

## 华东师范大学软件学院课程作业

课程名称：软件质量分析	年级：2023 级本科	姓名：张梓卫
作业主题：游戏服务器系统的可信度量	学号：10235101526	作业日期：2024/12/04
指导老师：陈仪香	组号：	

## 目录

一 基于组件的软件可信性度量模型	1	3 可信等级	6
1 权重向量的计算	1	二 附录	7
2 计算 $T_s$ 的值	4	1 完整可执行代码	7

## 一 基于组件的软件可信性度量模型

## 1 权重向量的计算

首先需要计算 EV、LLSM、CSM，我们可以拿 Homework 6 的算法直接使用：

```
1 import numpy as np
2
3 A = np.array([
4     [1, 2, 1/2, 2, 1/4],
5     [1/2, 1, 2, 3, 1/2],
6     [2, 1/2, 1, 1, 1/2],
7     [1/2, 1/3, 1, 1, 1/2],
8     [4, 2, 2, 2, 1]
9 ])
10
11 def calculate_ev_weights(matrix):
12     eigenvalues, eigenvectors = np.linalg.eig(matrix) # 返回值为元组，第一个元素为特征值，第二个元素为特征向量
13     max_index = np.argmax(eigenvalues) # 取最大特征值对应的特征向量
14     weights = eigenvectors[:, max_index].real
15     # 归一化
16     return weights / np.sum(weights)
17
18 def calculate_llsm_weights(matrix):
19     # 分子 = 每行的乘积 × 开维度数次方
20     numerator = np.prod(matrix, axis=1) ** (1 / matrix.shape[0])
21     # 分母 = 每行的分子相加
22     denominator = np.sum(numerator)
23     # 归一化
24     return numerator / denominator
25
26 def calculate_csm_weights(matrix: np.array, epsilon: float, max_iterations: int) -> np.ndarray:
27     '''计算 CSM，传入: matrix, precision, maximum iterations'''
28     n = matrix.shape[0]
29     # 初始化初始解
30     W = np.ones(n) / n
31
32     for k in range(max_iterations):
```

```

33     e = np.zeros(n)
34     for i in range(n):
35         e[i] = np.sum([(1 + matrix[j, i] ** 2) * (W[i] / W[j]) - (1 + matrix[i, j] ** 2) * (W[j] / W[i])
36                        for j in range(n) if j != i])
37
38     max_e = np.max(np.abs(e))
39     if max_e <= epsilon: # 若精度已到达, 那么停止迭代
40         print(f"CSM 算法在迭代次数为 {k} 次时收敛")
41         break
42
43     m = np.argmax(np.abs(e)) # 查找最大无差的索引
44
45     # 计算 T(k)
46     up = np.sum([(1 + matrix[m, j] ** 2) * (W[j] / W[m]) for j in range(n) if j != m])
47     bottom = np.sum([(1 + matrix[j, m] ** 2) * (W[m] / W[j]) for j in range(n) if j != m])
48     T = np.sqrt(up / bottom)
49
50     # 更新矩阵向量, 归一化
51     X = W.copy()
52     X[m] *= T
53     W = X / np.sum(X)
54
55     return W
56
57 def calculate_td(matrix: np.array, weight_vector: np.ndarray) -> float:
58     n = len(weight_vector)
59     td = 0.0
60     for i in range(n):
61         for j in range(n):
62             td += abs(matrix[i, j] - (weight_vector[i] / weight_vector[j]))
63     return td
64
65 weights_ev = calculate_ev_weights(A)
66 print("判断矩阵:\n", A)
67 print("权重向量 EV:", weights_ev)
68
69 weights_llsm = calculate_llsm_weights(A)
70 print("权重向量 LLSM:", weights_llsm)
71
72 weights_csm = calculate_csm_weights(A, 1e-10, 1000)
73 print("CSM 权重向量:", weights_csm)
74
75 TD_EV = calculate_td(A, weights_ev)
76 TD_LLSM = calculate_td(A, weights_llsm)
77 TD_CSM = calculate_td(A, weights_csm)
78
79 print("TD EV:", TD_EV)
80 print("TD LLSM:", TD_LLSM)
81 print("TD CSM:", TD_CSM)
82
83 td_values = {"EV": TD_EV, "LLSM": TD_LLSM, "CSM": TD_CSM}
84 min_method = min(td_values, key=td_values.get)
85 print(f"最小的 TD 值为 {td_values[min_method]}, 选 {min_method} 计算的权重向量为可信属性的权重向量。")

