机器学习-第二讲课后作业

姓名：周宝航 学号：2120190442 专业：计算机科学与技术

一、问题描述

要求：使用 Python 语言编程实现 Find-S 算法和候选消除算法（参见《Machine Learning》一书相关内容），并使用这两种算法归纳给定训练样本的目标概念。最后，对归纳结果进行分析讨论。

难点：深入理解 Find-S 算法和候选消除算法，并使用 Python 语言编程实现它们。

二、基本思路

根据讲义内容和参考书籍，我们先理解 Find-S 算法和候选消除法的流程思路。针对不同的算法，我们对训练样本进行不同的处理方式。比如：Find-S 算法只考虑正例样本，而候选消除算法是分别对正例与负例样本进行处理。所以，在对算法有了深入理解后再去编程实现。最后根据两种算法的运行结果进行深入分析讨论。

三、解题步骤

1. 样本数据分析

在本次实验中，我们考虑进行归纳的样本一共有 14 个，其中关于概念 PlayTennis 为正例的样本有 9 个，而负例一共有 5 个。我们考虑到样本涉及的特征有：Outlook、Temperature、Humidity、Wind。如

下表 1，我们对每个特征涉及的取值进行了统计，以为后面的实验提供基础。

特征	Outlook	Temperature	Humidity	Wind
取值	Rain Sunny Overcast	Hot Cool Mild	High Normal	True False

表 1 数据特征统计

2. 算法描述

2.1 Find-S 算法

输入：训练样本、假设 h

1）将假设 h 初始化为假设空间 H 的最特殊假设

2）对每个训练样本正例 x

对假设 h 中的每个属性约束 $a_i = v$

如果 x 不满足 h 中的约束 $a_i = v$

则将 $a_i = v$ 替换为 x 满足的更一般约束

输出：假设 h

2.2 候选消除算法

输入：训练样本、边界集合 G、S

1）将 G 集合初始化为 H 中极大一般假设

2）将 S 集合初始化为 H 中极大特殊假设

3）对每个训练样本 d，进行下一步操作：

a. 如果 d 样本为正例：

a) 从 G 中移去所有与 d 不一致的假设

b) 对 S 中每个与 d 不一致的假设 s

从 S 中移去 s

把 s 的所有的极小一般化式 h 加入到 S 中，其中 h 满足：h 与 d 一致，而且 G 的某个成员比 h 更一般

从 S 中移去这样的假设：它比 S 中另一假设更一般

b. 如果 d 样本为负例：

a) 从 S 中移去所有与 d 不一致的假设

b) 对 G 中每个与 d 不一致的假设 g

从 G 中移除 g

把 g 的所有的极小特殊化式 h 加入到 G 中，其中 h 满足：h 与 d 一致，而且 S 的某个成员比 h 更特殊

从 G 中移去所有这样的假设：它比 G 中另一假设更特殊

输出：边界集合 G、S

3. 算法实现

3.1 Find-S 算法

```
def FindS(data):
    # 目标概念
    h = None
    for i, d in enumerate(data):
        # 若为正例样本
        if d[-1] == 'Yes':
            if h is None:
                h = d[:-1]
            else:
                for j, v in enumerate(d[:-1]):
                    # 若不符合约束，替换为更宽泛的约束
                    if h[j] != v:
                        h[j] = '?'
    return h
```

在上面我们实现的 Find-S 算法中，我们循环遍历样本。当遇到正例样本时，我们将其与假设 h 进行比较，对于不满足约束的属性，我们替换为更一般的约束，并用'?'符号代表一般约束。最后返回所求的假设 h。

3.2 候选消除算法

我们将该算法封装为一个类，而各个成员函数负责算法的不同步骤。我们将其分开一一列出：

```
def collect_attr(self, data):
    '''
        统计样本的所有属性
    '''
    attr = []
    data = np.array(data)
    n_attr = data.shape[1] - 1
    for i in range(n_attr):
        attr.append(np.unique(data[:,i]).tolist())
    return attr
```

