



THE ICS DEMPSTER-SHAFER HOW TO

28.10.2023

Submitted To : Subrina Akter - Assistant Professor Dept of CSE.
International Islamic University of Chittagong.

Submitted By : Sufia Akter (C173203R) & Sadia Anjum (C211255)
Department : B.Sc in CSE.

Semester & Section : 6th semester section of 6BF

Introduction :

This report discusses the application of Dempster-Shafer theory and the Evidential Reasoning Algorithm for Multiple Attribute Decision Analyses. The focus is on managing uncertainties in assessments when used in multi-attribute decision analyses. The paper also explores the use of Architecture Theory Diagrams (ATD) and provides an example of assessing the property data quality using an ATD.

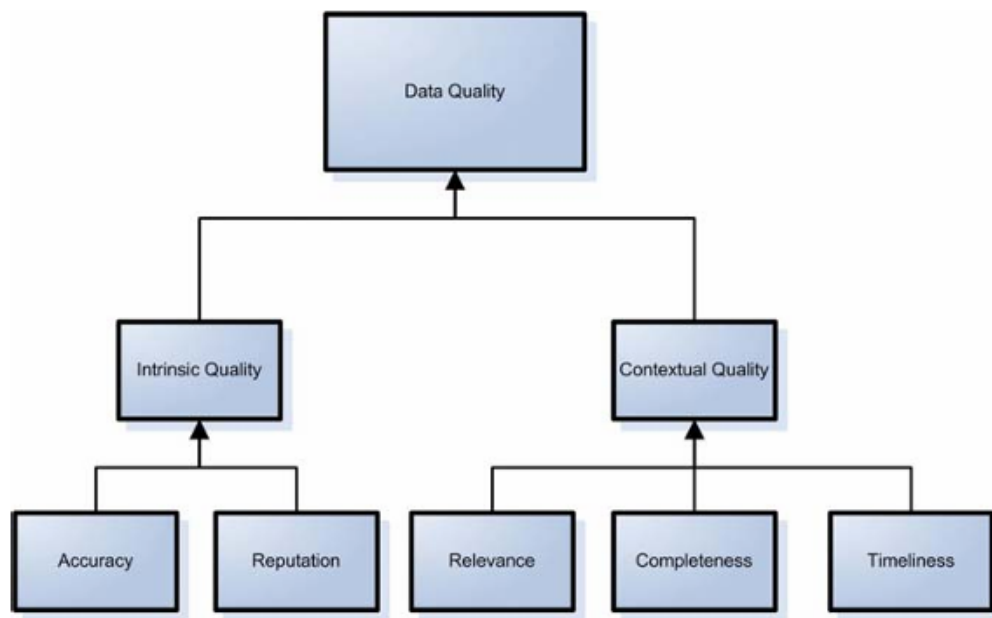



Figure 1. Evaluation of data quality represented as an ATD with three levels.

Problem Analysis :

In this analysis we will discuss how this problem works , which algorithms are used and what is the advantage and limitation of this problem.

- 
- ❖ **How it works :** This work presents a six-step process for applying Dempster-Shafer theory and the Evidential Reasoning Algorithm. The process involves
 - defining and representing a multiple attribute decision problem
 - assigning basic probability
 - combining probability assignments
 - calculating combined degrees of belief
 - representing the distributed overall assessment
 - calculating utility intervals of incomplete assessments.
 - ❖ **Algorithm used :** The project uses the following algorithms:
 - **Dempster-Shafer Theory:** This is a mathematical theory of evidence used to combine separate pieces of information (or evidence) to calculate the probability of an event.
 - **Evidential Reasoning Algorithm:** This algorithm is used for Multiple Attribute Decision Analyses. It provides a rational process to generate an overall assessment by aggregating subjective judgments.
 - ❖ **Advantage :** The main advantage of this project is its ability to manage uncertainties in assessments when used in multi-attribute decision analyses. It provides a systematic way to aggregate subjective judgments to generate an overall assessment.
 - ❖ **Limitation :** One limitation is that it assumes that the utilities of the respective assessment grade can be appreciated in a linear fashion. This may not always be the case in real-world scenarios where utilities can have non-linear relationships.

Implementation :

❖ Usage Language : Python .

