VE变量消除算法实现

（1）算法原理

1. 形式定义

- factor：包含某些变量的函数，比如：$P(C|A)=f(A,C)$，使用因子表示变量之间存在的条件依赖关系。
- 查询变量query：需要求的变量集合。
- 证据变量evidence_list：已知变量，给定变量的取值。
- 剩余变量ordered_list_of_hidden_variables ：需要使用VE算法一步步消除的变量。

2. 存储结构

- Node：使用类结点存储每一个因子。
  - name：因子结点的名字，用变量名字命名。
  - var_list：因子结点的变量集合，按照父子关系，存储当前变量和其父变量的名字。
  - new_cpt：变量集合各个变量取值组合的联合概率，使用字典存储。

3. VE算法

- 根据证据变量，对将相关的因子中的对应变量取指定的值，更新new_cpt取值集合。

对于$f(E,Y)$，此时假设E变量为证据变量，取值为e，则更新为$f_{E=e}(E,Y)$。

- 按照剩余变量的顺序依次消除变量。

  假设当前的因子集合F，对于每一个剩余变量Z：
  - 找出包括Z的所有因子$f_1,f_2,...f_n$。
  - 计算其所有因子的累乘：$h_i=\sum f_1*f_2*...*f_n$。
  - 对h_i，合并剩余变量Z的概率，消除Z的影响，得到h_i'
  - 将$f_1,f_2,...f_n$从F移除。
  - 将h_i'加入到因子集合F。
- 将剩余变量消除后，剩下的因子集合只涉及查询变量与证据变量，将所有因子进行累乘，得到最终一个因子。
- 对查询变量取不同的值分别得到一个概率，通过归一化从而得出对于不同查询变量取值下的条件概率。

（2） 伪代码

```
Algorithm VE
  input:factor_list,query,hidden_var,evidence_var #因子列表、查询变量、剩余变量、证据变量
  output:conditional rate list #条件概率集合

    #对于每个证据变量，让因子中的相关变量更新为特定值
```

```
for ev in evidence_var:
    #找出包含该变量的因子
    for factor in factor_list which include ev:
        #更新值域
        update factor value according specific value
#消除剩余变量
for var in hidden_var:
    #得出所有包含该变量的因子
    get all factors which include var from factor_list
    #将这些因子从列表中删除
    delete those factors from factor_list
    #将所有因子相乘得到一个新因子
    multiply those factors -> new_factor
    #在新因子中删除该变量并且合并相应的概率
    remove var from new_factor and merge corresponding rate
    #将新因子加入到因子列表
    add new_factor to factor_list
#将剩余的因子相乘得到最终的因子
multiply all factors in factor_list -> final_factor
#将所有概率求和
total <- sum(all rate in final_factor)
#归一化，求出不同查询变量取值下的条件概率
update every rate r in final_factor by r/total
#返回条件概率列表
return rate_list in final_factor
```

（3） 关键代码

1. 将所有证据变量取指定的值，更新所有因子结点的值域。

```
#给每个证据变量赋予指定的值
    for ev in evidence_list:
        #遍历所有变量结点
        for i in range(len(factor_list)):
            #如果该变量结点涉及证据变量，则更新其值域
            if ev in factor_list[i].var_list:
                factor_list[i] = factor_list[i].restrict(ev,evidence_list[ev])
```

2.对每个因子结点，给定变量和变量的值，遍历结点所有的值域，保留有指定变量值的值域。

```
#在指定变量的特定值条件下更新结点的值域
def restrict(self, variable, value):
    '''function that restricts a variable to some value
    in a given factor'''
    #定位指定变量在变量列表的下标
    index = self.var_list.index(variable)
    #新值域字典
    new_cpt = {}
    #遍历当前所有值域
    for key in self.cpt:
```

```
            #如果key在指定变量位置上的取值为特定值，则将
            #该值域和概率放入新字典
            if key[index] == str(value):
                new_cpt[key] = self.cpt[key]
        #复制变量列表
        new_var_list = self.var_list[:]
        #生成新结点
        new_node = Node('f' + str(new_var_list), new_var_list)
        #初始化值域字典
        new_node.set_cpt(new_cpt)
        return new_node
```

3.进行变量消元

- 遍历每个要消元的变量，首先进行初始化。

```
        #遍历消除每个剩余变量
        for var in ordered_list_of_hidden_variables:
            #生层新的变量结点集
            new_factor_list = [x for x in factor_list]
            #将原变量结点集清空
            factor_list = []
            first = 0
```

- 对于每一个消除变量，都遍历一次因子结点集，凡是涉及到该消除变量的因子结点，取出并且相乘最终得到一个新的因子结点。

```
            #遍历新的变量结点集
            for i in range(len(new_factor_list)):
                #如果此时结点中的变量已经被赋值，则将其加入到factor_list
                if len(new_factor_list[i].var_list) == 0:
                    factor_list.append(new_factor_list[i])
                    continue
                #如果消除变量存在因子结点中的变量集，并且第一次出现，则进行下一步
                if var in new_factor_list[i].var_list and first == 0:
                    multi_res = new_factor_list[i] #赋值给第一个乘数
                    first += 1
                #如果消除变量在因子结点中的变量集，并且不是第一次出现，进入下面步骤
                elif var in new_factor_list[i].var_list and first != 0:
                    #将当前因子结点与multi_res进行相乘
                    multi_res = multi_res.multiply(new_factor_list[i])
                #否则将其加入到factor_list
                else:
                    factor_list.append(new_factor_list[i])
```