上面的成员函数负责统计样本数据中每个特征所有的可能取值。

```
def consistent(self, h, d):
    '''
        判断样本 d 是否符合假设 h
    '''
    flag = True
    # 遍历样本和假设的属性
    for x, y in zip(h, d):
        # 若假设 h 为最特殊假设
        if x == 'o':
            flag = False
            break
        # 若样本 d 不满足假设 h
        if x != '?' and x != y:
            flag = False
            break
    return flag
```

该成员函数负责判断：一个样本是否符合一个假设。

```
def normal(self, g, h):
    '''''
        判断假设 g 是否比 h 更一般
    '''
    flag = True
    # 考虑假设 g 中的一般性
    n_g = len([1 for i in g if i == '?'])
    # 考虑假设 g 中的最特殊性
    o_g = len([1 for i in g if i == 'o'])
    # 考虑假设 h 中的一般性
    n_h = len([1 for i in h if i == '?'])
    # 考虑假设 h 中的最特殊性
    o_h = len([1 for i in h if i == 'o'])
    # 若 g 更特殊
    if o_g > 0 and o_h == 0:
        flag = False
    # 若 h 更特殊
    elif o_g == 0 and o_h > 0:
        flag = True
    # 若 h 与 g 都为特殊
    elif o_g > 0 and o_g == o_h:
        flag = True
    # 若 g 更一般
    elif n_g > n_h:
        flag = True
    # 若 h 更一般
    elif n_g < n_h:
        flag = False
    elif n_g == n_h:
        flag = True
        if g == h:
            flag = False
    return flag
```

该成员函数主要负责判断：一个假设是否比另一个假设更一般。

```
def findMinSpecial(self, d):
    '''''
        寻找极小特殊化式加入 G
    '''
    tmp = []
    for g in self.G:
        # 若 d 符合假设 g
        if not self.consistent(g, d):
            tmp.append(g)
```

```
        else:
        # 若 d 不符合假设 g
            i = 0
            for x, y in zip(d, g):
                if y == '?' or x == y:
                    # 根据属性构造极小特殊化式 h
                    for a in self.attr[i]:
                        if a != x:
                            h = g[:]
                            h[i] = a
                            # 若 h 与 d 一致
                            if not self.consistent(h, d):
                                flag = False
                                # S 的某个成员比 h 更特殊
                                for s in self.S:
                                    flag = self.normal(h, s)
                                    if flag:
                                        break
                                if flag:
                                    tmp.append(h)
                i += 1
    G = []
    # 遍历添加极小化特殊式到 G
    for i, g1 in enumerate(tmp):
        flag = True
        for j, g2 in enumerate(tmp):
            if i != j:
                # 判断 g 是否比 G 中另一假设更特殊
                flag = self.normal(g1, g2)
                if not flag:
                    break
        # 若 g 满足条件则不被移去
        if flag:
            G.append(g1)
    self.G = G
```

该函数主要负责寻找最小特殊化式并加入集合 G。

```
def findMinNormal(self, d):
    '''''
        寻找极小一般化式加入 S
    '''
    tmp = []
    for s in self.S:
        # 若 d 符合假设 s
```

```
        if self.consistent(s, d):
            tmp.append(s)
        else:
        # 若 d 不符合假设 s
            if s[0] == 'o':
                # 若 S 为最特殊假设
                tmp.append(d)
            else:
                # 根据样本 d 构造极小一般化式 h
                h = d[:]
                for i, v in enumerate(s):
                    # 不符合的属性替换为一般约束'?'
                    if v != h[i]:
                        h[i] = '?'
                # 若 h 与 d 一致
                if self.consistent(h, d):
                    flag = False
                    for g in self.G:
                        # 若 G 的某个成员比 h 更一般
                        flag = self.normal(g, h)
                        if flag:
                            break
                    if flag:
                        tmp.append(h)
    S = []
    # 遍历添加极小一般化式到 S
    for i, s1 in enumerate(tmp):
        flag = False
        for j, s2 in enumerate(tmp):
            if i != j:
                # 判断 s 是否比 S 中另一假设更一般
                flag = self.normal(s1, s2)
                if flag:
                    break
        # 若 s 满足条件则不被移去
        if not flag:
            S.append(s2)
    self.S = S
```