❖ Code :

```
#function for step 2 calculation
def calculate_probability_of_mass(weights, degree_of_belief):
    probability_mass = []
    m_tilde_H = [] # Initialize m_tilde_H as a list
    mH = []
    m_bar_H = []

    for i in range(len(weights)):
        p = []
        for j in range(3):
            p.append(round(weights[i] * degree_of_belief[i][j], 4))
        p.append(1 - weights[i])
        p.append(round(weights[i] * degree_of_belief[i][3], 4))
        p.insert(3, round(1 - weights[i] + p[-1], 4))
        probability_mass.append(p)
        # Calculate m_tilde_H for each attribute  $\epsilon_i$ 
        m_tilde_H_i = weights[i] * (1 - sum(degree_of_belief[i]))
        m_tilde_H.append(round(weights[i] * degree_of_belief[i][3],
4))

        # Calculate remaining probability mass mH for each basic
attribute  $\epsilon_i$ 
        mH_i = 1 - sum(p[j] for j in range(3))
        mH.append(mH_i)

        # Decompose mH into  $\bar{m}H$  for each basic attribute  $\epsilon_i$ 
        m_bar_H_i = 1 - weights[i]
        m_bar_H.append(m_bar_H_i)
```

```

    return probability_mass, mH, m_bar_H, m_tilde_H

# function for constant calculation
def calculate_constant_k(probability_masses):
    k = [None]

    for i in range(1, len(probability_masses)):
        total_k = 0
        for t in range(3): # t represents indices for N
            for j in range(3): # j represents indices for N
                if t == j:
                    continue
                total_k += probability_masses[i - 1][t] *
probability_masses[i][j]
            total_k = 1 - total_k
            k_i = 1 / total_k
            k.append(round(k_i, 5)) # Round to four decimal places as
required

    return k

#step 3 calculation
def calculate_probability_mass_aggregation(probability_masses, m_H,
m_bar_H, m_tilde_H, constant):
    PoMa = [None] # Initialize with a None placeholder

    updated_probability_masses = probability_masses.copy()
    updated_m_H = m_H.copy()
    updated_m_bar_H = m_bar_H.copy()
    updated_m_tilde_H = m_tilde_H.copy()

    for i in range(1, len(updated_probability_masses)):
        pma = []

        for j in range(3):
            p = constant[i] * (updated_probability_masses[i - 1][j] *
updated_probability_masses[i][j] +

```

```

                                updated_m_H[i - 1] *
updated_probability_masses[i][j] +
                                updated_probability_masses[i - 1][j] *
updated_m_H[i])
    pma.append(round(p, 4))

    m_bar = constant[i] * (updated_m_bar_H[i - 1] *
updated_m_bar_H[i])
    pma.append(round(m_bar, 4))

    m_tilde = constant[i] * (updated_m_tilde_H[i - 1] *
updated_m_tilde_H[i] +
                                updated_m_bar_H[i - 1] *
updated_m_tilde_H[i] +
                                updated_m_tilde_H[i - 1] *
updated_m_bar_H[i])
    pma.append(round(m_tilde, 4))

    total_m = round(m_bar + m_tilde, 4)
    pma.insert(3, total_m)

    pma = [round(val, 4) for val in pma]

    PoMa.append(pma)

    # Update the values in the updated variables
    updated_probability_masses[i] = pma
    updated_m_H[i] = m_bar + m_tilde
    updated_m_bar_H[i] = m_bar
    updated_m_tilde_H[i] = m_tilde

    return PoMa

#step 4 calculation
def calculate_combined_degrees_of_belief(PoMa):
    combined_degrees_of_belief = []

    m_n_L = PoMa[-1][: -3] # Get the last row of PoMa without the

```

```

last element
    m_bar = PoMa[-1][-2] # Get the last element of the last row of
PoMa
    m_tilda = PoMa[-1][-1] # Get the last element of the last row of
PoMa

    for n in range(len(m_n_L)):
        # Calculate  $\beta_n$ 
        beta_n = m_n_L[n] / (1 - m_bar)
        combined_degrees_of_belief.append(round(beta_n, 4))

    # Calculate  $\beta_H$ 
    beta_H = m_tilda / (1 - m_bar)
    combined_degrees_of_belief.append(round(beta_H, 4))

    return combined_degrees_of_belief

#step 5 calculation for estimate utilities
def estimate_utilities(assessment_grades):
    """
    Estimates utilities for different assessment grades. This
    function returns a dictionary where the keys are the
    assessment grades and the values are the estimated utilities.

    :param assessment_grades: A dictionary where keys are assessment
    grades (H1, H2, H3, etc.) and values are the
        corresponding utility values.
    :return: A dictionary of utilities for each assessment grade.
    """
    utilities = {}
    for grade, utility in assessment_grades.items():
        utilities[grade] = utility
    return utilities

#step 5 calculation for expected utility
def calculate_expected_utility(assessment_weights, utilities):
    """
    Calculate the expected utility for a complete assessment using