```

## Homework 6

将关键组件的矩阵 A 输入到代码中, 输出结果为:

```

D:\Python\python.exe F:\Project\Python\SoftwareQuality\SoftwareQuality\Main9.py
判断矩阵:
[[1.      2.      0.5      2.      0.25     ]
 [0.5      1.      2.      3.      0.5      ]
 [2.      0.5      1.      1.      0.5      ]
 [0.5      0.33333333 1.      1.      0.5      ]
 [4.      2.      2.      2.      1.      ]]
权重向量 EV: [0.172769  0.20021897 0.16090765 0.10824147 0.35786291]
权重向量 LLSM: [0.16020622 0.19957385 0.16020622 0.11195645 0.36805725]
CSM 算法在迭代次数为 52 次时收敛
CSM 权重向量: [0.15088289 0.19804454 0.16081124 0.10539108 0.38487026]
TD EV: 11.41112202170799
TD LLSM: 11.376383332147004
TD CSM: 11.47657996163459
最小的 TD 值为 11.376383332147004, 选 LLSM 计算的权重向量为可信属性的权重向量。
Process finished with exit code 0

```

图 1: 权重向量

整理如下:

#### 解答 一 .1: 关键组件

正互反判断矩阵:

$$A = \begin{bmatrix} 1 & 2 & \frac{1}{2} & 2 & \frac{1}{4} \\ \frac{1}{2} & 1 & 2 & 3 & \frac{1}{2} \\ 2 & \frac{1}{2} & 1 & 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{3} & 1 & 1 & \frac{1}{2} \\ 4 & 2 & 2 & 2 & 1 \end{bmatrix}$$

方法	权重向量
EV	(0.1728, 0.2002, 0.1609, 0.1082, 0.3579)
LLSM	(0.1602, 0.1996, 0.1602, 0.1120, 0.3681)
CSM	(0.1509, 0.1980, 0.1608, 0.1054, 0.3849)

三个权重向量的值分别为  $TD^{EV} = 11.4111$ ,  $TD^{LLSM} = 11.3764$ ,  $TD^{CSM} = 11.4766$ 。

最小的 TD 值为 11.3764, 选 LLSM 计算的权重向量为可信属性的权重向量。

表 1: 关键组件正互反判断矩阵及权重

组件名	CP1	CP2	CP3	CP4	CP5	属性权重
CP1	1	2	1/2	2	1/4	0.1602
CP2	1/2	1	2	3	1/2	0.1996
CP3	2	1/2	1	1	1/2	0.1602
CP4	1/2	1/3	1	1	1/2	0.1120
CP5	4	2	2	2	1	0.3681

同理, 非关键组件代入矩阵 A, 得到的结果如下:

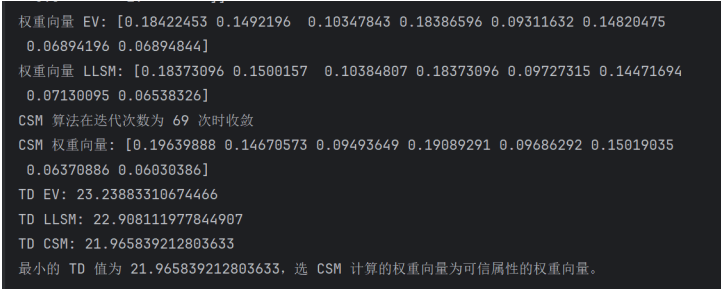


图 2: 权重向量

解答 一 .2: 非关键组件

正互反判断矩阵:

$$A = \begin{bmatrix} 1 & 3 & 2 & \frac{1}{2} & 2 & 1 & 3 & 3 \\ \frac{1}{3} & 1 & 2 & 1 & 2 & 2 & 2 & 2 \\ \frac{1}{2} & \frac{1}{2} & 1 & \frac{1}{2} & 1 & \frac{1}{2} & 3 & 3 \\ 2 & 1 & 2 & 1 & 3 & \frac{1}{2} & 3 & 3 \\ \frac{1}{2} & \frac{1}{2} & 1 & \frac{1}{3} & 1 & 1 & 2 & 2 \\ 1 & \frac{1}{2} & 2 & 2 & 1 & 1 & 2 & 2 \\ \frac{1}{3} & \frac{1}{2} & 1 & \frac{1}{3} & \frac{1}{2} & \frac{1}{2} & 1 & 2 \\ \frac{1}{3} & \frac{1}{2} & 2 & \frac{1}{3} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix}$$

