最高學習滿意度 解題報告

December 20, 2022

1 簡介與結果

依照題目要求，根據學生間的磨合指標，將彼此磨合指標好的分在同一班，磨合指標不佳的分在不同班，找出一個分班方式使得學習滿意度最大。採用了partitioning-based algorithm方式。先從一個初始的分班方法，計算當前分班方式的滿意度，每一回合找出對滿意度貢獻最少的學生轉班，若轉班後的滿意度比轉班前的滿意度，代表抵達local optimum。根據該分班作調整，試著跳脫該local optimum，以抵達global optimum。最後得到的最大滿意度爲742.6103，班別分配爲A B B A B B A A B B A A A B A A B B B B B B B A B B B A B A A A B A B A A A A A B B A A B B A B A B A B A A A A A A B B B B B B A A B B B B B A A A B A B B B B A B B A A A B A A B A B B B B B A A A B B B A。

2 演算法講解

2.1 符號定義

根據題目共有N位學生，矩陣W存放$w_{i,j}$爲同學i, j之間的磨合指標，且$w_{i,j} = w_{j,i}$，$w_{i,i} = 0$。

$$W_{n\times n} = \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n-1} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n-1,0} & w_{n,2} & \cdots & w_{n-1,n-1} \end{bmatrix}$$

向量S存放s_i爲第i位學生所在的班級，以1爲A班，以-1爲B班。

$$S_{n\times 1} = \begin{bmatrix} s_0 & s_1 & \cdots & s_{n-1} \end{bmatrix}^T$$

向量E存放e_i爲第i位學生與其他學生的學習滿意度總和，即該學生所貢獻的學習滿意度。

$$e_i = \sum_{j=0}^{N-1} e_{i,j} = \sum_{j=0}^{N-1} w_{i,j} s_i s_j$$

$$E_{n\times 1} = \begin{bmatrix} e_0 & e_1 & \cdots & e_{n-1} \end{bmatrix}^T = (W \times C) \circ C$$

整體學習滿意度爲E_{grand}，由於e_i中的$e_{i,j}$和e_j中的$e_{j,i}$屬於重複計算，所以e_i加總後要除以2。

$$E_{grand} = \sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} e_{i,j} = \frac{1}{2} \sum_{i=0}^{N-1} e_i$$

2.2 基礎演算法

給定一個猜測的分班方式S，根據當前的S算出E和E_{grand}，利用E找出貢獻度最低的學生，並更新S將該生由原班級轉到另一班，重複上述步驟直到更新後的E_{grand}比更新前的E_{grand}小，就結束迭代。

```
def GetLocalOptimum(self, S):
    ePrevious = -1e6
    E = np.matmul(self.W, S) * S
    eGrand = Solution.GetEValue(E)
    while eGrand > ePrevious:
        ePrevious = eGrand
        transferedStudent = Solution.GetTransferedStudent(E)
        S[transferedStudent] *= -1
        E = np.matmul(self.W, S) * S
        eGrand = Solution.GetEValue(E)

    eGrand = ePrevious
    S[transferedStudent] *= -1
    return eGrand, S
```

Listing 1: 找出Local Optimum

然而以上算法不不代表最後得到的答案爲最佳解。考慮到以下例子；

$$W = \begin{bmatrix} 0 & -1.0389 & 2.3049 & 0.482 & 1.3438 & -0.6013 & 0.0721 & -0.7811 & -0.1372 & -1.0739 \\ -1.0389 & 0 & -0.9695 & -0.8277 & 0.1481 & 0.5234 & 0.1609 & 0.4749 & 1.3733 & -1.6083 \\ 2.3049 & -0.9695 & 0 & 0.4272 & -1.3756 & -0.8237 & -1.4639 & -0.8184 & 1.415 & -1.4685 \\ 0.482 & -0.8277 & 0.4272 & 0 & 0.7398 & -0.5925 & 0.7684 & 0.4937 & -0.0691 & -0.1005 \\ 1.3438 & 0.1481 & -1.3756 & 0.7398 & 0 & 0.0583 & -0.7546 & -0.1031 & -1.0344 & 0.0475 \\ -0.6013 & 0.5234 & -0.8237 & -0.5925 & 0.0583 & 0 & -1.0608 & 1.0952 & -1.149 & 1.7517 \\ 0.0721 & 0.1609 & -1.4639 & 0.7684 & -0.7546 & -1.0608 & 0 & -0.3575 & 0.7297 & 0.5021 \\ -0.7811 & 0.4749 & -0.8184 & 0.4937 & -0.1031 & 1.0952 & -0.3575 & 0 & -1.2578 & 0.6817 \\ -0.1372 & 1.3733 & 1.415 & -0.0691 & -1.0344 & -1.149 & 0.7297 & -1.2578 & 0 & 0.8612 \\ -1.0739 & -1.6083 & -1.4685 & -0.1005 & 0.0475 & 1.7517 & 0.5021 & 0.6817 & 0.8612 & 0 \end{bmatrix}$$

