深 圳 大 学 实 验 报 告

课程名称 人工智能导论

项目名称 实验三：强化学习实践

学　　院 计算机与软件学院

专　　业 软件工程（腾班）

指导教师 高 灿

报 告 人 黄亮铭 学号 2022155028

实验时间 2023 年 11 月 29 日至 2023 年 12 月 24 日

实验报告提交时间 2023 年 12 月 20 日

教务处制

一、实验目的与要求

实验目的：

1．熟悉强化学习相关概念；

2．了解表格解决算法；

3．了解探索与利用的平衡策略，运用强化学习解决问题；

实验要求：

1. 实验提交文件为实验报告和相关程序代码，以压缩包的形式提交，命名规则为“学号数字+姓名+Task3”，如 2020154099 张三 Task3；

2. 所有素材和参考材料需列明出处，实验报告中的图片和程序代码建议标注个人水印或标识信息：姓名，班级，学号信息；

3．实验报告内容原则上控制在 10 页之内。

二、实验内容与方法

实验内容（三选一）：

1. 使用贪心算法和 ϵ-贪心算法解决多臂老虎机问题；

2. 使用价值迭代算法完成网格世界问题；具体细节参考文件“RL 指引.doc”；

3. 自选其它强化学习案例并实现；

三、实验步骤与过程

实验内容选择：使用贪心算法和 ϵ-贪心算法解决多臂老虎机问题。

1） 设计思路及代码：

a) 参考来源：第 11-12 课强化学习 pdf、多臂老虎机实验指引和 github 的多臂老虎机代码；

b) 根据多臂老虎机实验指引的示例代码，编写多臂老虎机类；

```python
class BernoulliBandit(Bandit):

    def __init__(self, n, probas=None):
        assert probas is None or len(probas) == n
        self.n = n
        if probas is None:
            np.random.seed(int(time.time()))
            self.probas = [np.random.random() for _ in range(self.n)]
        else:
            self.probas = probas

        self.best_proba = max(self.probas)

    def generate_reward(self, i):
        # The player selected the i-th machine.
        if np.random.random() < self.probas[i]:
            return 1
        else:
            return 0
```

多臂老虎机

c) 参考 github 的多臂老虎机代码编写求解基类 BaseSolution。成员变量：bandit 代表老虎机，counts 代表每个老虎机臂的选择次数，actions 存储每次选择的老虎机臂的编号，regret 代表当前懊悔，regrets 存储每次懊悔；成员方法：update_regret 用于更新当前懊悔值和历史懊悔值，start 用于开始游戏，其中的参数 step 代表要玩的次数，estimated_probas 和 run 由子类实现。

```python
class BaseSolution:
    def __init__(self, bandit):
        assert isinstance(bandit, BernoulliBandit)
        np.random.seed(int(time.time()))
        self.bandit = bandit
        self.counts = [0] * self.bandit.n
        self.actions = []
        self.regret = 0.0
        self.regrets = [0.0, ]

    def update_regret(self, index):
        self.regret = self.bandit.best_proba - self.bandit.probas[index]
        self.regrets.append(self.regret)

    def start(self, step):
        for i in range(step):
            self.run()

    @property
    def estimated_probas(self):
        raise NotImplementedError

    def run(self):
        raise NotImplementedError
```

BaseSolution 类

d) **Greedy 类（贪心算法求解）**：用贪心的方法求解多臂老虎机问题。

成员变量：Estimates：存储每一个老虎机得到奖赏的概率，该概率由方法决定，并非真实概率；Sum_reward：存储当前轮结束后的累计奖赏 Sum_rewards：存储历史每一轮结束后的奖赏

成员方法：Estimated_probas：返回 estimates；Run：获取 estimates 中值最大的索引，模拟操作老虎机然后得到奖赏，最后更新相关信息；Update_reward:用于维护 sum_reward 和 sum_rewards

```python
class Greedy(BaseSolution):
    def __init__(self, bandit, init_proba=1.0):
        super(Greedy, self).__init__(bandit)
        self.estimates = [init_proba] * self.bandit.n
        self.sum_reward = 0.0
        self.sum_rewards = [0.0, ]

    @property
    def estimated_probas(self):
        return self.estimates

    def run(self):
        index = max(range(self.bandit.n), key=lambda x: self.estimates[x])
        reward = self.bandit.generate_reward(index)
        self.estimates[index] = (self.estimates[index] * self.counts[index] + reward) \
                                / (1.0 * self.counts[index] + 1)
        self.counts[index] += 1
        self.actions.append(index)
        self.update_regret(index)
        self.update_reward(reward)

    def update_reward(self, reward):
        self.sum_reward += reward
        self.sum_rewards.append(self.sum_reward)
```

