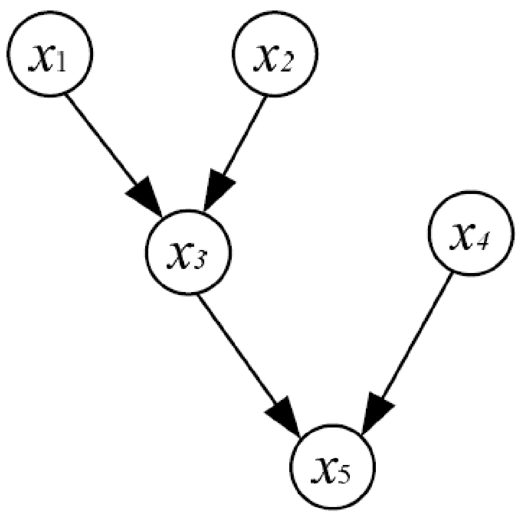
机器学习导论第五次作业

韦俊林 (201928016029023)

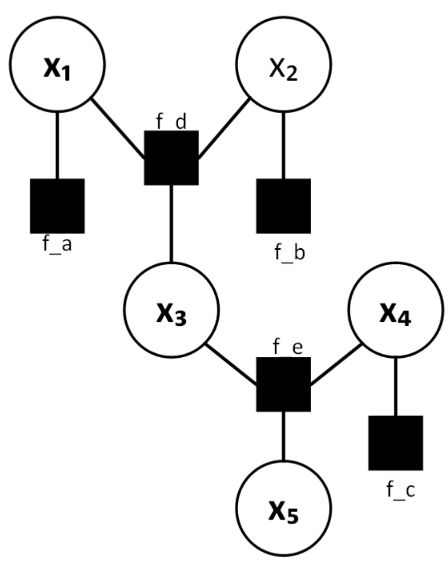
2020 年 5 月 24 日

1 计算与证明题

1. 针对如下概率图模型，试将该有向图转换成因子图，并给出各因子结点对应的因子。利用 sum-product 算法计算 $P(x_3)$ 和 $P(x_4)$（要求写出消息传递的具体步骤）；



答：转换为因子图如下图所示，



其中，

$$f_a(x_1)=P(x_1),\ f_b(x_2)=P(x_2),\ f_c(x_4)=P(x_4)$$

$$f_d(x_1,x_2,x_3)=P(x_3|x_1,x_2),\ f_e(x_3,x_4,x_5)=P(x_5|x_3,x_4)$$

sum-product 算法的消息传递基于两个方法：
变量节点向因子节点传送等于，传入该变量所有信息的乘积;
因子节点向变量节点传送等于，传入该因子所有信息的乘积，再乘上该因子节点相关的因子。
计算 $P(x_3)$，设 x_3 为根节点，此时叶子节点分别为 f_a, f_b, f_c, x_5，
从叶节点向根节点传递信息：

$$\begin{aligned}
\mu_{f_a\to x_1}(x_1) &= f_a(x_1)\\
\mu_{x_1\to f_d}(x_1) &= \mu_{f_a\to x_1}(x_1)\\
\mu_{f_b\to x_2}(x_2) &= f_b(x_2)\\
\mu_{x_2\to f_d}(x_2) &= \mu_{f_b\to x_2}(x_2)\\
\mu_{x_5\to f_e}(x_5) &= 1\\
\mu_{f_c\to x_4}(x_4) &= f_c(x_4)\\
\mu_{x_4\to f_e}(x_4) &= \mu_{f_c\to x_4}(x_4)\\
\mu_{f_d\to x_3}(x_3) &= \sum_{x_1} f_d(x_1,x_3)\sum_{x_2} f_d(x_2,x_3)\mu_{x_1\to f_d}(x_1)\mu_{x_2\to f_d}(x_2)\\
\mu_{f_e\to x_3}(x_3) &= \sum_{x_4} f_d(x_3,x_4)\sum_{x_5} f_d(x_3,x_5)\mu_{x_4\to f_e}(x_4)\mu_{x_5\to f_e}(x_5)
\end{aligned}$$