方法	权重向量
EV	(0.1842, 0.1492, 0.1035, 0.1839, 0.0931, 0.1482, 0.0689, 0.0689)
LLSM	(0.1837, 0.1500, 0.1038, 0.1837, 0.0973, 0.1447, 0.0713, 0.0654)
CSM	(0.1964, 0.1467, 0.0949, 0.1909, 0.0969, 0.1502, 0.0637, 0.0603)

三个权重向量的值分别为  $TD^{EV} = 23.2388, TD^{LLSM} = 22.9081, TD^{CSM} = 21.9658$ 。  
最小的 TD 值为 21.9658, 选 CSM 计算的权重向量为可信属性的权重向量。

表 2: 非关键组件正互反判断矩阵及权重

组件名	CP6	CP7	CP8	CP9	CP10	CP11	CP12	CP13	属性权重
CP6	1	3	2	1/2	2	1	3	3	0.1964
CP7	1/3	1	2	1	2	2	2	2	0.1467
CP8	1/2	1/2	1	1/2	1	1/2	3	3	0.0949
CP9	2	1	2	1	3	1/2	3	3	0.1909
CP10	1/2	1/2	1	1/3	1	1	2	2	0.0969
CP11	1	1/2	2	2	1	1	2	2	0.1502
CP12	1/3	1/2	1	1/3	1/2	1/2	1	2	0.0637
CP13	1/3	1/2	2	1/3	1/2	1/2	1/2	1	0.0603

2 计算  $T_s$  的值

先写出整体最重要的函数部分, 如下所示:

```
def calculate_system_trustworthiness(  
    Critical_Confidence_value, Non_Critical_Confidence_value,  
    alpha, beta, FC, NFC  
):  
    ...
```

```

6  计算系统可信度值 T_S
7
8  参数：
9  - Critical_Confidence_value: 关键组件的可信值数组 (numpy array)
10 - Non_Critical_Confidence_value: 非关键组件的可信值数组 (numpy array)
11 - alpha: 关键组件权重系数
12 - beta: 非关键组件权重系数
13 - FC: 关键组件权重向量 (numpy array)
14 - NFC: 非关键组件权重向量 (numpy array)
15
16 返回：
17 - 系统可信度值 T_S
18 """
19 critical_product = np.prod(
20     np.power(Critical_Confidence_value, FC)
21 )
22 print(f"关键属性乘积: {critical_product}")
23
24 non_critical_product = np.prod(
25     np.power(Non_Critical_Confidence_value, NFC)
26 )
27 print(f"非关键属性乘积: {non_critical_product}")
28
29 T_S = alpha * critical_product + beta * non_critical_product
30
31 return T_S

```

$T_s$

接下来，我们可以计算关键组件的可信值数组和非关键组件的可信值数组，并输入到函数中：

```

1  alpha = 0.7
2  beta = 0.3
3  print(f"此时的 Alpha 值为 {alpha}, Beta 值为 {beta}")
4
5  T_S = calculate_system_trustworthiness(
6      Critical_Confidence_value, Non_Critical_Confidence_value,
7      alpha, beta, final_critical_weights, final_non_critical_weights
8  )
9
10 print(f"Ts 的值为: {T_S}")