- 对该新的因子结点，去除消除变量的影响，将对应值域的概率合并。

```
            #最后消除当前变量var，得到新的结点
            new_factor = multi_res.sum_out(var)
            #将生成的新因子结点放入factor_list，进入下一轮变量消除
            factor_list.append(new_factor)
```

4.因子结点相乘，分别在各自结点找到消除变量相同的值，将对应的概率相乘得到一个新的概率，将对应的变量取值合并作为新的值。

```
    #因子结点相乘
    def multiply(self, factor):
        '''function that multiplies with another factor'''
        #初始化变量
        new_cpt = {}
        same_index1 = 0
        same_index2 = 0
        #定位两个因子结点相同变量在各自变量列表的下标
        for i in range(len(self.var_list)):
            if self.var_list[i] in factor.var_list:
                same_index1 = i
                same_index2 = factor.var_list.index(self.var_list[i])
        #遍历因子结点1的cpt
        for key1 in self.cpt:
            #对于每一个key1，遍历结点2的cpt
            for key2 in factor.cpt:
                #如果key1和key2在相同变量的对应位置取值相同，则可以相乘
                if key1[same_index1] == key2[same_index2]:
                    #去掉相同变量
                    new_key1 = key1[:same_index1] + key1[same_index1+1:]
                    new_key2 = key2
                    #取两者变量的并集
                    new_key = new_key1 + new_key2
                    #将相乘的结果放入新生成的字典new_cpt
                    new_cpt[new_key] = self.cpt[key1] * factor.cpt[key2]
        #合并两者的变量列表
        new_list = self.var_list + factor.var_list
        #删除多余的相同变量
        new_list.pop(same_index1)
        #生成新结点
        new_node = Node('f' + str(new_list), new_list)
        #初始化值域字典
        new_node.set_cpt(new_cpt)
        return new_node
```

5.对于当前要消除的变量，在值域中，凡是除了该变量取值不同，其他变量相同的值，其概率相加得到一个新的概率值作为其他变量对应取值的概率，同时变量列表去除该消除变量。

```
    #消除指定变量的影响
```

```
    def sum_out(self, variable):
        '''function that sums out a variable given a factor'''
        #初始化
        new_var_list = []
        new_cpt = {}
        #定位指定变量在变量列表的下标
        index = self.var_list.index(variable)
        #遍历结点的所有值域
        for key in self.cpt:
            #将key中指定变量的位置去除
            new_key = key[:index] + key[index+1:]
            #如果新字典new_cpt已经有new_key键值，累加，否则创建新键值
            if new_key in new_cpt:
                new_cpt[new_key] += self.cpt[key]
            else:
                new_cpt[new_key] = self.cpt[key]
        #复制变量列表
        new_var_list = self.var_list[:]
        #去除指定变量
        new_var_list.remove(variable)
        #创建新结点
        new_node = Node('f' + str(new_var_list), new_var_list)
        #初始化值域字典
        new_node.set_cpt(new_cpt)
        return new_node
```

(5) 实验结果

1. task1

```
P(A) **********
RESULT:
Name = f['A']
 vars ['A']
   key: 1 val : 0.0025164420000000002
   key: 0 val : 0.997483558
```

2. task2

```
P(J && ~M) **********
RESULT:
Name = f['J', 'M']
 vars ['J', 'M']
   key: 11 val : 0.002084100239
   key: 10 val : 0.050054875461
   key: 01 val : 0.009652244741
   key: 00 val : 0.938208779559
```

3. task3

```
P(A | J, ~M) **********************
RESULT:
Name = f['J', 'M', 'A']
 vars ['J', 'M', 'A']
   key: 101 val : 0.013573889331307633
   key: 100 val : 0.9864261106686925
```

4. task4

```
P(B | A) **********************
RESULT:
Name = f['B', 'A']
 vars ['B', 'A']
   key: 01 val : 0.626448771718164
   key: 11 val : 0.373551228281836
```

5. task5

```
P(B | J, ~M) **********************
RESULT:
Name = f['J', 'M', 'B']
 vars ['J', 'M', 'B']
   key: 100 val : 0.9948701418665987
   key: 101 val : 0.0051298581334013015
```

6. task6

```
P(J&&~M | ~B) **********************
RESULT:
Name = f['J', 'M', 'B']
 vars ['J', 'M', 'B']
   key: 110 val : 0.001493351
   key: 100 val : 0.049847948999999996
   key: 010 val : 0.009595469
   key: 000 val : 0.939063231
```

总结

1. 变量消元中首先需要想好存储结构，以父子关系构建树，给每个因子创建一个结点，将其因子变量以及父母的变量存进变量列表，存好各种取值对应的概率。
2. 变量消除需要制定一个顺序，本次实验已经初步定好顺序，跟桶消除不太一样的是，一开始因子列表是全部存在的，需要通过特例化去掉某些因子中一些无关的取值，然后消除变量时，都需要遍历所有因子找出相关的结点相乘求和去除。
3. 因子相乘时，如果遇到有多个变量相同的情况时，需要分情况讨论，会比一个相同变量复杂点。