接着从根节点向叶节点传递信息：

$$\begin{aligned}
\mu_{x_3\to f_d}(x_3) &= 1\\
\mu_{f_d\to x_1}(x_1) &= \sum_{x_2} f_d(x_1,x_2)\sum_{x_3} f_d(x_1,x_3)\mu_{x_2\to f_d}(x_2)\mu_{x_3\to f_d}(x_3)\\
\mu_{x_1\to f_a}(x_1) &= \mu_{f_d\to x_1}(x_1)\\
\mu_{f_d\to x_2}(x_2) &= \sum_{x_1} f_d(x_1,x_2)\sum_{x_3} f_d(x_2,x_3)\mu_{x_1\to f_d}(x_1)\mu_{x_3\to f_d}(x_3)\\
\mu_{x_1\to f_a}(x_1) &= \mu_{f_d\to x_1}(x_1)\\
\mu_{x_3\to f_e}(x_3) &= 1\\
\mu_{f_e\to x_4}(x_4) &= \sum_{x_3} f_d(x_3,x_4)\sum_{x_5} f_d(x_4,x_5)\mu_{x_3\to f_e}(x_3)\mu_{x_5\to f_e}(x_5)\\
\mu_{x_4\to f_c}(x_4) &= \mu_{f_e\to x_4}(x_4)\\
\mu_{f_e\to x_5}(x_5) &= \sum_{x_3} f_d(x_3,x_5)\sum_{x_4} f_d(x_4,x_5)\mu_{x_3\to f_e}(x_3)\mu_{x_4\to f_e}(x_4)
\end{aligned}$$

所以，$P(x_3)=\mu_{f_d\to x_3}(x_3)\mu_{f_e\to x_3}(x_3)$；
同理可得，$P(x_4)=\mu_{f_e\to x_4}(x_4)\mu_{f_c\to x_4}(x_4)$。

2. 给定隐马尔可夫模型，定义在时刻 t 状态为 q_i 的条件下，从 $t+1$ 到 T 的部分观测序列为 $o_{t+1},o_{t+2},\ldots,o_T$ 的概率为后向概率 $\beta_t(i)=P(o_{t+1},o_{t+2},\ldots,o_T|i_t=q_i,\lambda)$。试用概率基本公式

证明后向概率满足如下递推公式：

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j),\ i = 1, 2, \ldots, N$$

$$\beta_T(i) = 1,\ i = 1, 2, \ldots, N$$

答：

$$\begin{aligned}
\beta_t(i) \quad & = P(o_{t+1}, o_{t+2}, \ldots, o_T | i_t = q_i, \lambda) \\
& = \sum_{j=1}^{N} P(o_{t+1}, o_{t+2}, \ldots, o_T, i_{t+1} = q_j | i_t = q_i, \lambda) \\
& = \sum_{j=1}^{N} P(o_{t+1}, o_{t+2}, \ldots, o_T | i_{t+1} = q_j, i_t = q_i, \lambda) P(i_{i+1} = q_j | i_t = q_i, \lambda) \\
& = \sum_{j=1}^{N} P(o_{t+1}, o_{t+2}, \ldots, o_T | i_{t+1} = q_j, i_t = q_i, \lambda) a_{ij} \\
& = \sum_{j=1}^{N} P(o_{t+1}, o_{t+2}, \ldots, o_T | i_{t+1} = q_j, \lambda) a_{ij} \\
& = \sum_{j=1}^{N} P(o_{t+1} | o_{t+2}, \ldots, o_T, i_{t+1} = q_j, \lambda) P(o_{t+2}, \ldots, o_T | i_{t+1} = q_j, \lambda) a_{ij} \\
& = \sum_{j=1}^{N} P(o_{t+1} | i_{t+1} = q_j, \lambda) P(o_{t+2}, \ldots, o_T | i_{t+1} = q_j, \lambda) a_{ij} \\
& = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)
\end{aligned}$$

2 程序设计题

1. 利用 Sarsa 算法和 Q-Learning 算法开发走迷宫机器人

   OpenAI Gym 是一个专注于强化学习的工具包，里面提供了很多游戏、模拟控制的实验环境。在 Gym 的 Tony Text 环境组里，有一个冰湖游戏 FrozenLake-v0。该环境是一个冰冻的湖面，其中有很多冰洞，Agent 的任务是从湖的一点出发到湖面的另外一点。在该过程中如果误入冰洞，则该项任务失败。

   要求：

   1) 编程实现 Sarsa 算法实现 Agent 穿越冰湖，并分析不同学习率和折扣因子下算法的表现；
   2) 编程实现 Q-Learning 算法实现 Agent 穿越冰湖，并分析不同学习率和折扣因子下算法的表现；
   3) 分析并实验对比上述两个算法的性能表现。