```

$T_s$

调整  $\alpha$  和  $\beta$  的值，可以得到不同的  $T_s$  值。

示例输出如下：

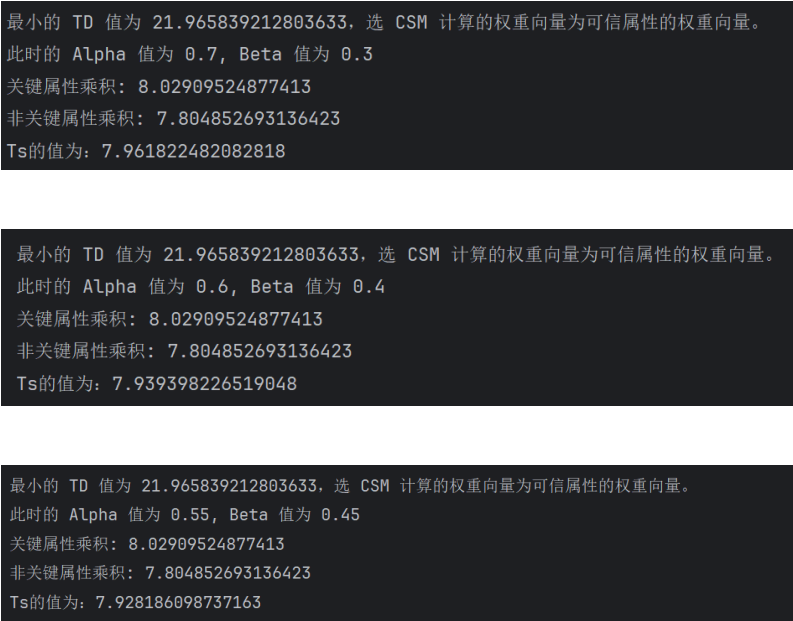


图 3: 不同 Ts 值的对比

最后, 我们可以制作为 Ts 的表格, 如下所示:

Alpha	Beta	$T_s$ 值
0.70	0.30	7.9618
0.60	0.40	7.9394
0.55	0.45	7.9282

3 可信等级

根据表格中的评判标准, 我们可以轻易看出

7.0 <= T < 8.5 或者 T > 8.5 且不能评为IV级别及以上者	1. 低于7.0分的关键组件个数 不超过 $n - \lceil n \times 2/3 \rceil$ 2. 没有低于4.5分的可信属性	III
--	--	-----

图 4: 可信等级

显然所有的关键属性都没有低于 7.0 分的, 而  $T_s \geq 7$  在不同的  $\alpha$  和  $\beta$  之下都成立。  
故可信软件等级为 **III**, 当然, 我们还可以编写代码来实现:

```
1 def classify_trust_level(T, scores):
2     """
3     分类可信等级
4     :param T: 系统可信度值
5     :param scores: 可信属性值数组
6     :return: 可信等级
7     """
8     n = len(scores)
9     threshold_2_3 = n - int(np.ceil(2 * n / 3)) # 计算 n - n * 2/3
10
11     # 统计低于某些阈值的属性个数
12     low_9_5 = np.sum(scores < 9.5)
13     low_8_5 = np.sum(scores < 8.5)
14     low_7_0 = np.sum(scores < 7.0)
15     low_4_5 = np.sum(scores < 4.5)
16
17     if T >= 9.5:
```

```

18     if low_9_5 <= threshold_2_3 and low_8_5 == 0:
19         return "V"
20
21     if 8.5 <= T < 9.5 or (T > 9.5):
22         if low_8_5 <= threshold_2_3 and low_7_0 == 0:
23             return "IV"
24
25     if 7.0 <= T < 8.5 or (T > 8.5):
26         if low_7_0 <= threshold_2_3 and low_4_5 == 0:
27             return "III"
28
29     if 4.5 <= T < 7.0 or (T > 7.0):
30         if low_4_5 <= threshold_2_3:
31             return "II"
32
33     if T < 4.5 or (T > 4.5):
34         return "I"

```

可信等级代码

## 二 附录

### 1 完整可执行代码

```

1  from pprint import pprint
2
3  import numpy as np
4
5  # 作业
6  Critical_Confidence_value = np.array([8.430, 8.530, 6.042, 9.094, 8.289])
7
8  Non_Critical_Confidence_value = np.array([6.192, 8.020, 7.984, 8.713, 9.211, 7.777, 7.897, 8.075])
9
10 Critical_A = np.array([
11     [1, 2, 1/2, 2, 1/4],
12     [1/2, 1, 2, 3, 1/2],
13     [2, 1/2, 1, 1, 1/2],
14     [1/2, 1/3, 1, 1, 1/2],
15     [4, 2, 2, 2, 1]
16 ])
17
18 Non_Critical_A = np.array([
19     [1, 3, 2, 1/2, 2, 1, 3, 3],
20     [1/3, 1, 2, 1, 2, 2, 2, 2],
21     [1/2, 1/2, 1, 1/2, 1, 1/2, 3, 3],
22     [2, 1, 2, 1, 3, 1/2, 3, 3],
23     [1/2, 1/2, 1, 1/3, 1, 1, 2, 2],
24     [1, 1/2, 2, 2, 1, 1, 2, 2],
25     [1/3, 1/2, 1, 1/3, 1/2, 1/2, 1, 2],
26     [1/3, 1/2, 2, 1/3, 1/2, 1/2, 1/2, 1]
27 ])
28
29 # Test Data
30 #
31 # Critical_Confidence_value = np.array([9.536, 7.531, 9.167, 8.423])
32 #
33 # Non_Critical_Confidence_value = np.array([7.556, 7.858, 8.979, 7.708, 9.265])
34 #
35 # Critical_A = np.array([
36 #     [1, 2, 1/2, 1/3],
37 #     [1/2, 1, 1/2, 1/3],
38 #     [2, 2, 1, 1/2],

