第 9 章

强化学习：使用新获得的知识进行强化自身

在强化学习中，每个数据点都将执行一个操作。操作的执行将生成评估决策好坏的奖励信号。

然后，该算法修改其行动策略，以提高在特定环境下获得最高回报的可能性。

由于缺少训练数据，学习器从重复经验中学习。它通过记录收集知识（比如"这个动作是好的，那个动作是坏的"）。它通过尝试和错误来学习，试图修改和执行更好的奖励任务，唯一的目标是从应用环境中获得最大的长期回报。

本章还介绍了深度学习，它增强了深度强化学习的概念，以解决当前的业务问题。

目录

马尔科夫决策过程

马尔科夫决策过程 （MDP） 是一个离散时间随机控制过程。它提供了一个数学框架，用于在结果部分是随机的、部分受决策者控制的情况下对决策进行建模。MDP 有助于研究通过动态规划和强化学习解决的优化问题。

转换工作的基本数据流和交互在图 9-1中。

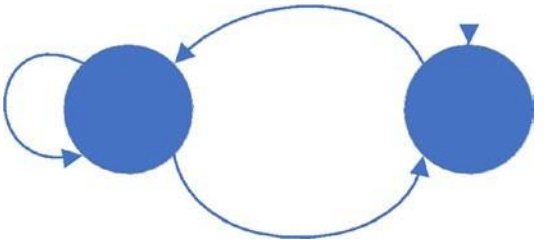

图9-1。 过渡图

我们将通过打开Jupyter笔记本将这个转化转换为代码：

Chapter-009-001-Markov-Decision-Process-01.ipynb

机器人行走

下面的过渡图说明了机器人 （1，1）如何通过避免（2，4）到达（3， 4）。

机器人有80%的可能性朝它面对的方向旅行，但也可以有10%的可能性转向，最终向左或右。参见图 9-2。

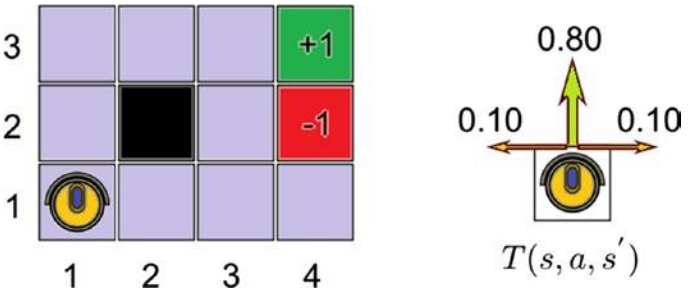

图9-2。 机器人行走过渡

要创建这些机器人过渡，您需要Jupyter 笔记本：

Chapter-009-002-Robot-Walk-01.ipynb

请遵循代码，因为它将过渡图反映到代码，以解决问题，让"Robbie"回家。

动态编程

动态生成（DP）既是一种数学优化方法，也是一种计算机编程方法。

DP

针对"复杂问题"问题进行部署，将问题分解为子集，然后解决已发现的子集问题，同时使用发现的信息来解决比较困难的原始的复杂问题。

您只需将问题分解为较小的可解决问题，并将结果合并以解决更大的问题。使用DP，您通常将结果存储在一个表中，以跟踪您的进度。

DP 是一种强大的技术，允许您在时间复杂度为 O （n2） 或 O（n 3）的情况下解决用朴素的方法需要指数时间的不同类型的问题。

迪杰斯特拉算法

最短路径问题的Dijkstra 算法可以通过解决一系列较小的路径圈，然后共同得到解决方案。

要分析算法的工作原理，请打开笔记本：

Chapter-009-003-Dijkstra-algorithm-01.ipynb

该代码演示了如何使用该算法解决在不同节点之间查找最短路径的问题。

```
print (Graph.dijkstra(graph,'a','e'))
Result:
deque(['a', 'b', 'd', 'e'])
```

下一个方法演示了怎样解决更复杂的问题。

打开 Jupyter Notebook : Chapter-009-004-Dijkstra-algorithm-02.ipynb

结果为:

```
Name:
Type: Graph
Number of nodes: 6
Number of edges: 9
Average degree:   3.0000

Dijkstra path:

['a', 'b', 'd', 'e']
```

我还建议您查看此代码生成的其他图。

提示 networkx 库对于处理节点（即代理）之间的关系非常有用。

活动选择问题

此问题要求您选择可以在给定时间段内执行的活动列表 。

注意 以下的实现假定活动已根据完成时间进行排序。

结果应每次显示一个可由一人完成的最大的活动集。

其中：

- n --> 活动总数
- S[]- - > 包含所有活动的开始时间的数组
- F[] -- > 包含所有活动的完成时间的数组

打开Jupyter 笔记本称为：

Chapter-009-005-Activity-Selection-Problem-01.ipynb

选择的活动如下：

```
Activity: A Start: 1h00 Finish: 2h00
Activity: B Start: 3h00 Finish: 4h00
Activity: D Start: 5h00 Finish: 7h00
Activity: E Start: 8h00 Finish: 9h00
```

汉诺塔

游戏"河内塔"使用三个棍。许多磁盘自底向上按递减顺序堆叠，即底部是最大的磁盘，顶部是最小的磁盘。磁盘建造一个圆锥形塔。游戏的目的是将磁盘塔整个从一根杆移动到另一根杆。参见图 9-3。