【备注】OpenAI Gym 环境的安装与使用

√ OpenAI Gym 环境的安装方法可网上搜索

√ Gym 环境的使用示例代码

import gym

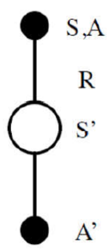
env = gym.make('FrozenLake-v0') # 构建环境

env.reset() # 开始一个 episode

env.render() # 显示初始 environment

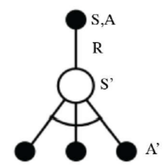
observation,reward,done,info=env.step(env.action_space.sample()) # 随机执行一步

实验报告： Sarsa 算法与 Q-Learning 算法执行的过程都相同，核心区别在于更新 Q 表的算法不同。如下图所示，其中左图为 Sarsa 算法，右图为 Q-Learning 算法；



$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma Q(S',A') - Q(S,A)\right)$$

(a)



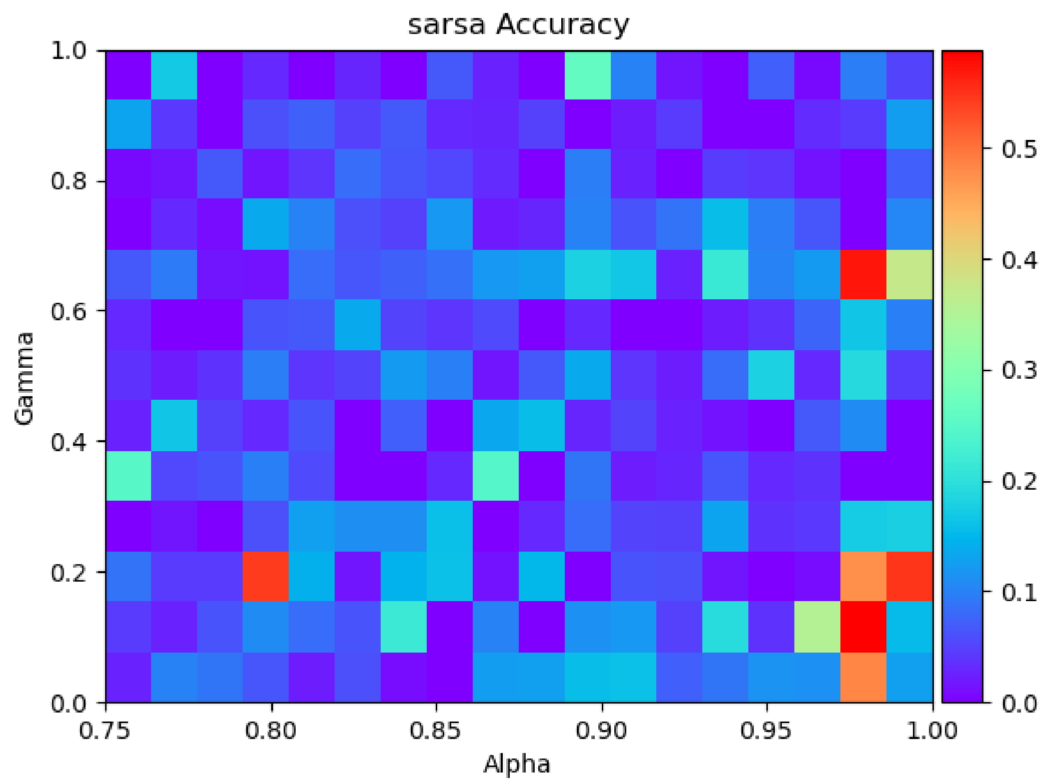
$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$

(b)

