Journal of Fuzzy Extension and Applications

www.journal-fea.com

J. Fuzzy. Ext. Appl. Vol. 6, No. 1 (2025) 190–215.

Paper Type: Research Paper

A Comprehensive Python Tool for Interval Valued Bipolar Neutrosophic Sets and Operations

Pranesh Kumar S Prakash[1] , Sofia Jennifer John[2] , Kalaivani Chandran[3] and Florentin Smarandache[4]

[1]Department of Information Technology, Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam 603110, India; praneshkumar2210773@ssn.edu.in

[2]Department of Information Technology, Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam 603110, India; sofiajenniferj@ssn.edu.in

[3]Department of Mathematics, Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam 603110, India; kalaivanic@ssn.edu.in.

[4]Department of Mathematics and Science, University of New Mexico, Gallup, NM 87301, USA; smarand@unm.edu

Citation:

Received: 19-02-2024 Revised: 20-04-2024 Accepted: 11-7-2024	Prakash, P. K. S., John, S. J., Chandran, K., & Smarandache, F. (2024). A Comprehensive Python Tool for Interval Valued Bipolar Neutro-sophic Sets and Operations. *Journal of Fuzzy Extension and Applications,6*(1), 190-215

Abstract

The increasing prominence of single-valued neutrosophic sets (SVNSs) and interval-valued neutrosophic sets (IVNSs) among researchers has propelled their adoption in diverse real-world applications. These concepts effectively tackle the inherent indeterminacy and inconsistent information prevalent in many practical scenarios. While applications for IVNS environments have been established, this chapter proposes a novel Python toolbox specifically designed for handling Interval Valued Bipolar Neutrosophic Sets (IVBNSs) within the neutrosophic framework. The abstract provides comprehensive definitions for SVNSs, IVNSs, and IVBNSs, along with detailed descriptions of their respective operations. Additionally, the toolbox's implementation and usage are vividly illustrated through practical examples.

Keywords: Exchange option, Lookback option, Black–Scholes Option pricing, Mesh-free method.

Corresponding Author: kalaivanic@ssn.edu.in

https://doi.org/10.22105/jfea.2024.446338.1395

1|Introduction

Real-world problems are inherently complex and often riddled with uncertainty and inconsistency. Traditional set-theoretic approaches, which rely on crisp boundaries between sets, struggle to effectively model these complexities. Fuzzy set theory, introduced by Zadeh[27], offered a significant advancement by allowing elements to have partial membership in a set. However, even fuzzy sets[27] have limitations in capturing the nuances of uncertainty.

Atanassov[3, 2] is credited with introducing intuitionistic fuzzy sets, a specific type of fuzzy set that lays some groundwork for understanding neutrosophic concepts. In response to limitations of classical fuzzy sets, Smarandache extended the framework to form neutrosophic sets, including Single-valued neutrosophic sets (SVNSs)[26] and Interval-valued neutrosophic sets (IVNSs). These neutrosophic sets provide a more general framework to model uncertainty. They extend classical fuzzy sets by incorporating truth (T), indeterminacy (I), and falsity (F) degrees, allowing for a richer representation of information. This generalization has fueled applications in various domains, such as decision-making[25, 4, 18], image processing[23, 24], pattern recognition[8] in fields like stock markets[6] and investment fidelities[8], cognitive modelling[9], performance evaluation and assessment[17, 16] and medical diagnosis[22, 23, 1].

Despite their significance, these neutrosophic concepts have limitations in addressing certain types of uncertainty and inconsistency. For instance, SVNSs and IVNSs cannot adequately represent situations where the truth values have both positive and negative components. This limitation arises from the fact that SVNSs and IVNSs only consider truth, indeterminacy, and falsity, while neglecting the possibility of both positive and negative aspects of uncertainty.

To overcome this limitation, the concept of interval-valued bipolar neutrosophic sets (IVBNSs) was introduced. IVBNSs extend IVNSs by incorporating an additional value, representing the bipolar aspect of uncertainty. This addition allows for a more comprehensive and nuanced representation of uncertainty, enabling the modelling of situations where both positive and negative aspects of uncertainty are present. Various versions of this has also been used for solving linear programming problems and shortest path problems.[12, 11, 14]

Traditional set theory struggles to represent the inherent complexities of real-world problems. While fuzzy set theory offers a significant advancement in modeling uncertainty by allowing partial membership in sets it falls short in capturing situations with both positive and negative aspects of uncertainty. Smarandache developed a more general framework that offered a richer representation of information known as neutrosophic sets.

However, the existing neutrosophic concepts like single-valued neutrosophic sets (SVNSs)[26] and interval-valued neutrosophic sets (IVNSs)[5] have limitations in handling bipolar truth values. They struggle to represent situations where the truth value has both positive and negative aspects. For instance, in the medical domain, on diagnosis, a symptom might exhibit both favorable and unfavorable indicators for a disease.

To overcome these limitations and enable a more comprehensive representation of uncertainty, the concept of interval-valued bipolar neutrosophic sets (IVBNSs) was introduced. IVBNSs extend IVNSs by incorporating an additional truth value, allowing the modeling of both positive and negative aspects of uncertainty. Despite the theoretical advantages of IVBNSs, their practical application is hindered by the lack of user-friendly computational tools. This gap restricts researchers and practitioners from fully utilizing the potential of IVBNSs and the proposed idea aims to bridge this gap.