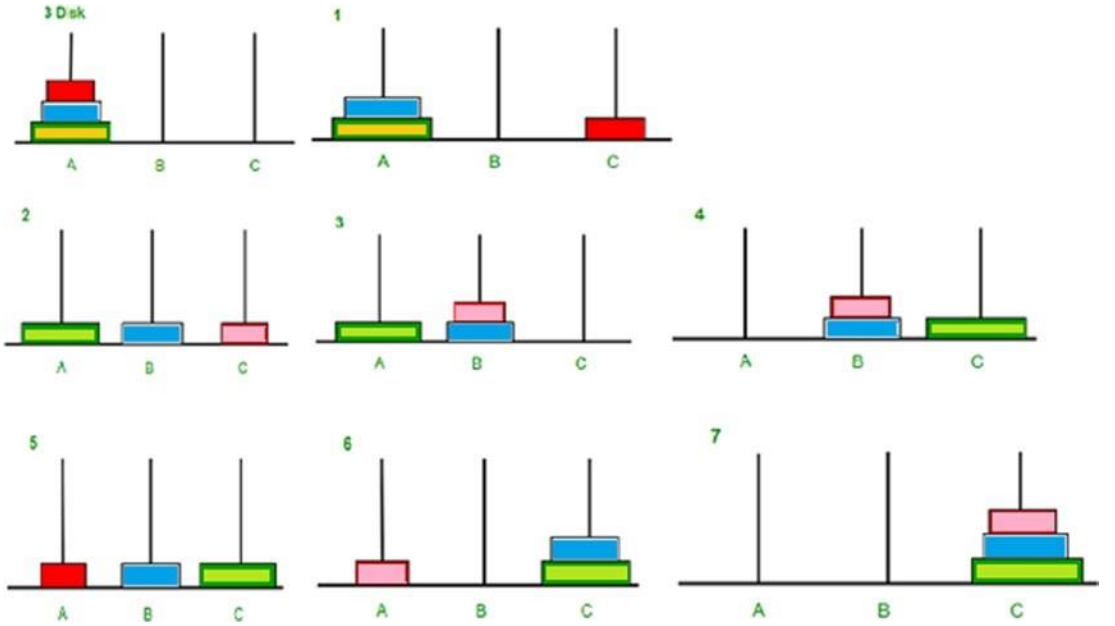

图9-3。 汉诺塔

必须遵守以下规则：

- 一次只能移动一个磁盘。
- 一根杆只有最上部盘才能 移动 。
- 如果此杆是空的，或者此杆的最上面的圆盘大于移动的圆盘，则它可以放在另一根杆上。

3 磁盘示例：

打开Jupyter 笔记本： Chapter-009-006-Towers-of-Hanoi-01.ipynb

结果：

```
Results:

Begin: ([3, 2, 1], 'source') ([], 'helper') ([], 'target')
moving 1 from source to target
moving 2 from source to helper
moving 1 from target to helper
moving 3 from source to target
moving 1 from helper to source

moving 2 from helper to target
moving 1 from source to target
End: ([], 'source') ([], 'helper') ([3, 2, 1], 'target')
```

3 到 11 个磁盘的示例：

现在查看增加的磁盘，因为这将演示该过程适用于任何数量的磁盘。

打开Jupyter 笔记本： Chapter-009-007-Towers-of-Hanoi-02.ipynb

您可以看到随着磁盘的增加，进程的运行时间也在增
加，但它仍然解决了问题，因为它只是遵循着规则。

它看起来可能只是一个游戏，但我有一个客户有一个自动仓储解决方案，它使用相同的解决方案域来计算高架堆垛机器人所使用的箱子的顺序，以达到正确的重量堆叠标准。

旅行推销员问题

旅游推销员问题 （TSP） 是说
以下问题："给定一个城市列表和城市之间的距离，访问每个城市并返回起始城市的最短路线是什么？在组合优化中，这是一个NP难题。

打开Jupyter Notebook： Chapter-009-008-Travelling-Salesman-01.ipynb

您的结果如图9-4所示。

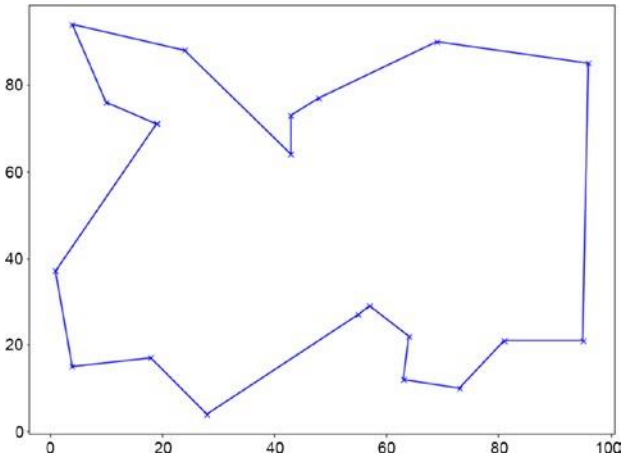

图9-4。 旅行推销员问题

旅行购买者问题与车辆路线问题同样是 TSP 的一种。它通常用于解决公共交通或公交线路的路线问题。

现在，我将向您展示对相同问题使用更好的优化版本。

打开Jupyter Notebook： Chapter-009-009-Travelling-Salesman-02.ipynb

您的结果如图9-5所示。

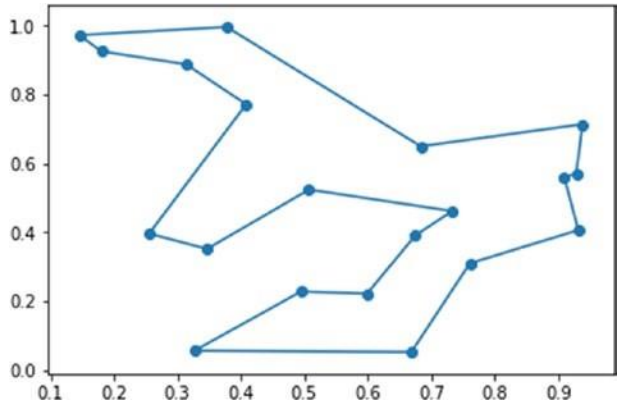

图9-5。 旅行路线问题

```
Route: [ 0 13 12 4 17 7 11 15 18 2 6 14 8 3 1 5 10 16 9 19]
Distance: 4.033827398325876
```

囚徒困境

囚徒困境是博弈论中分析类游戏的典型例证，它说明了为什么两个完全理智的人可能选择不合作，即使这样做似乎符合他们的最佳利益。

打开Jupyter Notebook称为： Chapter-009-010-Prisoner-Dilemma-01.ipynb

结果：

调查结果正在调查议员们是否会合作组成一个伙伴关系（C）或不合作（D）。当政党在选举中没有达到明确的多数原则时，这种测试很常见。

A 部分的结果如下：

```
When B cooperates and A cooperates, A receives 3 units of
political gains

When B cooperates and A defects, A receives 5 units of political gains

When B defects and A cooperates, A receives 0 units of political gains

When B defects and A defects, A receives 1 unit of political gains
```

一些可能的选择如下：

```
Strategies selected by A: [('D', 'D'), ('C', 'D')]

Strategies selected by B: [('D', 'D'), ('C', 'D')]

The pure strategies Nash equilibria are: ('D', 'D')
```

这表明，在获得一些收益后，最好不要形成伙伴关系。

多类队列网络 （MQN）

多类排队网络是一种网络，它模拟由实际处理情况创建的典型网络结构，其中不同处理点之间存在依赖关系，队列充当点之间的缓冲区。

打开 Jupyter 笔记本称为： Chapter-009-011-MQN-01.ipynb

提示 重新运行该过程，直到您获得收敛的组合。

该解决方案在 5，572 步收敛。

队列 A：

```
[[1.47025168 0.83294556]
 [0.68592644 0.36197673]
 [0.2433538  1.72692295]
 [2.0728354  1.49116639]
 [0.64876782 1.38615141]
 [0.30623    0.64057238]
 [1.35141951 0.78225212]
 [0.64335072 1.63394717]]
```

队列 B：

```
[[2.17254805 3.64517471]
 [3.16230687 2.97824253]
 [2.79010516 4.11028026]
 [2.38418286 4.05329159]
 [2.69742024 3.57219974]
 [3.03045585 2.64732508]
 [1.74762308 3.58107411]
 [3.09587674 2.65190328]]
```