```

the provided assessment weights and utilities.

```

:param assessment_weights: A list of assessment weights ( $\beta_1$ ,  $\beta_2$ ,  $\beta_3$ , etc.).
:param utilities: A dictionary of utilities for each assessment grade.
:return: The expected utility for the complete assessment.
"""

```

```

    expected_utility = sum(assessment_weights[i] * utilities[f'H{i + 1}'] for i in range(len(assessment_weights)))
    return expected_utility

```

#step 5 calculation for utility interval

```

def calculate_utility_interval(assessment_weights, utilities, unassigned_belief):
    """

```

Calculate the utility interval for incomplete assessments.

```

:param assessment_weights: A list of assessment weights ( $\beta_1$ ,  $\beta_2$ ,  $\beta_3$ , etc.).
:param utilities: A dictionary of utilities for each assessment grade.
:param unassigned_belief: The unassigned belief degree ( $\beta_H$ ).
:return: A tuple (u_min, u_max, u_avg) representing the utility interval.
"""

```

```

    u_max = sum(assessment_weights[i] * utilities[f'H{i + 1}'] for i in range(len(assessment_weights) - 1)) + \
              (assessment_weights[-1] + unassigned_belief) *
    utilities[f'H{len(assessment_weights)}']

```

```

    u_min = (assessment_weights[0] + unassigned_belief) *
    utilities['H1'] + \
              sum(assessment_weights[i] * utilities[f'H{i + 1}'] for i in range(1, len(assessment_weights)))

```

```

    u_avg = (u_max + u_min) / 2

```



```

    return u_min, u_max, u_avg

#output format
def print_results(data_type, weights, degree_of_belief,
probability_masses, mH, m_bar_H, k, PoMa, combined_degrees):
    print(f"{data_type} Data Quality:")
    print("Weight =", weights)
    print("Belief =", degree_of_belief)
    print("Probability Mass =", probability_masses)
    print("Constant =", k)
    print("Probability Mass (aggregation) =", PoMa)
    print("Combined Degrees of Belief =", combined_degrees)

# Define intrinsic weights and degree of belief
intrinsic_weights = [0.35, 0.65]
intrinsic_degree_of_belief = [[0.4, 0.5, 0.0, 0.1], [0.1, 0.75, 0.15,
0.0]]

# Calculate intrinsic data quality
intrinsic_probability_masses, mH_intrinsic, m_bar_H_intrinsic,
m_tilde_H_intrinsic = calculate_probability_of_mass(
    intrinsic_weights, intrinsic_degree_of_belief)
k_intrinsic = calculate_constant_k(intrinsic_probability_masses)
PoMa_intrinsic =
calculate_probability_mass_aggregation(intrinsic_probability_masses,
mH_intrinsic,

m_bar_H_intrinsic, m_tilde_H_intrinsic, k_intrinsic)
combined_degrees_intrinsic =
calculate_combined_degrees_of_belief(PoMa_intrinsic)

# Define contextual weights and degree of belief
contextual_weights = [0.45, 0.25, 0.3]
contextual_degree_of_belief = [[0.6, 0.2, 0.05, 0.15], [0.25, 0.45,
0.3, 0.0], [0.55, 0.35, 0.0, 0.1]]

# Calculate contextual data quality