```

```

39 # [3, 3, 2, 1]
40 # ])
41 #
42 # Non_Critical_A = np.array([
43 # [1, 1/2, 1, 1/3, 1/2],
44 # [2, 1, 2, 1/2, 1],
45 # [1, 1/2, 1, 1/2, 1],
46 # [3, 2, 2, 1, 2],
47 # [2, 1, 1, 1/2, 1]
48 # ])
49
50
51 def classify_trust_level(T, scores):
52     """
53     分类可信等级
54     :param: T: 系统可信度值
55     :param: scores: 可信属性值数组
56     :return: 可信等级
57     """
58     n = len(scores)
59     threshold_2_3 = n - int(np.ceil(2 * n / 3)) # 计算  $n - n \times 2/3$ 
60
61     # 统计低于某些阈值的属性个数
62     low_9_5 = np.sum(scores < 9.5)
63     low_8_5 = np.sum(scores < 8.5)
64     low_7_0 = np.sum(scores < 7.0)
65     low_4_5 = np.sum(scores < 4.5)
66
67     if T >= 9.5:
68         if low_9_5 <= threshold_2_3 and low_8_5 == 0:
69             return "V"
70
71     if 8.5 <= T < 9.5 or (T > 9.5):
72         if low_8_5 <= threshold_2_3 and low_7_0 == 0:
73             return "IV"
74
75     if 7.0 <= T < 8.5 or (T > 8.5):
76         if low_7_0 <= threshold_2_3 and low_4_5 == 0:
77             return "III"
78
79     if 4.5 <= T < 7.0 or (T > 7.0):
80         if low_4_5 <= threshold_2_3:
81             return "II"
82
83     if T < 4.5 or (T > 4.5):
84         return "I"
85
86
87 def calculate_system_trustworthiness(
88     Critical_Confidence_value, Non_Critical_Confidence_value,
89     alpha, beta, FC, NFC
90 ):
91     """
92     计算系统可信度值 T_S
93
94     参数:
95     - Critical_Confidence_value: 关键组件的可信值数组 (numpy array)
96     - Non_Critical_Confidence_value: 非关键组件的可信值数组 (numpy array)
97     - alpha: 关键组件权重系数
98     - beta: 非关键组件权重系数
99     - FC: 关键组件权重向量 (numpy array)
100     - NFC: 非关键组件权重向量 (numpy array)
101
102     返回:

```



```

103     - 系统可信度值 T_S
104     """
105     critical_product = np.prod(
106         np.power(Critical_Confidence_value, FC)
107     )
108     print(f"关键属性乘积: {critical_product}")
109
110     non_critical_product = np.prod(
111         np.power(Non_Critical_Confidence_value, NFC)
112     )
113     print(f"非关键属性乘积: {non_critical_product}")
114
115     T_S = alpha * critical_product + beta * non_critical_product
116
117     return T_S
118
119 def calculate_ev_weights(matrix):
120     eigenvalues, eigenvectors = np.linalg.eig(matrix) # 返回值为元组, 第一个元素为特征值, 第二个元素为特征
121     向量
122     max_index = np.argmax(eigenvalues) # 取最大特征值对应的特征向量
123     weights = eigenvectors[:, max_index].real
124     # 归一化
125     return weights / np.sum(weights)
126
127 def calculate_llsm_weights(matrix):
128     # 分子 = 每行的乘积 × 开维度数次方
129     numerator = np.prod(matrix, axis=1) ** (1 / matrix.shape[0])
130     # 分母 = 每行的分子相加
131     denominator = np.sum(numerator)
132     # 归一化
133     return numerator / denominator
134
135 def calculate_csm_weights(matrix: np.array, epsilon: float, max_iterations: int) -> np.ndarray:
136     '''计算 CSM, 传入: matrix, precision, maximum iterations'''
137     n = matrix.shape[0]
138     # 初始化初始解
139     W = np.ones(n) / n
140
141     for k in range(max_iterations):
142         e = np.zeros(n)
143         for i in range(n):
144             e[i] = np.sum([(1 + matrix[j, i] ** 2) * (W[i] / W[j]) - (1 + matrix[i, j] ** 2) * (W[j] / W[i])
145                             for j in range(n) if j != i])
146
147         max_e = np.max(np.abs(e))
148         if max_e <= epsilon: # 若精度已到达, 那么停止迭代
149             print(f"CSM 算法在迭代次数为 {k} 次时收敛")
150             break
151
152         m = np.argmax(np.abs(e)) # 查找最大无差的索引
153
154         # 计算 T(k)
155         up = np.sum([(1 + matrix[m, j] ** 2) * (W[j] / W[m]) for j in range(n) if j != m])
156         bottom = np.sum([(1 + matrix[j, m] ** 2) * (W[m] / W[j]) for j in range(n) if j != m])
157         T = np.sqrt(up / bottom)
158
159         # 更新矩阵向量, 归一化
160         X = W.copy()
161         X[m] *= T
162         W = X / np.sum(X)
163
164     return W
165
166 def calculate_td(matrix: np.array, weight_vector: np.ndarray) -> float:

```

```

166     n = len(weight_vector)
167     td = 0.0
168     for i in range(n):
169         for j in range(n):
170             td += abs(matrix[i, j] - (weight_vector[i] / weight_vector[j]))
171     return td
172
173
174 print("-----")
175 print("以下是关键组件的判断矩阵：")
176
177
178 print("判断矩阵:")
179 pprint(Critical_A)
180 print("-----")
181
182 weights_ev = calculate_ev_weights(Critical_A)
183 print("权重向量 EV:", weights_ev)
184
185 weights_llsm = calculate_llsm_weights(Critical_A)
186 print("权重向量 LLSM:", weights_llsm)
187
188 weights_csm = calculate_csm_weights(Critical_A, 1e-10, 1000)
189 print("CSM 权重向量:", weights_csm)
190
191 TD_EV = calculate_td(Critical_A, weights_ev)
192 TD_LLSM = calculate_td(Critical_A, weights_llsm)
193 TD_CSM = calculate_td(Critical_A, weights_csm)
194
195 print("TD EV:", TD_EV)
196 print("TD LLSM:", TD_LLSM)
197 print("TD CSM:", TD_CSM)
198
199 td_values = {"EV": TD_EV, "LLSM": TD_LLSM, "CSM": TD_CSM}
200 min_method = min(td_values, key=td_values.get)
201 print(f"最小的 TD 值为 {td_values[min_method]}, 选 {min_method} 计算的权重向量为可信属性的权重向量。")
202 final_critical_weights = weights_ev if min_method == "EV" else weights_llsm if min_method == "LLSM" else
weights_csm
203
204 print("-----")
205 print("以下是非关键组件的判断矩阵：")
206
207
208 print("判断矩阵:")
209 pprint(Non_Critical_A)
210 print("-----")
211
212 weights_ev = calculate_ev_weights(Non_Critical_A)
213 print("权重向量 EV:", weights_ev)
214
215 weights_llsm = calculate_llsm_weights(Non_Critical_A)
216 print("权重向量 LLSM:", weights_llsm)
217
218 weights_csm = calculate_csm_weights(Non_Critical_A, 1e-10, 1000)
219 print("CSM 权重向量:", weights_csm)
220
221 TD_EV = calculate_td(Non_Critical_A, weights_ev)
222 TD_LLSM = calculate_td(Non_Critical_A, weights_llsm)
223 TD_CSM = calculate_td(Non_Critical_A, weights_csm)
224
225 print("TD EV:", TD_EV)
226 print("TD LLSM:", TD_LLSM)
227 print("TD CSM:", TD_CSM)
228
229 td_values = {"EV": TD_EV, "LLSM": TD_LLSM, "CSM": TD_CSM}

```

```
229 min_method = min(td_values, key=td_values.get)
230 # min_method = "LLSM"
231 print(f"最小的 TD 值为 {td_values[min_method]}, 选 {min_method} 计算的权重向量为可信属性的权重向量。")
232 final_non_critical_weights = weights_ev if min_method == "EV" else weights_llsm if min_method == "LLSM" else
    weights_csm
233
234 alpha = 0.6
235 beta = 0.4
236 print(f"此时的 Alpha 值为 {alpha}, Beta 值为 {beta}")
237
238 T_S = calculate_system_trustworthiness(
239     Critical_Confidence_value, Non_Critical_Confidence_value,
240     alpha, beta, final_critical_weights, final_non_critical_weights
241 )
242
243 print(f"Ts的值为: {T_S}")
244 Level = classify_trust_level(T_S, Critical_Confidence_value)
245 print("该软件的可信等级为: " + Level)
```

完整代码