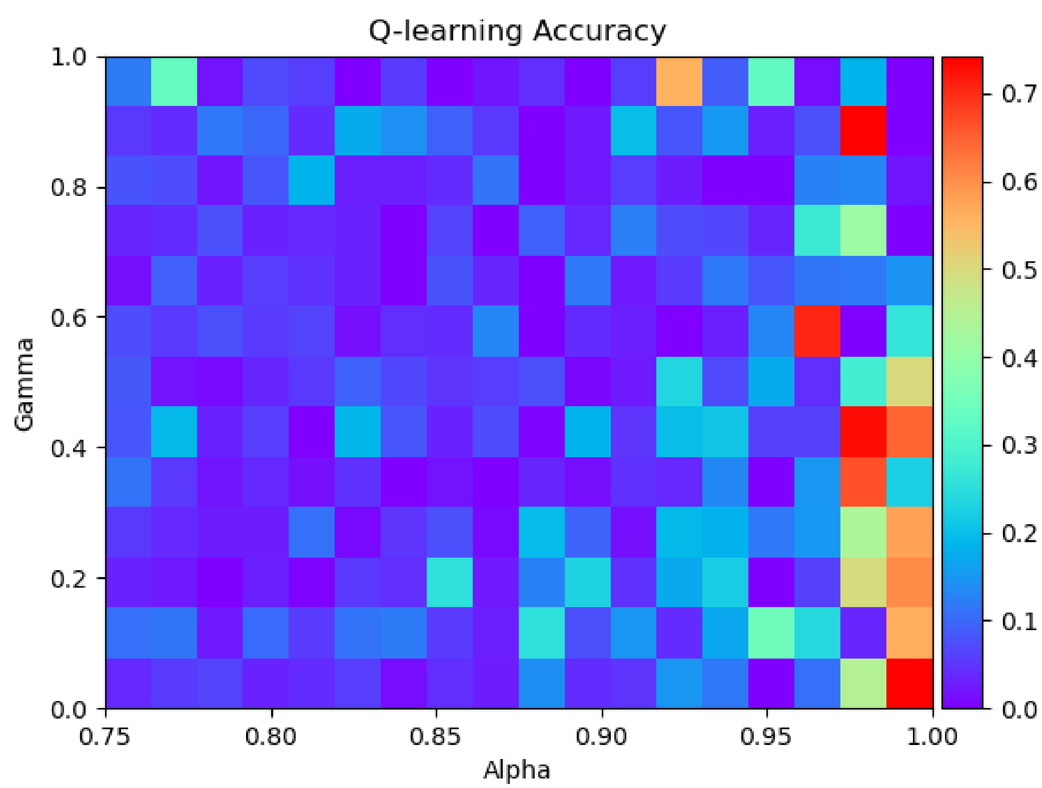
根据上述原理，基于 python 实现上述两个算法代码如下图：

```
# 更新Q表
if alg == 'Q-learning':
    # Q-learning
    Q_all[s, a] = Q_all[s, a] + alpha * (r + gamma * np.max(Q_all[s1, :]) - Q_all[s, a])
    # sarsa
elif alg == 'sarsa':
    a_ = np.argmax(Q_all[s1, :] + np.random.randn(1, env.action_space.n) * (1. / (i + 1)))
    Q_all[s, a] = Q_all[s, a] + alpha * (r + gamma * Q_all[s1, a_] - Q_all[s, a])
```