```

```

contextual_probability_masses, m_H_contextual, m_bar_H_contextual,
m_tilde_H_contextual = calculate_probability_of_mass(
    contextual_weights, contextual_degree_of_belief)
k_contextual = [None, 1.07174, 1.0797]

PoMa_contextual =
calculate_probability_mass_aggregation(contextual_probability_masses,
m_H_contextual,

m_bar_H_contextual, m_tilde_H_contextual, k_contextual)
combined_degrees_contextual =
calculate_combined_degrees_of_belief(PoMa_contextual)

# Define data quality's weight and degree of belief
data_weights = [0.6, 0.4]
data_degree_of_belief = [combined_degrees_intrinsic,
combined_degrees_contextual]

# Calculate data quality
data_probability_masses, m_H_data, m_bar_H_data, m_tilde_H_data =
calculate_probability_of_mass(
    data_weights, data_degree_of_belief)
k_data = [None, 1.16499]

PoMa_data =
calculate_probability_mass_aggregation(data_probability_masses,
m_H_data,

m_bar_H_data,
m_tilde_H_data, k_data)
combined_degrees_data =
calculate_combined_degrees_of_belief(PoMa_data)

# Print the results for intrinsic data quality
print_results("Intrinsic", intrinsic_weights,
intrinsic_degree_of_belief, intrinsic_probability_masses,
m_H_intrinsic, m_bar_H_intrinsic, k_intrinsic,
PoMa_intrinsic, combined_degrees_intrinsic)

```

```

# Print the results for contextual data quality
print_results("Contextual", contextual_weights,
contextual_degree_of_belief, contextual_probability_masses,
              m_H_contextual, m_bar_H_contextual, k_contextual,
PoMa_contextual, combined_degrees_contextual)

print_results("Final", data_weights, data_degree_of_belief,
data_probability_masses,
              m_H_data, m_bar_H_data, k_data, PoMa_data,
combined_degrees_data)

print("-----")

# Define utilities for assessment grades (H1, H2, H3, etc.)
assessment_utilities = {'H1': 0, 'H2': 0.5, 'H3': 1}

# Calculate expected utility for complete assessments
complete_assessment_weights = combined_degrees_data[:3]

# Define unassigned belief ( $\beta_H$ ) for incomplete assessments
unassigned_belief = combined_degrees_data[-1]

expected_utility =
calculate_expected_utility(complete_assessment_weights,
assessment_utilities)
print(f"Expected Utility for Complete Assessment:
{expected_utility:.4f}")

# Calculate utility interval for incomplete assessments
incomplete_assessment_weights = complete_assessment_weights #
Assuming the same weights
u_min, u_max, u_avg =
calculate_utility_interval(incomplete_assessment_weights,
assessment_utilities, unassigned_belief)
print(f"Utility Interval (u_min): {u_min:.4f}")
print(f"Utility Interval (u_max): {u_max:.4f}")

```

```

print(f"Utility Interval (u_avg): {u_avg:.4f}")

# Define your thresholds
Threshold_Good = 0.8 # Adjust this value as needed
Threshold_Poor = 0.4 # Adjust this value as needed

# Make a decision based on the thresholds
decision = ""
if u_avg >= Threshold_Good:
    decision = "Good"
elif u_avg < Threshold_Poor:
    decision = "Poor"
else:
    decision = "Average"

# Print the decision
print(f"-----\nData Quality Decision: {decision}")

```

Result and Analysis :