队列 C：

```
[[ 2.83518099e+00  1.67734127e-03]
 [ 3.08180062e+00  1.68267411e-01]
 [ 2.71948131e+00 -3.15558079e-01]
 [ 2.79336116e+00 -4.95018881e-01]
 [ 2.52133725e+00  8.35166780e-01]
 [ 2.95958847e+00  3.69114538e-01]
 [ 2.79235514e+00  5.09964946e-01]
 [ 4.46361008e+00  4.24490087e-01]]
```

推荐系统

推荐者

系统或推荐系统是一种信息过滤系统，用于搜索任何用户对项目给予的"评级"或"偏好"的有效预测。它试图通过观察特定用户以前的行为或找到使用相同服务的其他用户的共同匹配模式来预测未来。这是电子商务系统上的一个常见过程。例如，它会建议您购买新项目或推荐

别人购买的类似物品。在工业行业中，这称为"追加销售"或"交叉销售"。

电影推荐系统

打开Jupter Notebook：Chapter-009-012-Recommender-01.ipynb

您的结果如图9-6所示。

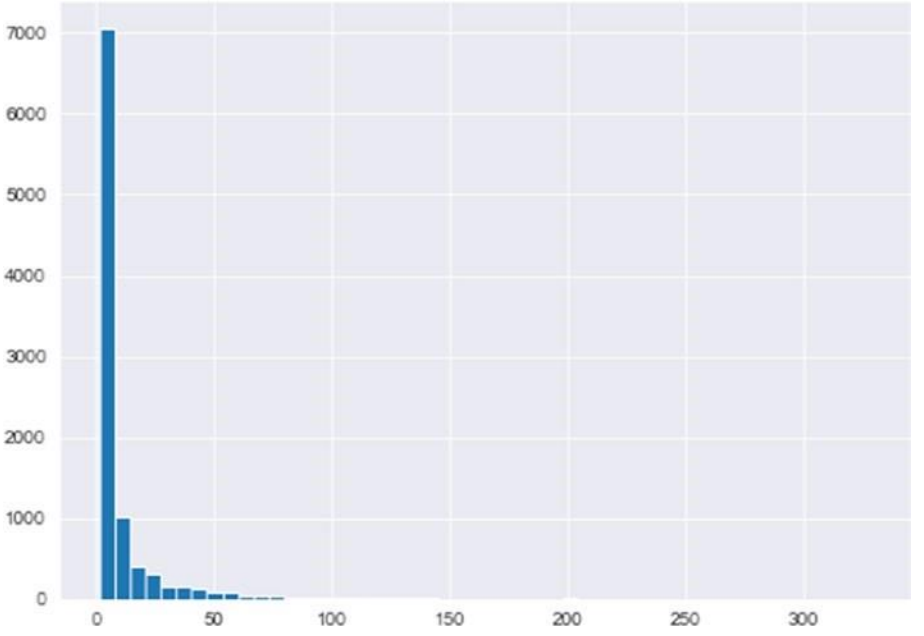

图9-6。 推荐系统-额定计数直方图

您的结果如图 9-7 所示。

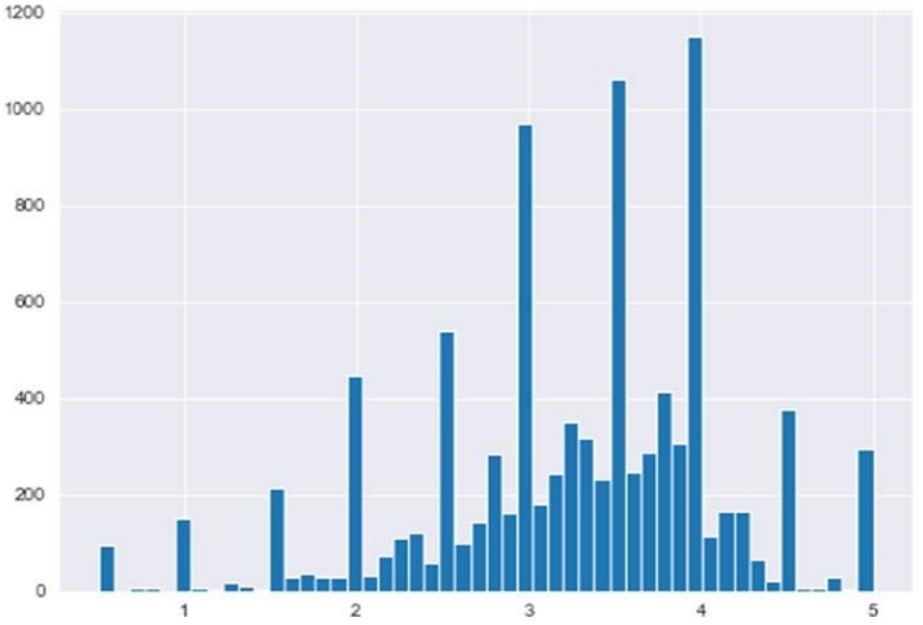

图9-7。 推荐系统 - 额定直方图

您的结果如图 9-8 所示。

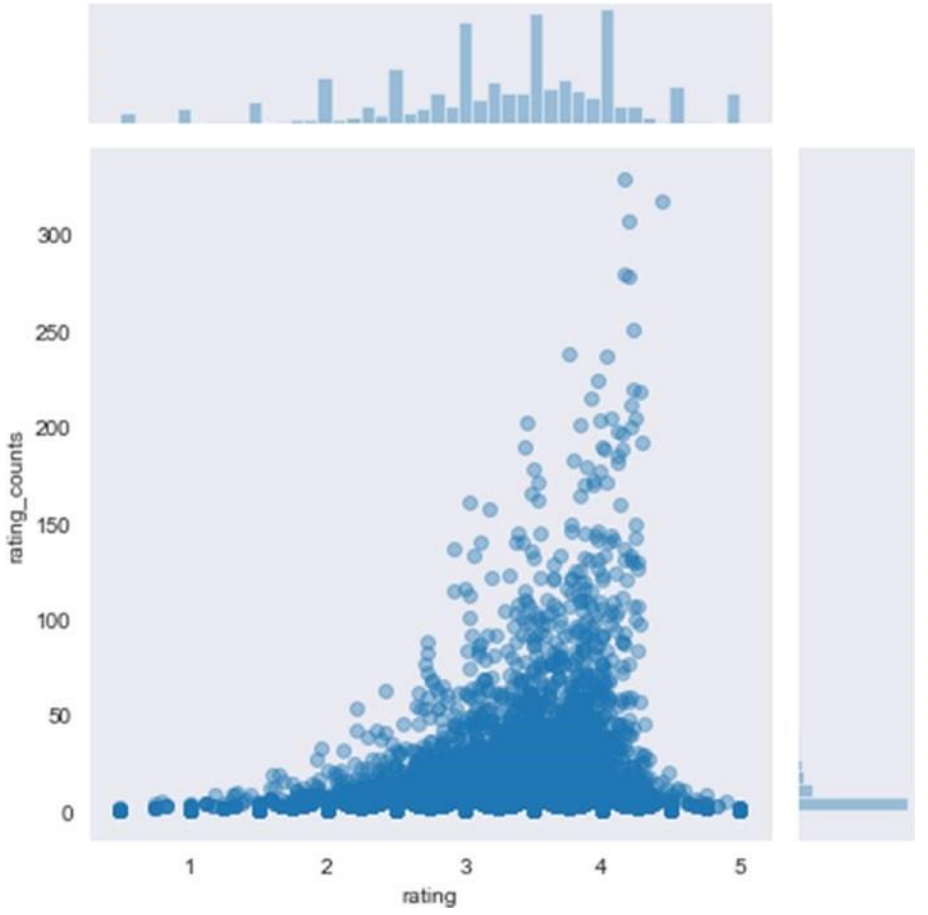

图9-8。 推荐系统-拼接图

title	Correlation
'burbs, The (1989)	0.197712
(500) Days of Summer (2009)	0.234095
*batteries not included (1987)	0.892710
...And Justice for All (1979)	0.928571
10 Cent Pistol (2015)	-1.000000

```
                                          Correlation
title
Lost & Found (1999)                               1.0
Century of the Self, The (2002)                   1.0
The 5th Wave (2016)                               1.0
Play Time (a.k.a. Playtime) (1967)                1.0
Memories (Memorîzu) (1995)                        1.0
Playing God (1997)                                1.0
Killers (2010)                                    1.0
Girl Walks Home Alone at Night, A (2014)          1.0
Tampopo (1985)                                    1.0
                              Correlation  rating_counts
title
'burbs, The (1989)               0.197712            17
(500) Days of Summer (2009)      0.234095            42
*batteries not included (1987)   0.892710             7
...And Justice for All (1979)    0.928571             3
10 Cent Pistol (2015)           -1.000000             2
```

```
print(corr_forrest_gump[corr_forrest_gump ['rating_counts']>50].sort_
values('Correlation', ascending=False).head())
```

Title	Correlation	rating counts
Forrest Gump (1994)	1	329
Mr. Holland's Opus (1995)	0.652144	80
Pocahontas (1995)	0.550118	68
Grumpier Old Men (1995)	0.534682	52
Caddyshack (1980)	0.520328	52

下一个推荐人系统使用一些共享文章。

打开Jupyter Notebook：Chapter-009-013-Recommender-02.ipynb

```