Greedy 类

e) **EpsGreedy 类（ε-贪心算法求解）**：Greedy 类的优化版本，使用参数 epsilon 在一定概率下降低贪心出错的次数。与 Greedy 类不同之处在于成员方法 run，EpsGreedy 中的 run 方法添加了选择结构：有 epsilon 的概率随机选择一个老虎机臂，有 1-epsilon 的概率选择 estimates 中值最大的老虎机臂。

```python
class EpsGreedy(BaseSolution):
    def __init__(self, bandit, eps, init_proba=1.0):
        super(EpsGreedy, self).__init__(bandit)
        self.eps = eps
        self.estimates = [init_proba] * self.bandit.n
        self.sum_reward = 0.0
        self.sum_rewards = [0.0, ]

    @property
    def estimated_probas(self):
        return self.estimates

    def run(self):
        if np.random.random() < self.eps:
            index = np.random.randint(0, self.bandit.n)
        else:
            index = max(range(self.bandit.n), key=lambda x: self.estimates[x])

        reward = self.bandit.generate_reward(index)
        self.estimates[index] = (self.estimates[index] * self.counts[index] + reward) \
                                / (1.0 * self.counts[index] + 1)
        self.counts[index] += 1
        self.actions.append(index)
        self.update_regret(index)
        self.update_reward(reward)

    def update_reward(self, reward):
        self.sum_reward += reward
        self.sum_rewards.append(self.sum_reward)
```

EpsGreedy 类

f) UCB 类（上置信界求解）：引入不确定性度量，代替 ε-贪心算法中依靠概率降低贪心出错的次数。对霍夫丁不等式求解，我们得到不确定性度量为 $\delta = \sqrt{\frac{\ln N}{2n}}$，然后我们老虎机臂的选择策略变为：评分=探索得到的概率+不确定度，选取评分最高的老虎机臂。

```python
class UCB(BaseSolution):
    def __init__(self, bandit, init_proba=1.0):
        super(UCB, self).__init__(bandit)
        self.count = 0
        self.estimates = [init_proba] * self.bandit.n
        self.sum_reward = 0.0
        self.sum_rewards = [0.0, ]

    def estimated_probas(self):
        return self.estimates

    def run(self):
        ucb_list = [self.estimates[j] + math.sqrt(math.log2(self.count + 1) / 2.0 / (self.counts[j] + 1))
                    for j in range(self.bandit.n)]
        index = ucb_list.index(max(ucb_list))

        reward = self.bandit.generate_reward(index)
        self.estimates[index] = (self.estimates[index] * self.counts[index] + reward) \
                                / (1.0 * self.counts[index] + 1)
        self.counts[index] += 1
        self.count += 1
        self.actions.append(index)
        self.update_regret(index)
        self.update_reward(reward)

    def update_reward(self, reward):
        self.sum_reward += reward
        self.sum_rewards.append(self.sum_reward)
```

UCB 类

g) **Show_results_with_different_methods 方法**：用于生成不同方法的结果的图片。

I. 首先初始化绘图：

```
122         graph = plt.figure(figsize=(15, 15))
123         graph.suptitle('K={} N={}'.format(K, N))
124         ax1 = graph.add_subplot(211)
125         ax2 = graph.add_subplot(212)
```

II. 然后绘制步数-累计收益的折线图，设置折现点间隔 100；

```
127     # 累计收益
128     for index, method in enumerate(methods):
129         ax1.plot(range(len(method.sum_rewards)), method.sum_rewards,
130                  label=method_names[index], alpha=0.7)
131
132     ax1.set_xlabel('步数')
133     ax1.set_xticks(range(0, 1001, 100))
134     ax1.set_ylabel('累计收益')
135     ax1.legend(loc='best')
```

III. 绘制每一步懊悔值得散点图，设置每个点间隔 20，最后保存图片；

```
136     # 单次懊悔
137     for index, method in enumerate(methods):
138         ax2.scatter(range(0, len(method.regrets), 20), method.regrets[0:len(method.regrets):20],
139                     label=method_names[index], s=(300 - index * 100))
140
141     ax2.set_xlabel('步数')
142     ax2.set_xticks(range(0, 1001, 20))
143     ax2.set_ylabel('单次懊悔')
144     ax2.legend(loc='best')
145
146     graph.savefig(figname)
```

h) **Show_results_with_different_arguments 方法：**与上述方法类似，但横坐标参数变为‘参数’；最后该方法需要返回最佳参数 eps 和最佳奖励 reward。

```
149 def show_results_with_different_arguements(methods, figname):
150     graph = plt.figure(figsize=(10, 5))
151     graph.suptitle('K={} N={}'.format(K, N))
152     ax1 = graph.add_subplot(111)
153
154     # 不同参数的最后收益
155     ax1.plot([_.eps for _ in methods], [_.sum_reward for _ in methods])
156     ax1.set_xlabel('参数')
157     ax1.set_xticks([_ / 100.0 for _ in range(0, 101, 5)])
158     ax1.set_ylabel('总收益')
159
160     graph.savefig(figname)
161     best_eps, best_reward = 0.01, 0
162     for _ in methods:
163         if _.sum_reward > best_reward:
164             best_eps = _.eps
165             best_reward = _.sum_reward
166     return best_eps, best_reward
167
```

