上海交通大学安泰经济与管理学院

BUSS3620 人工智能导论

代码实践 #2. 概率与优化

刘佳璐 助理教授

Q1. Markov Chain

在传统自然语言处理领域，人们会使用马尔科夫链生成文本。

现在你有一些商品的用户评价。你的任务是创建一个简单的马尔可夫链模型，基于给定的用户评价生成一些新的评价。这个模型应能够生成包含指定单词数量的新评价。

任务：

假设用户评价是马尔科夫链，其中每个单词都是马尔可夫链上的一个状态，你可以从单词之间的前后顺序得到每个单词与单词之间的转换概率。基于这个概率，给定一个单词，你可以预测下一个更有可能出现的单词，直到生成给定单词数量的新评价。

输入:

- 用户评价。
  `reviews = [ "This product is amazing", "Amazing quality and quick delivery", "Quick response from customer support", "Product quality is good but delivery was late"]`
- 生成评价的单词数量 `n=5`
- 起始单词 `starting_word="Product"`

输出:

- 一条新评价。

提示:

- 你可以先构建一个字典储存单词之间的转换信息。字典的主键是某一个单词，字典的值为该单词可能的下个单词的列表。例如：`{'is':['amazing','good']}`。注意，如果该单词为句子的末尾，那么字典的值是一个空列表。
- 接下来，你可以根据根据单词与单词之间的转换关系生成一段新的评价。注意：如果某个单词没有后续转换（即它是评价的结尾），生成过程需要提前停止。
- 你可以参考以下的代码框架：

```
def build_markov_chain(reviews):
    """创建一个字典来存储转换关系"""

def generate_review(markov_chain, start_word, n):
    """生成一条新的评价"""

if __name__=="__main__":
    reviews = [
        "This product is amazing",
        "Amazing quality and quick delivery",
        "Quick response from customer support",
        "Product quality is good but delivery was late" ]
```

```
    markov_chain = build_markov_chain(reviews)
    result = generate_review(markov_chain, "Product", 5)
    print(result)
```

Q2. Hill Climbing

一家公司在不同地点拥有多家门店，他们希望找到放置促销海报的最佳地点来优化销售策略。该区域被划分为一个 5x5 的网格，网格中的每个单元格代表一个具有特定销售潜力值（正整数）的地点。目标是将促销海报放置在某个位置上，使得相邻位置（上下左右及自己）的总销售潜力最大化。然而，为了简化过程，公司希望从随机位置开始，然后应用爬山算法找到更好的位置。

任务:
实现一个简单的爬山算法，找到一个能够最大化相邻位置（上下左右及自己）销售潜力总和的位置。

输入:

- 5x5 的销售潜力网格

```
grid = [
    [4, 2, 3, 7, 5],
    [1, 6, 8, 3, 4],
    [9, 7, 2, 5, 6],
    [3, 8, 4, 1, 2],
    [5, 6, 7, 3, 8]
]
```

输出:

- 放置促销海报的位置的行和列（从 0 开始计数）。

提示:

- 你可以参考课上的 `hospital.py`。
- 你可以参考以下的代码框架

```
class Space():

    def __init__(self, grid):
        """记录所需信息"""


    def get_neighbors_and_sales(self, row, col):
        """返回相邻位置以及当前坐标下的销售潜力总和"""


    def hill_climb(self, maximum=None):
        """实现爬山算法"""


grid = [
    [4, 2, 3, 7, 5],
    [1, 6, 8, 3, 4],
    [9, 7, 2, 5, 6],
    [3, 8, 4, 1, 2],
    [5, 6, 7, 3, 8]
]
```

```
location = Space(grid)
location.hill_climb()
print(location.location)
```

Q3. Constraint Satisfaction Problem

一家零售商商店有三个班次：早班（1）、午班（2）和晚班（3）。商店有四位员工：Alice、Bob、Claire 和 David。每位员工每天必须被分配一个班次。经理希望合理地为他们分配班次，有以下约束条件：

- Alice 只能在早班（1）或午班（2）工作
- Bob 拒绝在晚班（3）工作
- David 只能在晚班（3）上班
- Claire 和 Bob 不能被分配到同一个班次。
- 同一个班次不能分配超过两名员工

任务：
使用回溯算法，找到是否存在满足这些约束条件的排班表。

输入:

- 约束条件

输出:

- 排班表。请以字典的形式返回，字典的主键为员工，值为班次。

提示:

- 你可以参考课上的 `schedule0.py`。
- 你可以参考以下的代码框架

```
VARIABLES = […]
DOMAINS = {…}

def backtrack(assignment):
    """通过回溯算法找到排班表"""

def select_unassigned_variable(assignment):
    """找到一个未分配的变量"""

def consistent(assignment):
    """判断排班表是否满足全部约束"""

solution = backtrack(dict())
print(solution)
```