Loaded 3122 articles with 13 columns

Loaded 72312 articles with 8 columns
```

您的结果如图 9-9 所示。

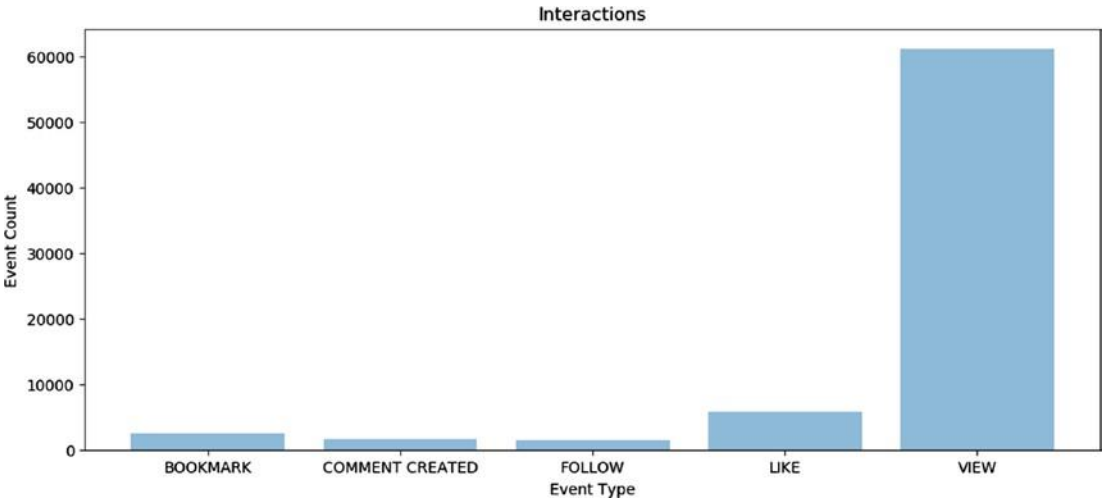

图9-9。事件传播-交互

Number of interactions: 72312

Number of users: 1895

Number users with at least 5 interactions: 1140

Number of interactions from users with at least 5 interactions: 69868

Number of unique user/item interactions: 39106

Number of interactions on Train set: 31284

Number of interactions on Test set: 7822

您的结果：

```
Evaluating Popularity recommendation model...
1139 users processed

Global metrics:
{'modelName': 'Popularity', 'recall@5': 0.2418818716440808, 'recall@10':
0.3725389925850166}

print(pop_detailed_results_df.head(10))
```

您的结果：

	_person_id	hits@10 _count	hits@5 _count	interacted _count	recall@10	recall@5
76	3609194402293569455	50	28	192	0.260417	0.145833
17	-2626634673110551643	25	12	134	0.186567	0.089552
16	-1032019229384696495	23	13	130	0.176923	0.100000
10	-1443636648652872475	9	5	117	0.076923	0.042735
82	-2979881261169775358	40	26	88	0.454545	0.295455
161	-3596626804281480007	18	12	80	0.225000	0.150000
65	1116121227607581999	34	20	73	0.465753	0.273973
81	692689608292948411	23	17	69	0.333333	0.246377
106	-9016528795238256703	18	14	69	0.260870	0.202899
52	3636910968448833585	28	21	68	0.411765	0.308824

基于内容的筛选模型

基于内容的筛选方法利用用户交互过的项目的描述或属性来推荐类似的项目。

它仅取决于用户以前的选择，使此方法具有健壮性，以避免冷启动问题。

“冷启动”发生在收集的数据集不适合引擎提供最佳的推荐结果时。

忽略来自英语和葡萄牙语的停止词（没有语义的单词）（因为你有一个混合语言的语料库）：

```
get_ipython().system('pip install -U nltk')
```

在您的系统上设置 NLTK 内容。

```
import nltk
nltk.download
```

现在，您可以启动程序，以演示可以使用 NLTK 的方法。

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
stopwords_list = stopwords.words('english') + stopwords.words('portuguese')
```

现在，你训练一个矢量大小为5000，由语料库中的核心单字和双元组组成，忽略了停止语的模型。

```
vectorizer = TfidfVectorizer(analyzer='word',
                             ngram_range=(1, 2),
                             min_df=0.003,
                             max_df=0.5,
                             max_features=5000,
                             stop_words=stopwords_list)
item_ids = articles_df['contentId'].tolist()
tfidf_matrix = vectorizer.fit_transform(articles_df['title'] +
"" + articles_df['text'])
tfidf_feature_names = vectorizer.get_feature_names()
print(tfidf_matrix.shape)
```

您的结果：