❖ **Output :** Here is out project's output

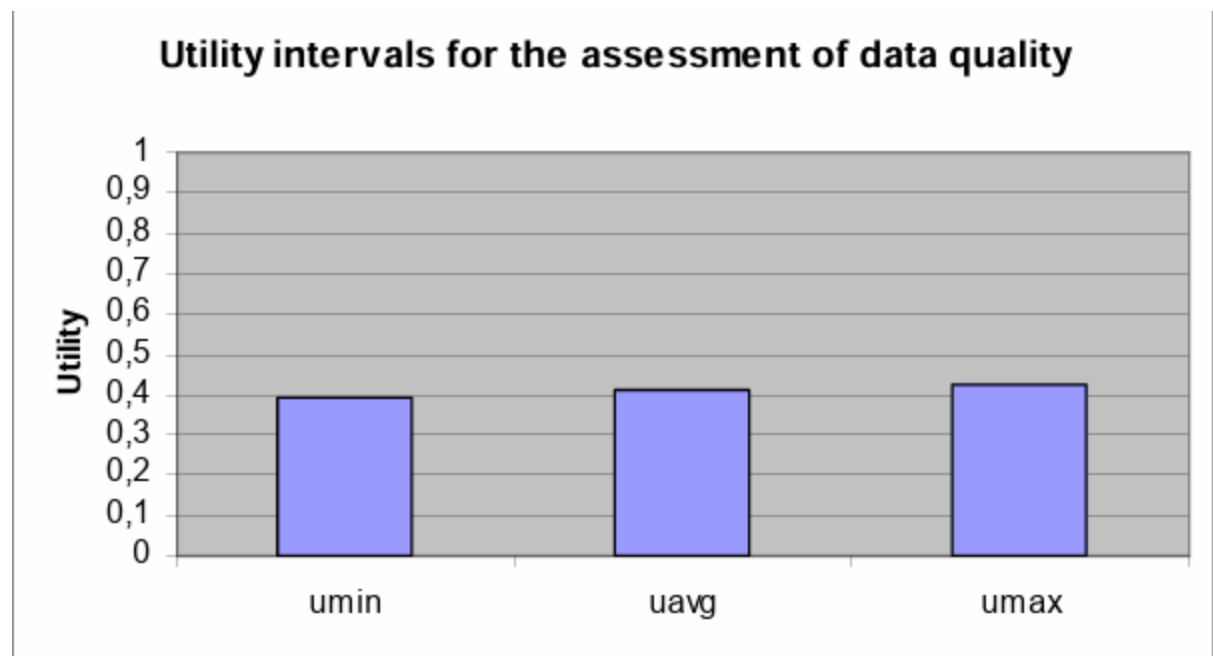
```


D:\laragon\bin\python\python-3.10\python.exe "D:\vrsty temp\ai\project\dst_full.py"
Intrinsic Data Quality:
Weight = [0.35, 0.65]
Belief = [[0.4, 0.5, 0.0, 0.1], [0.1, 0.75, 0.15, 0.0]]
Probability Mass = [[0.14, 0.175, 0.0, 0.685, 0.65, 0.035], [0.065, 0.4875, 0.0975, 0.35, 0.35, 0.0]]
Constant = [None, 1.12402]
Probability Mass (aggregation) = [None, [0.1154, 0.5401, 0.0751, 0.2695, 0.2557, 0.0138]]
Combined Degrees of Belief = [0.155, 0.7256, 0.1009, 0.0185]
Contextual Data Quality:
Weight = [0.45, 0.25, 0.3]
Belief = [[0.6, 0.2, 0.05, 0.15], [0.25, 0.45, 0.3, 0.0], [0.55, 0.35, 0.0, 0.1]]
Probability Mass = [[0.27, 0.09, 0.0225, 0.6175, 0.55, 0.0675], [0.0625, 0.1125, 0.075, 0.75, 0.75, 0.0], [0.165, 0.105, 0.0, 0.73, 0.7, 0.03]]
Constant = [None, 1.07174, 1.0797]
Probability Mass (aggregation) = [None, [0.2765, 0.1576, 0.0695, 0.4963, 0.4421, 0.0543], [0.3556, 0.1984, 0.0548, 0.3912, 0.3341, 0.0571]]
Combined Degrees of Belief = [0.534, 0.2979, 0.0823, 0.0857]
Final Data Quality:
Weight = [0.6, 0.4]
Belief = [[0.155, 0.7256, 0.1009, 0.0185], [0.534, 0.2979, 0.0823, 0.0857]]
Probability Mass = [[0.093, 0.4354, 0.0605, 0.4111, 0.4, 0.0111], [0.2136, 0.1192, 0.0329, 0.6343, 0.6, 0.0343]]
Constant = [None, 1.16499]
Probability Mass (aggregation) = [None, [0.1942, 0.4393, 0.0628, 0.3038, 0.2796, 0.0242]]
Combined Degrees of Belief = [0.2696, 0.6098, 0.0872, 0.0336]
-----

```

```
-----  
Expected Utility for Complete Assessment: 0.3921  
Utility Interval (u_min): 0.3921  
Utility Interval (u_max): 0.4257  
Utility Interval (u_avg): 0.4089  
-----  
Data Quality Decision: Average  
  
Process finished with exit code 0
```

- ❖ **Analysis :** The application of Dempster-Shafer theory and the Evidential Reasoning Algorithm allows for a comprehensive analysis of data quality using an ATD. The result is a utility interval that provides a measure of the quality of data based on multiple attributes. The utility interval is calculated under the assumption of incomplete assessments, which is a common occurrence due to high data collection costs.





❖ **Final Decision :** According to our project output and paper's graph we can say ,given data of data quality is average.

Conclusion :

Dempster-Shafer theory and the Evidential Reasoning Algorithm provide a robust framework for managing uncertainties in assessments when used in multi-attribute decision analyses. Despite some limitations, their application can significantly enhance decision-making processes by providing more accurate and comprehensive assessments.

Our Project : [gitHub link](#)