i) 主函数

I. 设置 K、N 的值，初始化老虎机；

```
    K, N = 5, 1000
    b = BernoulliBandit(K)
    print("Randomly generated Bernoulli bandit has reward probabilities:\n", b.probas)
    print("The best machine has index: {} and proba: {:.4f}".format(
        max(range(K), key=lambda i: b.probas[i]), max(b.probas)))
```

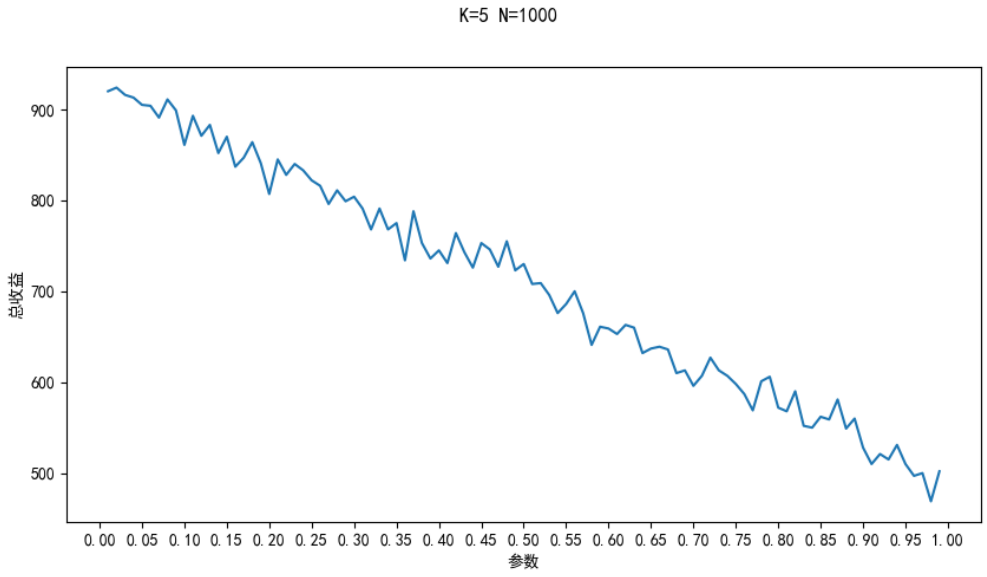
II. 设置不同参数（范围为 0.01-1，步进 0.01），然后使用 methods 存储不同参数的 EpsGreedy 对象，最后调用 show_results_with_different_arguments 方法获取最佳参数和奖赏。

```
methods = []
i = 0.01
while i <= 1:
    methods.append(EpsGreedy(b, i))
    i += 0.01
for method in methods:
    method.start(N)
best_arguement, best_reward = show_results_with_different_arguements(methods, "EGreedy不同参数_K{}_N{}.png".format(K, N))
print("The best arguement is {:.2f}\nThe best reward is {:.2f}".format(best_arguement, best_reward))
```

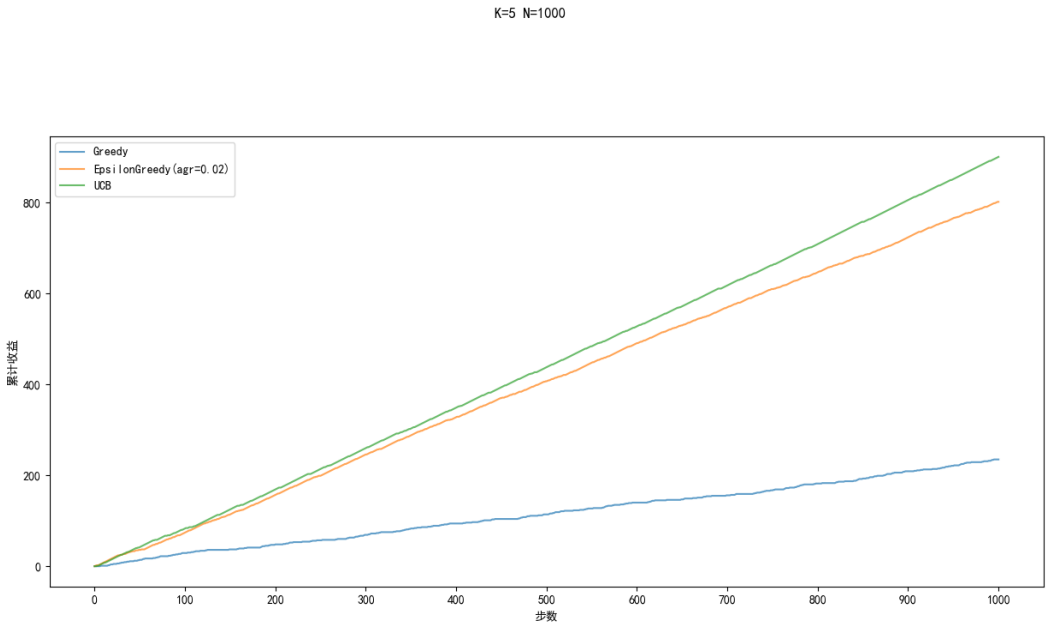
III. Methods 中存储 Greedy 对象和拥有最佳参数的 EpsGreedy 对象，然后调用 show_results_with_different_methods 方法。

```
    methods = [
        Greedy(b),
        EpsGreedy(b, best_arguement)
    ]
    names = [
        'Greedy',
        'EpsilonGreedy(agr={:.2f})'.format(best_arguement),
    ]
    for method in methods:
        method.start(N)
    show_results_with_different_methods(methods, names, "不同方法_K{}_N{}.png".format(K, N))
```