```
(3047, 5000)
```

现在，您可以循环浏览结果：

我只循环前 20 条记录。

```
for i in range(20):
```

```
print(' %2.0d => %0.4f - %0.4f' % (i, tfidf_matrix[i,0], tfidf_
matrix[i,1]))
```

您的结果：

```
 0 => 0.0000 - 0.0000
 1 => 0.0957 - 0.0000
 2 => 0.0000 - 0.0000
 3 => 0.0000 - 0.0000


 4 => 0.0000 - 0.0398
 5 => 0.0000 - 0.0000
 6 => 0.0000 - 0.0000
 7 => 0.0000 - 0.0134
 8 => 0.0000 - 0.0151
 9 => 0.0000 - 0.0000
10 => 0.0000 - 0.0000
11 => 0.0104 - 0.0000
12 => 0.0000 - 0.0000
13 => 0.0000 - 0.0145
14 => 0.0000 - 0.0000
15 => 0.0000 - 0.0000
16 => 0.0000 - 0.0276
17 => 0.0000 - 0.0000
18 => 0.0000 - 0.0000
19 => 0.0000 - 0.0300
```

干的好，现在你有了基本的强化学习模型。

现在，您已经认识了该技术，我建议你研究这些更复杂的模型。

提示 我建议你采取各种类型和组合的数据和分布来练习你的 ML 模型。这是有效和有效率地实现 IML 的唯一方法。

解决强化学习问题的框架

最短路径问题

给你举一个例子（图 9-10）来说明一下。

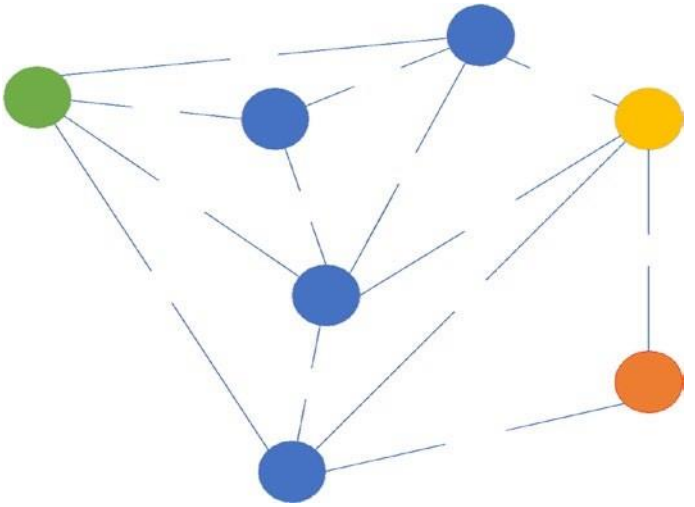

图9-10。 *最短路径问题*

这是最短路径问题的表示形式。任务是从 A 位转到 F 位，成本尽可能低。两个位置之间的边上的数字表示走这段距离所花费的成本。负成本是一些在路上有回报的部分。使用给定策略时，必须将"值"定义为累计总奖励。

这里

状态集是节点 [A、B、C、D、E、F、G]

要采取的行动是从一个地方转到另一个地方 [A -> B， C -> D 等]

奖励函数是由边表示的值，也就是说，成本

策略是完成任务[A -> ？ -> F]的的方法

你能解决这条路线吗？

打开Jupyter 笔记本称为： Chapter-009-014-Recommender-Network-03.ipynb

n = ['A','B','C','D','E','F']

```
import networkx as nx
G=nx.Graph()
G.add_nodes_from(n)
print ("Nodes of graph: ")
print(G.nodes())
```

结果：

```
Nodes of graph: ['A', 'B', 'C', 'D', 'E', 'F']
G.add_edge ('A','B', weight=25)
G.add_edge ('A','C', weight=150)
G.add_edge ('A','D', weight=25)
G.add_edge ('A','E', weight=22)
G.add_edge ('B','C', weight=100)
G.add_edge ('B','D', weight=15)
G.add_edge ('C','E', weight=333)
G.add_edge ('C','F', weight=280)
G.add_edge ('D','F', weight=150)
G.add_edge ('E','F', weight=60)
G.add_edge ('E','G', weight=180)
G.add_edge ('F','G', weight=390)
print ("Nodes of graph: ")
print(G.nodes())
print ("Edges of graph: ")
print(G.edges())
```

Nodes of graph:

['A', 'B', 'C', 'D', 'E', 'F', 'G']

Edges of graph:

[('A', 'B'), ('A', 'C'), ('A', 'D'), ('A', 'E'), ('B', 'C'), ('B', 'D'),

('C', 'E'), ('C', 'F'), ('D', 'F'), ('E', 'F'), ('E', 'G'), ('F', 'G')]

```
%matplotlib inline
import matplotlib.pyplot as plt
pos=nx.circular_layout(G, dim=2, scale=2)
nx.draw(G, pos=pos, node_size=800, node_shape='D')
labels=nx.draw_networkx_labels(G,pos=pos)
edge_labels=nx.draw_networkx_edge_labels(G,pos=pos)
plt.draw()
```

你的结果如图 9-11 所示

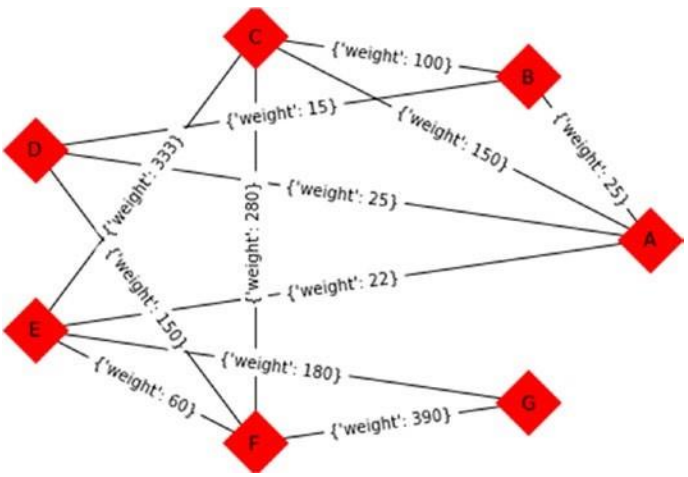

***图 9-11。** 最短路径解决方案*

```
print (nx.shortest_path(G, source='A',target='F', weight='weight'))
```

```
    Result: [‘A’, ‘E’ , ‘F’]
print (nx.shortest_path_length(G,source='A',target='F', weight='weight'))
    Result: 82
print (nx.shortest_path(G,source='A',target='G', weight='weight'))
    Result: [‘A’, ‘E’ , ‘G’]
print (nx.shortest_path_length(G,source='A',target='G', weight='weight'))
    Result: 202

for c in range(3):
    print ('Cut-off:', c)
    print ('=================')
    p = nx.single_source_shortest_path_length(G, source='A', cutoff=c)
    for key, value in p.items():
        print(key, ' => ', value)
    print('=================')
```

你的结果：