该函数主要负责寻找最小一般化式并加入集合 S。

```
class CandidateElimination(object):
    def __init__(self, data):
        self.data = data
        self.attr = self.collect_attr(data)
```

```
        self.S = [['o'] * len(self.attr)]
        self.G = [['?'] * len(self.attr)]
    def __call__(self):
        '''''
            算法主入口
        '''
        for d in self.data:
            # 特征
            x = d[:-1]
            # 标签
            y = d[-1]
            if y == 'Yes':
                # 从 G 中移去与 d 不一致的假设
                tmp = []
                for g in self.G:
                    if self.consistent(g, x):
                        tmp.append(g)
                self.G = tmp
                # 对 S 进行一般化
                self.findMinNormal(x)
            else:
                # 从 S 中移去与 d 不一致的假设
                tmp = []
                for s in self.S:
                    if not self.consistent(s, d):
                        tmp.append(s)
                self.S = tmp
                # 对 G 进行特殊化
                self.findMinSpecial(x)
            print(f"Data: {d}")
            print(f"S:{self.S}")
            print(f"G:{self.G}")
            print("..............................................")
```

上面是算法类的定义以及算法流程主函数。

四、结果与分析

1. 实验内容与步骤

我们采用 Find-S 算法和候选消除算法对给定训练样本进行训练，

得到对目标概念的学习结果。

2. 实验结果

2.1 Find-S 算法结果

```
Data: ['Overcast', 'Hot', 'High', 'False', 'Yes']
h: ['Overcast', 'Hot', 'High', 'False']
.................................................
Data: ['Rain', 'Mild', 'High', 'False', 'Yes']
h: ['?', '?', 'High', 'False']
.................................................
Data: ['Rain', 'Cool', 'Normal', 'False', 'Yes']
h: ['?', '?', '?', 'False']
.................................................
Data: ['Overcast', 'Cool', 'Normal', 'True', 'Yes']
h: ['?', '?', '?', '?']
.................................................
Data: ['Sunny', 'Cool', 'Normal', 'False', 'Yes']
h: ['?', '?', '?', '?']
.................................................
Data: ['Rain', 'Mild', 'Normal', 'False', 'Yes']
h: ['?', '?', '?', '?']
.................................................
Data: ['Sunny', 'Mild', 'Normal', 'True', 'Yes']
h: ['?', '?', '?', '?']
.................................................
Data: ['Overcast', 'Mild', 'High', 'True', 'Yes']
h: ['?', '?', '?', '?']
.................................................
Data: ['Overcast', 'Hot', 'Normal', 'False', 'Yes']
h: ['?', '?', '?', '?']
.................................................
Find-S算法结果
['?', '?', '?', '?']
```

图 1 Find-S 算法求解过程图

上图展示了 Find-S 算法根据正例样本求解假设 h 的过程。我们可以看到，随着正例样本的数量增多，假设 h 的属性会逐渐变得更加一般。这证明 Find-S 算法是在对目标概念进行学习。但在本次实验中，我们的目标概念 h 最后变成最一般假设。这种情况说明，根据这些训练样本，我们对目标概念的学习是不成功的。