當

$$S = \begin{bmatrix} 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

此時

$$E = \begin{bmatrix} 7.691 & 3.6122 & 5.4855 & 1.9767 & 2.3462 & 3.1197 & 1.3524 & 1.8454 & 0.3831 & 4.7838 \end{bmatrix}$$

$$E_{grand} = 16.298$$

若將貢獻度最少的第8位同學轉班，得到

$$S = \begin{bmatrix} 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 \end{bmatrix}$$

$$E = \begin{bmatrix} 7.4166 & 0.8656 & 8.3155 & 1.8385 & 0.2774 & 5.4177 & -0.107 & 4.361 & -0.3831 & 3.0614 \end{bmatrix}$$

$$E_{grand} = 15.5318$$

但存在分班方式使E_{grand}更大，如下：

$$S' = \begin{bmatrix} 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$$

$$E' = \begin{bmatrix} 4.8732 & 0.84 & 8.1389 & 1.8957 & 1.2318 & 7.6559 & 1.6162 & 4.8698 & 3.1451 & 2.1522 \end{bmatrix}$$

$$E'_{grand} = 18.2094$$

2.3 改進

觀察上述例子可以發現，停在local optimum的原因爲$w_{6,8}$較大，因此若將第8位學生轉到另一班，會損失過多的滿意度，但若一起將第8和6位同學轉到另一班，便能跳脫此local optimum，同時也注意到e_6與e_8較小（在此例中，正好爲E最小的兩個），因此我在Listing 1的外面再做一次迭代。
以當前最好E_{grand}的分班S做爲基礎調整，將e值最小的2位學生，一起轉到另一班，並以更新後的S'作爲下一次的起始猜測，重新跑一次GetLocalOptimum，跑完後的E'_{grand}若比E_{grand}小，一樣以S做爲基礎，並將調整的學生數加1。若比E_{grand}大，更新E_{grand}和S，並將調整的學生數重新設爲2。直到調整的學生數等於N，就結束迭代。

```
def Solve(self):
    eGrandMax = -1e6
    adjustedNumber = 2
    s0 = np.ones(self.N) # initial guess

    # start iteration
    while adjustedNumber < self.N:
        eGrand, s0 = self.GetLocalOptimum(s0)
        if eGrand > eGrandMax:
            S = s0
            E = np.matmul(self.W, S) * S
            eGrandMax = eGrand
            adjustedNumber = 2

        s0 = Solution.adjust(S.copy(), E, adjustedNumber)
        adjustedNumber += 1

    return eGrandMax, S
```

Listing 2: 調整local optimum，嘗試抵達global optimum

2.4 時間複雜度

在GetLocalOptimum中，所需要的迭代回合數最多爲$\lfloor \frac{N}{2} \rfloor$，且內部時間複雜度最高的矩陣乘法爲$O(n^2)$，因此GetLocalOptimum的時間複雜度爲$O(n^3)$。
在Solve中，所需要的迭代回合數與一開始的S和W的分佈有關，以此題給定的W，共花費了144回合。我認爲迭代回合數最多不會超過N^2，內部時間複雜度最高的GetLocalOptimum爲$O(n^3)$，所以Solve的時間複雜度爲$O(n^5)$。

3 完整程式碼

```
import numpy as np

def ParseWeightFile(path, N):
    W = np.zeros((N, N), dtype=np.float32)
    with open(path, mode="r") as input:
        lines = input.read().splitlines()
    for line in lines:
        strings = line.split()
        i, j, w = int(strings[0])-1, int(strings[1])-1, float(strings[2])
        if i < N and j < N:
            W[i,j] = w
            W[j,i] = w
    return W

def ConvertClassToString(cls):
    res = ""
    for c in cls:
        res += "A " if c == cls[0] else "B "
    return res

def SaveOutcome(e, cls, N):
    with open(f"{N}_sol_E.txt", mode="w") as output:
        output.write(str(e) + "\n")
    with open(f"{N}_sol_class.txt", mode="w") as output:
        output.write(ConvertClassToString(cls) + "\n")
```

```
class Solution:
    def __init__(self, N, W):
        self.N = N
        self.W = W

    @staticmethod
    def GetEValue(eVector):
        return eVector.sum() / 2

    @staticmethod
    def GetTransferedStudent(eVector):
        return np.argmin(eVector)

    @staticmethod
    def adjust(S, eVector, adjustedNumber):
        sorted_index = np.argsort(eVector)
        for i in range(adjustedNumber):
            S[sorted_index[i]] *= -1
        return S

    def GetLocalOptimum(self, S):
        ePrevious = -1e6
        E = np.matmul(self.W, S) * S
        eGrand = Solution.GetEValue(E)
        while eGrand > ePrevious:
            ePrevious = eGrand
            transferedStudent = Solution.GetTransferedStudent(E)
            S[transferedStudent] *= -1
            E = np.matmul(self.W, S) * S
            eGrand = Solution.GetEValue(E)

        eGrand = ePrevious
        S[transferedStudent] *= -1
        return eGrand, S

    def Solve(self):
        eGrandMax = -1e6
        adjustedNumber = 2
        s0 = np.ones(self.N) # initial guess

        # start iteration
        iteration = 0
        while adjustedNumber < self.N:
            eGrand, s0 = self.GetLocalOptimum(s0)
            if eGrand > eGrandMax:
                S = s0
                E = np.matmul(self.W, S) * S
                eGrandMax = eGrand
                adjustedNumber = 2

            s0 = Solution.adjust(S.copy(), E, adjustedNumber)
            adjustedNumber += 1
            iteration = 0

        print(iteration)
        return eGrandMax, S

def test():
    N = 4
    W = ParseWeightFile("w4.txt", N)
    sol = Solution(N, W)
    eGrand, S = sol.Solve()
```

```
    print(round(eGrand, ndigits=4), ConvertClassToString(S.tolist()))
    # SaveOutcome(round(eGrand, ndigits=4), S.tolist(), N)

def main():
    N = 101
    W = ParseWeightFile("w101.txt", N)
    sol = Solution(N, W)
    eGrand, S = sol.Solve()
    # print(round(eGrand, ndigits=4), ConvertClassToString(S.tolist()))
    SaveOutcome(round(eGrand, ndigits=4), S.tolist(), N)

if __name__ == "__main__":
    # test()
    main()
```

Listing 3: 完整程式碼