```
Cutoff: 0
=================
A => 0
=================
Cutoff: 1
=================
A => 0
B => 1
C => 1
D => 1
E => 1
=================
Cutoff: 2
=================
A => 0
B => 1
C => 1
D => 1
E => 1
F => 2
G => 2
=================
```

因此，您的政策是采取['A'，'E'，'F']，你的价值是82。

另一条路['A'，"E"，"G"]，您的值是 202。

祝贺！您刚刚实现了强化学习算法。

强化学习的实施

您将使用深度 Q 学习算法。Q-learning 是一种基于策略的学习算法，其函数估计为一个神经网络。

您将首先了解什么是Cart Pole问题，然后继续编写解决方案。

当我还是个孩子的时候，我记得我会挑一根棍子，在一只手上试着平衡它。我的朋友和我曾经有个比赛， 谁平衡更久会得到一个 "奖励"，一块巧克力！

让我们编写它！

要设置我们的代码，您需要先安装一些东西。

打开 Jupyter Notebook称为： Chapter-009-015-Q-Learn-01.ipynb
您需要安装以下内容：

pip install keras-rl

pip install h5py

pip install gym

让我们开始构建您的 Q-learning流程：

```
import numpy as np
import gym
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.optimizers import Adam
from rl.agents.dqn import DQNAgent
from rl.policy import EpsGreedyQPolicy
from rl.memory import SequentialMemory
ENV_NAME = 'CartPole-v0'
```

您必须获取环境并提取 CartPole 问题中的可用操作 数

```
 env = gym.make(ENV_NAME)
 np.random.seed(20)
 env.seed(20)
 nb_actions = env.action_space.n
```

您需要创建一个隐藏的层神经网络模型来支持环境的操作。

```
model = Sequential()
model.add(Flatten(input_shape=(1,) + env.observation_space.shape))
```

model.add(Dense(16))

model.add(Activation('relu'))

model.add(Dense(nb_actions))

model.add(Activation('linear'))

现在，您可以查看你的模型。

print(model.summary())

接下来，您配置和编译我们的智能体。

我建议您使用Epsilon贪婪策略，并将内存设置为顺序内存，因为您必须存储您的

policy = EpsGreedyQPolicy()
memory = SequentialMemory(limit=50000, window_length=1)
dqn = DQNAgent(model=model, nb_actions=nb_actions, memory=memory, nb_steps_
warmup=10, target_model_update=1e-2, policy=policy)
dqn.compile(Adam(lr=1e-3), metrics=['mae'])
是时候执行培训过程了。

警告

这可能需要一些时间来提高生态系统的计算能力。在云上运行此功能时，请关闭可视化功能。

---

try:
dqn.fit(env, nb_steps=5000, visualize=True, verbose=2)
except:
dqn.fit(env, nb_steps=5000, visualize=False, verbose=2)
您的结果：
Training for 5000 steps...

```
   10/5000: episode: 1, duration: 0.005s, episode steps: 10, steps per
second: 1825, episode reward: 10.000, mean reward: 1.000 [1.000, 1.000],
mean action: 0.000 [0.000, 0.000], mean observation: 0.135 [-1.986, 3.027],
loss: --, mean_absolute_error: --, mean_q: --
/usr/local/lib/python3.6/dist-packages/rl/memory.py:39: UserWarning: Not
enough entries to sample without replacement. Consider increasing your
warm-up phase to avoid oversampling!
  warnings.warn('Not enough entries to sample without replacement. Consider
increasing your warm-up phase to avoid oversampling!')
```

(Removed steps 2 to 4999 to save space in book. See notebook for details.)

```
4976/5000: episode: 234, duration: 0.143s, episode steps: 50, steps per
second: 349, episode reward: 50.000, mean reward: 1.000 [1.000, 1.000],
mean action: 0.460 [0.000, 1.000], mean observation: -0.167 [-0.756,
0.225], loss: 2.869326, mean_absolute_error: 10.015974, mean_q: 19.497936
done, took 14.771 seconds
```

或者使用可视化，请参见 图 9-12。

图9-12。 CartPole

现在，您可以根据学习测试强化学习模型。

尝试：

```
try:
    dqn.test(env, nb_episodes=5, visualize=True, verbose=2)
except:
    dqn.test(env, nb_episodes=5, visualize=False, verbose=2)
```

结果：