2.1 候选消除算法结果

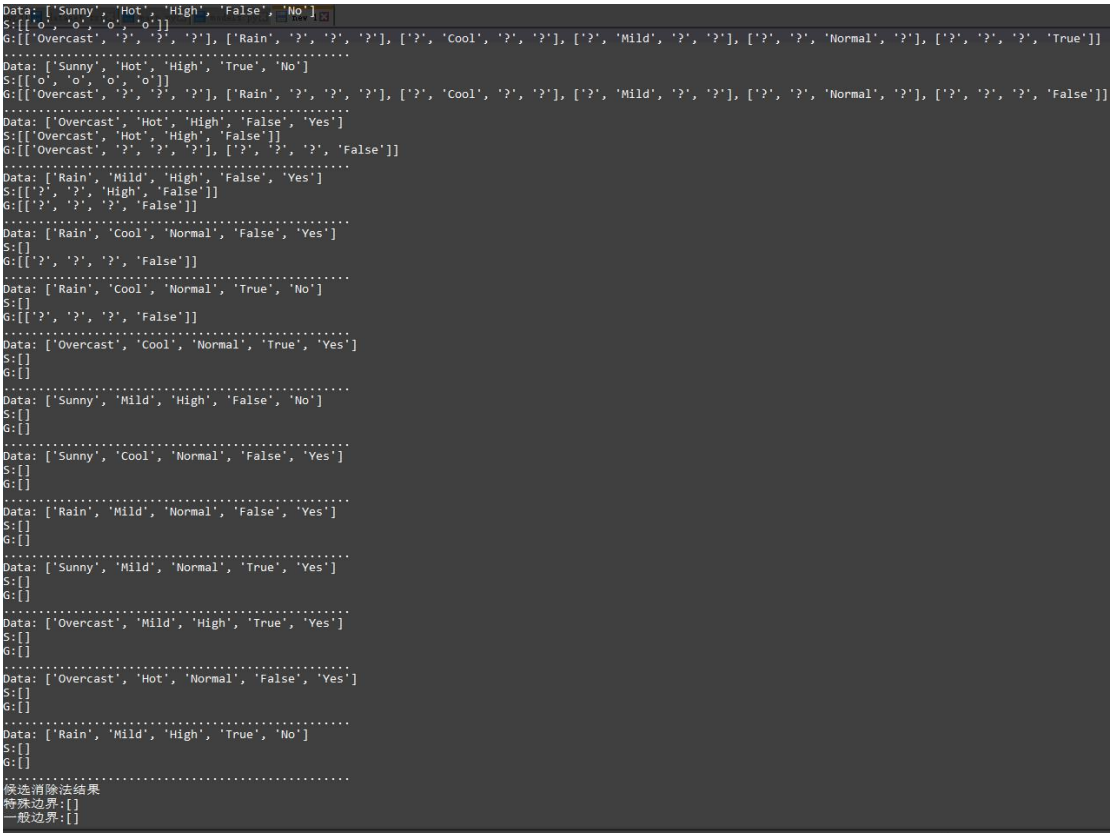

图 2 候选消除算法求解过程图

上图展示了候选消除算法根据训练样本求解边界集合的过程。我们可以看到，起始阶段的负例样本使得一般边界 G 集合变得更加特殊。而遇到正例样本，特殊集合 S 将变得更加一般，且不符合样本的 G 集合中的假设被移去。如此一来，G 集合与 S 集合将相互靠近。最后，两个集合变为空集，说明对目标概念的学习是不成功的。

3. 结果分析

Find-S 算法的结果为：[?,?,?,?]，而候选消除算法的结果为：G、S 集合均为空。Find-S 算法的结果表明：若要能够满足所有的正例样本，假设 h 的所有属性需要全部变为最一般化。可是这个假设并不能拒绝任一负例样本，所以该算法对目标概念的学习是不成功的。候选消除

算法的结果表明：若要满足所有的训练样本，特殊边界 S 最终无法一般化至某个假设集合，而一般边界 G 最终无法特殊化至某个假设集合。该算法本应将边界集合 G、S 分别变化至两个假设集合，且两边界之间的假设可以满足所有训练样本。可是，该算法作用在训练数据上，最终结果没有收敛，说明对目标概念的学习也是不成功的。