设定迭代次数为 350 次，在不同学习率 ($Alpha$) 和折扣率 ($Gamma$) 下两个算法的准确率：

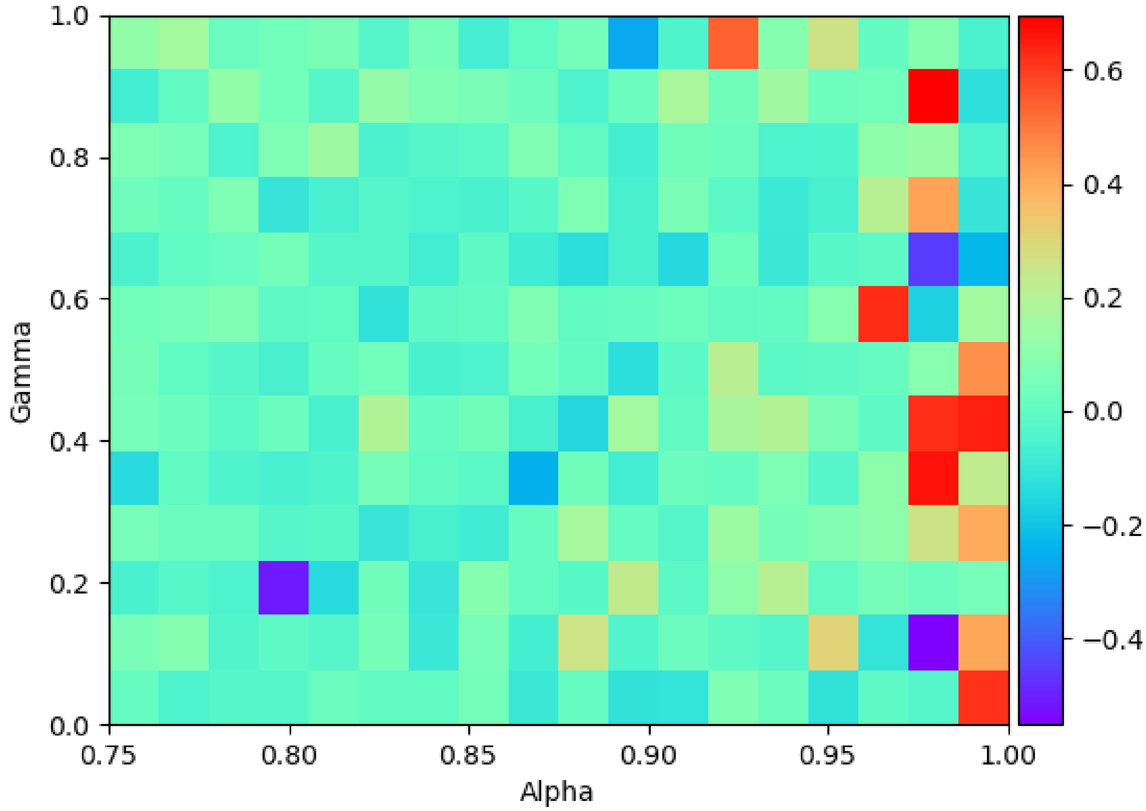


sarsa Accuracy
Gamma
Alpha
0.0
0.2
0.4
0.6
0.8
1.0
0.75
0.80
0.85
0.90
0.95
1.00
0.1
0.2
0.3
0.4
0.5



Q-learning Accuracy
Gamma
Alpha
0.0
0.2
0.4
0.6
0.8
1.0
0.75
0.80
0.85
0.90
0.95
1.00
0.1
0.2
0.3
0.4
0.5
0.6
0.7

在相同超参数的情况下，将 Q-Learning 算法准确率减去 Sarsa 算法准确率：



截取某个超参数情况下 Sarsa 算法与 Q-Learning 算法计算的 Q 表：

```
Score over time: 0.15142857142857144
sarsa: Alpha=0.850000   Gamma=0.800000
[[  1.08582138e-04   3.43333580e-05   1.37128030e-04   1.55073933e-04]
 [  5.02139359e-05   1.26390459e-05   9.14284375e-06   8.86431301e-03]
 [  1.18393748e-02   1.38138474e-05   1.89938252e-03   2.86513237e-06]
 [  1.71426171e-06   2.32981695e-05   1.65278040e-05   1.45375201e-03]
 [  3.89286754e-07   4.46285266e-08   4.63618918e-05   4.79899848e-05]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  1.56297633e-03   5.94588313e-06   5.48434722e-04   1.00518901e-05]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  1.22220835e-04   4.35503887e-07   1.49485278e-03   1.05189001e-04]
 [  2.93365657e-05   1.73312535e-03   0.00000000e+00   2.16666233e-03]
 [  8.89860871e-03   6.05019974e-03   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  4.78383106e-05   0.00000000e+00   0.00000000e+00   1.98848441e-03]
 [  0.00000000e+00   0.00000000e+00   3.20944618e-01   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]]
```

(c)

```
Score over time: 0.5485714285714286
Q-learning: Alpha=0.990000      Gamma=0.650000
[[  7.08552652e-04   2.23791130e-04   2.21222023e-04   2.23731991e-04]
 [  5.29713513e-07   1.32416727e-04   4.38553279e-04   3.40265934e-04]
 [  6.65812292e-04   6.82109453e-04   2.03718324e-04   1.26132208e-04]
 [  1.26558177e-04   1.25324321e-04   1.94724169e-06   1.32757072e-04]
 [  1.09016342e-03   9.58762305e-06   9.57486580e-04   4.75609142e-04]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  3.14616646e-04   1.05682931e-03   1.26171093e-06   1.40264051e-10]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  7.01568699e-04   4.80382366e-05   1.12566760e-07   1.48793352e-03]
 [  2.48166250e-03   3.14938883e-03   4.09968744e-05   1.69757668e-07]
 [  4.17926459e-03   6.39703037e-05   7.70179052e-05   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  2.59018171e-03   0.00000000e+00   3.44006681e-03   2.63867386e-07]
 [  0.00000000e+00   6.46965000e-03   0.00000000e+00   9.90027861e-01]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]]
```

(d)