```
Episode 1: reward: 200.000, steps: 200
Episode 2: reward: 82.000, steps: 82
Episode 3: reward: 119.000, steps: 119
Episode 4: reward: 187.000, steps: 187
Episode 5: reward: 83.000, steps: 83
```

请注意

这五项测试得出了不同的结果。结果是由随机的起始点驱动的，但它们在结束时都是平衡的

如果你有一个像上面这样的 CartBot 试图自我平衡，你就已经成功地得到了一个 Q-learning 解决方案。

增加难度

以下领域是强化学习应用非常成功的地方：

博弈论

目前，高端人工智能（AI）系统是建立在一个单一智能体参与给定任务的概念上的，或者在对抗模型的情况下，由多个智能体相互竞争以改善系统的整体行为。

多智能体交互

多智能体

系统很常见，我们模拟由在同一生态环境内交互的团队组成的任何系统。有些行动会积极增强其他智能体的能力，或消极地阻碍其他智能体。

- 机器人群
- 群

- 车辆导航
- 医学研究
- 用于过程控制的工业物流
- 计算机网络
- 交通管制
- 智慧城市

如果你看一下自己的环境，你能看到有这种技术的应用程序吗？

这些环境的以下工业化版本非常有趣：

AlphaGo- https://deepmind.com/research/alphago/

你还记得在新闻上看到的这个吗——AlphaGo 是第一个击败人类职业棋手的计算机程序。

这个处理引擎现在也在尝试解决一些重要的实际问题。

这里是目前的强化学习的一小部分（表9-1）。).

Table 9-1. *Subset of Reinforcement Learning*

Algorithm	Description	Model	Policy	Action Space	State Space	Operator
Monte Carlo	Monte Carlo Simulations	Model-Free	Off-policy	Discrete	Discrete	Sample-means
Q-learning	State-action-reward-state	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA	State-action-reward-state-action	Model-Free	On-policy	Discrete	Discrete	Q-value
Q-learning - Lambda	State-action-reward-state with eligibility traces	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA - Lambda	State-action-reward-state-action with eligibility traces	Model-Free	On-policy	Discrete	Discrete	Q-value
DQN	Deep Q-Network	Model-Free	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Model-Free	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Actor-Critic Algorithm	Model-Free	Off-policy	Continuous	Continuous	Q-value
NAF	Q-Learning with Normalized Advantage Functions	Model-Free	Off-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage

我可以说出大量关于这些强化学习过程的信息书籍，因为目前世界各地的数据科学家和机器学习社区正在开发无数不同的选项

这是科学领域最活跃的研究和开发性自操作生态系统。

我预测这个生态系统将在未来200年对人类产生最大的影响。强化学习是第一个真正有能力大规模替换或取代人的进程。

提示 如果这本书中有一种技术你学得很好，那应该是强化学习。

在本章的其余部分，我将继续讨论一个更为普遍的趋势。

建模环境

我想解释一个通用术语"模拟"。

模拟是对一个系统内一组真实过程随时间的运行进行的模拟。

本质上，有一个系统，它有几个输入端，将近似的数学函数应用于选定的输入端，并以数据的形式返回一个输出端，该输出端可以可视化的方式观察系统当时在做什么。

回想CartPole 问题。这是一个简单的可视化，但你可以看到小车在试图平衡木杆。

您的问题将是模型或仿真的范围。由于现实世界中的大多数事件都是以某种方式相关的，所以在开始之前，您需要弄清楚您的项目范围有多广。

问题是：如何确定范围？

所以，你有一个带杆子的小车...这是范围吗？

让我们想一想...那一辆完整的车呢？参见图 9-13。

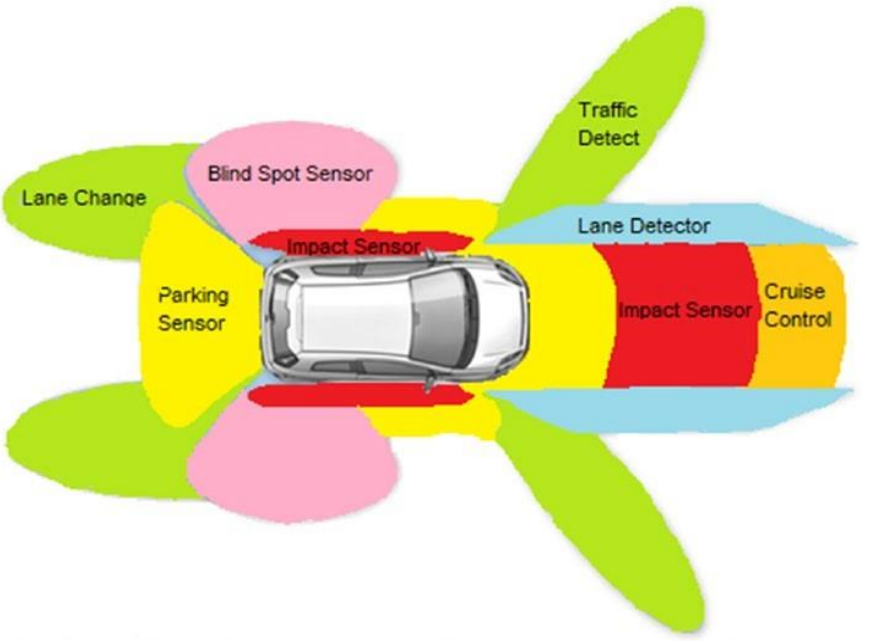

图9-13。 *汽车范围*

让我们再扩大一次范围...很多车辆？

每个穿过城镇的车辆可以参与绘制城市地图，然后共享信息（图 9-14）。

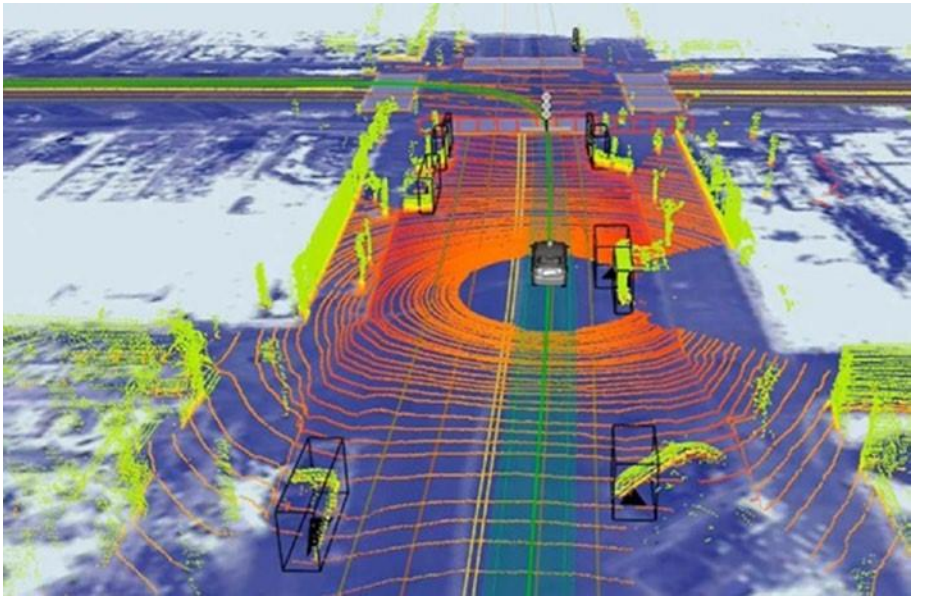

图9-14。 *小镇上的所有车*

更大...全世界所有汽车的数量？参见图 9-15。

图9-15。 *世界上所有车辆*

如果世界上的每辆车都关联起来呢？可能但又不可能的范围？

所以，我要说的是，你的范围越大，要处理的数据量就越大，你不能在一个给定的系统上解析模型的概率就越大。

状态功能创建

下一步是确定模型中的特征。

我建议你拍摄模型的图像，然后用每个有独特功能的标记标记它。

这里是标记的车辆型号（图9-16）。).

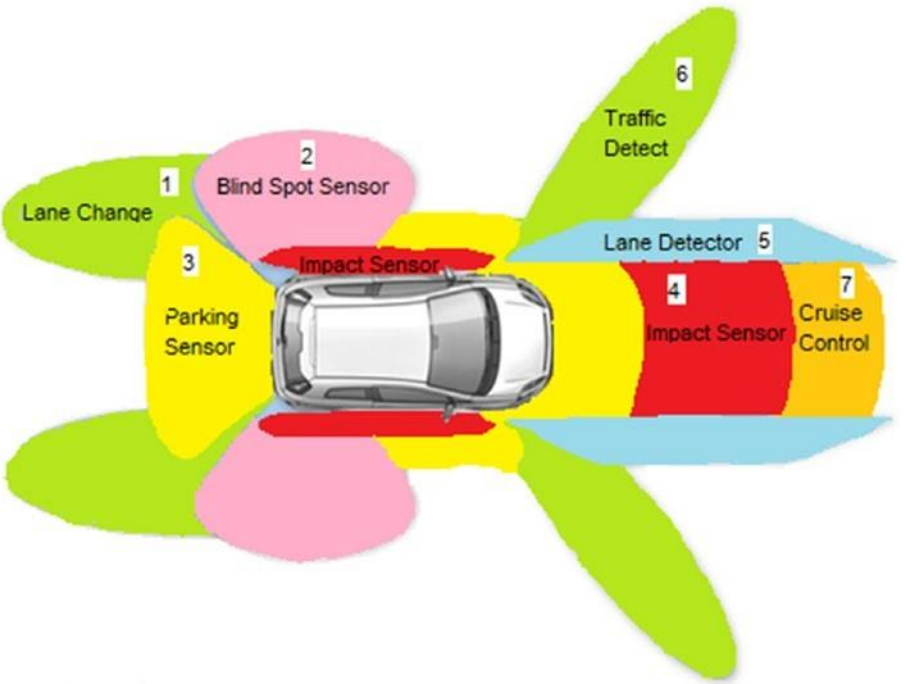