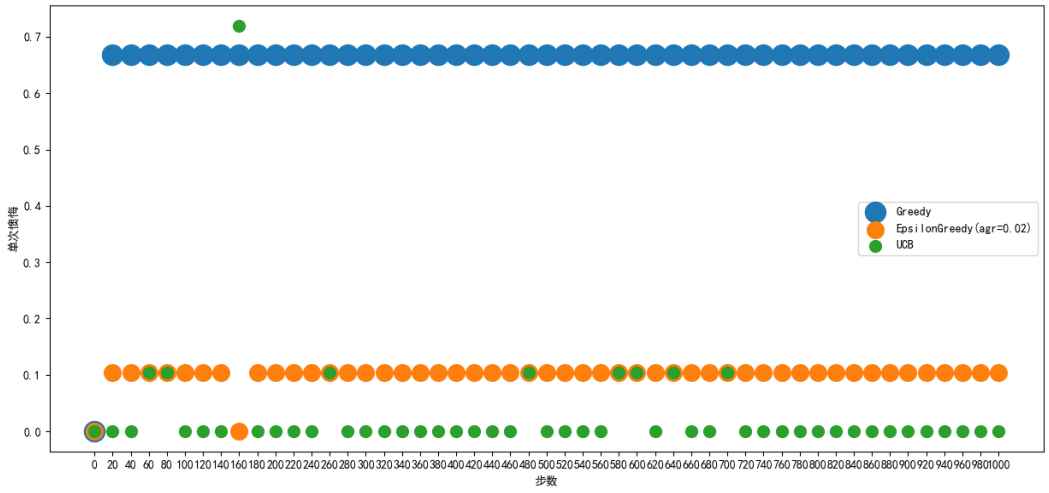
2） 代码运行结果（K=5，N=1000）



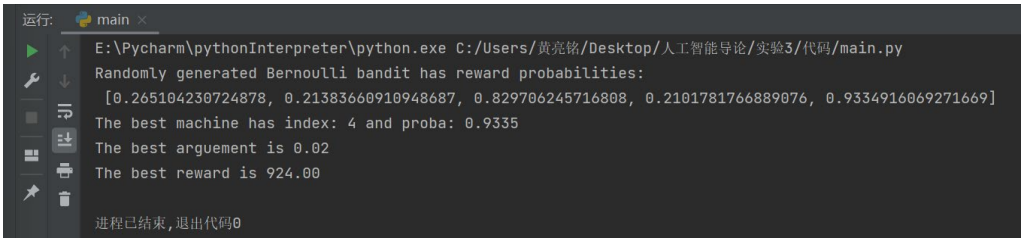
不同参数的 EpsGreedy 的总收益



不同方法的总收益（蓝为 Greedy，橙为 EpsGreedy）



不同方法每一轮的懊悔值（蓝为 Greedy，橙为 EpsGreedy）



主函数中 print 的结果

四、实验结论或体会

1. 经过多次测试，发现 EpsGreedy 的最佳参数位于 0.01-0.05 之间。在该范围外，随着参数的增加，总收益会波动下降；
2. 通过多轮测试，我们发现：三者的最佳收益相同，但是 UCB 的总收益平均最高，EpsGreedy 的总收益次之，Greedy 的总收益最低，远低于前面的两个；
3. 通过观察不同方法每一轮的懊悔值，发现 Greedy 的懊悔值最大，且远大于 EpsGreedy 和 UCB，EpsGreedy 的懊悔值次之，UCB 的懊悔值最小，EpsGreedy 和 UCB 的懊悔值相差不大。

指导教师批阅意见：   成绩评定： 指导教师签字：高灿 2023 年 12 月 22 日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。