图9-16。 车辆型号

现在，我将引导您完成我识别的功能。

功能 1：

换车道功能是二进制传感器，因为传感器只回传值 0 或 1。

功能 2：

盲点传感器功能是一个二进制传感器，值也为 0 或 1。

特点 3：

停泊传感器功能是一个五级传感器也会有一个 0.0 到 1.0 之间的值。

它通过车内的一组 LED 灯显示您离后面物体的接近。

第二个输出是，当加权正确时，1.0 的距离 为 5 米 +/- 10 米。

功能 4：

冲击传感器功能是一个十级传感器，也是回传一个 0.0 到 1.0 之间的值。

第二个输出是，当加权正确时，1.0 的距离是5米+/-十分之一米的距离。

功能 5：

车道检测器功能是四级传感器，也是输出一个 0.0 到 1.0 之间的值。

第二个输出是，当加权正确时，1.0 的距离为 15 米 +/- 四分之一米。

功能 6：

交通探测器功能是一个五级传感器 ，也是输出一个 0.0 到 1.0 之间的值。

第二个输出是，当加权正确时，1.0 的距离为 15 米 +/- 四分之一米。

功能 7：

导航传感器功能是五级传感器，他传回一个-1.0 到 1.0 之间的值。

如果巡航控制检测到<0.0，传感器报告0.0，并且开始降低车辆速度以返回0.0；如果>0.0，则速度将增加，直到达到正确的速度

你现在有七个功能，因此可以从七个维度反馈有关模型的整体状态的信息。

奖励功能

奖励是七阶段奖励公式。这七个功能分别传回
奖励。您可以在这里使用七乘七相关算法来检查传感器之间是否存在显著的相关性。

警告 在现实世界中，这个过程已经用数百万小时的驾驶训练数据进行了多年，它们仍然得到离群值。

可以预测的是，一个解决未知输入引起的意外结果问题的算法突然出现
将立即对大多数RL系统的能力造成重大影响。

生成操作

这些动作包括从开关
警示灯到主动改变车速，甚至是踩刹车。车道检测甚至可以让汽车回到道路上。

警告 多个
可能的行为会导致同样的结果，或者导致环境的彻底失败带来灾难性的负面回报

我建议你的行动要循序渐进，不要有过大的改变。

最终型号

我希望您现在能够理解一些实际项目的模型可以迅速发展为成百上千的功能、状态和实际行动。

建议 我创建了至少三个具有不同基本特性的模型，然后检查推荐
的操作之间的相关性，以保证整个系统的合理操作。这样你就不会轻易地得到灾难性的失败状态。

逆强化学习

逆强化学习（IRL）是从观察到的行为中获得奖励函数的过程。虽然普通的“强化学习”涉及使用奖惩来学习行为，但在IRL中，方向是相反的。

此建模过程允许系统尝试任何新操作，然后计算二进制结果以重复建议的操作。

这是一个简单的模型，用于处理大量可能的操作列表。

例如，车辆为了防止离开路面，可以对汽车施加制动或转动方向盘。在IRL中，车辆将尝试每个单独的选项，并与其他选项组合，以了解单个操作或组合操作的结果。所有行动的总和，即行为就是奖励。

这减少了奖励衡量的数量。在大型实际项目中表现良好。

IRL也被用于进化计算中，以开发能够为环境自动建模的动作和奖励。这将在第10章中详细讨论。

深度强化学习

深度学习是机器学习领域的一种新的研究途径。深度学习背后的核心知识是创建由多层表现形式组成的体系结构，以便学习高级的
抽象。深度学习的基础是模型能够处理复杂的特征集，因为它将复杂的特征交互作为一系列较小的交互进行处理，并返回最终结果。

典型的使用IRL模型的深度学习人工神经网络，通过反向传播来确定哪些动作得到了最大的效果。反向传播是人工神经网络中用来计算梯度的一种方法，在计算网络中使用的权值时需要梯度。反向传播是“错误反向传播”的简写，因为错误是在输出时计算的，然后向后分布到网络的各个层。它通常用于训练深度神经网络。

这种通过神经网络深度学习实现的训练是将多种方法与深度强化学习相结合的先驱，目的是制造出第一批在几个具有挑战性的领域中达到人类水平的人工智能体。

你现在可以创造出能够通过选择好的行为而不是坏的行为来持续做出价值判断的代理。这种知识由一个Q网络表示，该网络估计了一个智能体
在采取某一特定行动后可能获得的总回报。

随着越来越多的现实问题被这些由成千上万个输入特征和数千个隐藏层组成的、输出端只有一个或几个系统行为的大规模深度学习模型来解决，这一领域正在加速发展。

这一领域在之后的很多年将取得许多伟大的成果。

多代理强化学习

多智能体强化学习（Multi-agent-reinforcement learning）是一种将一个系统的观察结果传递给另一个系统模型以增强其状态和奖赏收集的概念。例如，这可以部署在两个互相信息交流的车辆上
表示要采取行动的地方。从另一个角度看，他们的环境是从默认的角度出发的。例如，道路的一个压力激活部分，通过激活另一个智能体
可以检测到的信号来阻止车辆在道路上行驶。

例如，在一个智能城市中，“车辆的巡航控制与城市交通系统相互作用，以确保通过城市的交通流更加顺畅。或者重型货物车辆会从不能负载它
的桥上驶离。多智能体世界是物联网的核心世界。物联网革命将把数十亿个智能体
连接到一个传感器和状态记录器的单一集群中。

在未来五年内，多智能体环境将成为系统的常态。

你的机器学习需要能够处理这些工业化规模的物联网系统，再加上我在第6章和第7章中讨论的新机器人能力，这些将在未来十年内改变世界。

你取得了什么成就？

在本书的这一点上，您现在已经完成了核心机器学习方法论。

现在您应该能够执行监督学习、无监督学习和强化学习。

现在，您还应了解，最佳解决方案是来自所有三种核心方法的一系列组合技术。

提示

我建议您为给定的建模需求尝试尽可能多的选项，因为您需要的解决方案的复杂程度需要更多有创造性的解决方案。

接下来是什么？

后面的第14和15章将向您解释如何将您到目前为止在书中获得的知识转换为工业化级别的操作，以生成IML disruptor风格的解决方案。

接下来我们将在第10章中介绍进化计算的世界

提示 我建议你花几个小时来回顾前面的章节，以确保您更理解现在在你知识库中的信息。

现在为本书的下一阶段做好准备。

下一章将使您能够通过进化计算自动生成部分模型。