On exploring the comparative landscape of toolboxes dedicated to Interval Valued Bipolar Neutrosophic Sets (IVBNS) analysis and processing, the only framework available in the market is the Matlab platform. The absence of a wider range of comparable tools underscores the novelty and potential significance of the Python toolbox developed in this study. In terms of cost and accessibility, Matlab is proprietary with license costs whereas the Python toolbox built is open-source and it allows the users to add and modify the source at free cost. Secondly concerning size and resource utilization, the Python toolbox runs effectively even on systems with limited computational resources and the proposed IVBNS framework involves no specific modules being imported, and the implementation is written in simple methods hence the storage space is less than 17KB whereas MATLAB is more resource-intensive, potentially impacting performance on lower-end machines. Thirdly the proposed toolbox offers a user-friendly GUI, simplifying interaction and reducing the learning curve for users unfamiliar with programming languages. In contrast, the existing MATLAB toolbox relies on a legacy interface, tailored for users with programming expertise to navigate and utilize its functionalities effectively.

The proposed toolbox provides a comprehensive collection of functions for defining, manipulating, and analysing IVBNSs. It also includes a command line user interface for realizing IVBNSs and their operations. The toolbox is designed to be user-friendly and easy to use, making it accessible to a wide range of users, including researchers, practitioners, and students.

Despite the theoretical advantages of IVBNSs, the lack of user-friendly computational tools hinders their practical application. This paper aims to bridge this gap by proposing a novel Python toolbox specifically designed for handling and manipulating IVBNSs.

This paper is organized as:

- A comprehensive introduction to SVNSs, IVNSs, and IVBNSs.
- A detailed description of the operations defined on IVBNSs.
- Implementation of a Python toolbox for handling IVBNSs.
- A command line interface for realizing IVBNSs and their operations.
- Demonstration of the toolbox's capabilities through examples.

We believe this proposed toolbox will significantly enhance the applicability of IVBNSs in various real-world domains, empowering researchers, and practitioners with a powerful tool for modelling and analysing uncertainty and inconsistency.

2|Interval Valued Bipolar Neutrosophic Sets and its Implementation

We recall some basic definitions from the literature. We also give the python codes for the definitions. The python implementations given are a slighlty modified version of [10]

Definition 0.1. [26] *Let X be a universe of discourse. Then a single valued neutrosophic set is defined as:*

$$A_{NS} = \{\langle x, T_A(x), I_A(x), F_A(x)\rangle : x \in X\}$$

which is characterised by a truth-membership function $T_A(x) : X \to [0,1]$, indeterminacy-membership function $I_A(x) : X \to [0,1]$, and a falsity-membership function $F_A(x) : X \to [0,1]$. There is no restriction on the sum of $T_A(x), I_A(x), F_A(x)$ so $0 \leq (T_A(x) + I_A(x) + F_A(x)) \leq 3$

Here each element in Single Valued Neutrosophic Set is refered to as Single Valued Neutrosophic Number.

Definition 0.2. [7] *A bipolar neutrosophic set A in X is defined as an object of the form*

$$A = \left\{\left\langle x, T^+(x), I^+(x), F^+(x), T^-(x), I^-(x), F^-(x)\right\rangle : x \in X\right\}$$

where $T^+, I^+, F^+ : X \to [1,0]$ and $T^-, I^-, F^- : X \to [-1,0]$.

The positive membership degree $T^+(x), I^+(x), F^+(x)$ denotes the truth membershihp, indeterminate membership and false membership of an element $x \in X$ corrbership degree $T^-(x), I^-(x), F^-(x)$ denotes the truth membership, indeterminate membership and false membership of an element $x \in X$ to some implicit counter-property corresponding to a bipolar neutrosophic set A.

Here, each and every element in Bipolar Neutrosophic Set is referred to as Bipolar Neutrosophic Element.

In this section, we will recall the definitions of IVBNS and their operations. Also, we will provide the Python implementation and code for the same.

Definition 0.3. [8] *A interval valued bipolar neutrosophic set A in X is defined as an object of the form*

$$A = \langle \left[T_L^+(x), T_R^+(x)\right], \left[I_L^+(x), I_R^+(x)\right], \left[F_L^+(x), F_R^+(x)\right], \\ \left[T_L^-(x), T_R^-(x)\right], \left[I_L^-(x), I_R^-(x)\right], \left[F_L^-(x), F_R^-(x)\right]\rangle$$

where

$$T_L^+(x), T_R^+(x), I_L^+(x), I_R^+(x), F_L^+(x), F_R^+(x) : X \to [0,1]$$

and

$$T_L^-(x), T_R^-(x), I_L^-(x), I_R^-(x), F_L^-(x), F_R^-(x) : X \to [-1,0]$$

In this context, every element present in Interval Valued Bipolar Neutrosophic Set is called as Interval Valued Bipolar Neutrosophic Element.

The python code for the implementation of Interval Valued Bipolar Neutrosophic Element is provided below:

```
class IntervalValuedBipolarNeutrosophicElement:
    def __init__(self, parent,
                 truth_pos_left, truth_pos_right,
                 indeterminacy_pos_left, indeterminacy_pos_right,
                 falsity_pos_left, falsity_pos_right,
                 truth_neg_left, truth_neg_right,
                 indeterminacy_neg_left, indeterminacy_neg_right,
                 falsity_neg_left, falsity_neg_right
                 ):

        assert 0 <= truth_pos_left <= 1
        assert 0 <= truth_pos_right <= 1
        assert 0 <= indeterminacy_pos_left <= 1
        assert 0 <= indeterminacy_pos_right <= 1
        assert 0 <= falsity_pos_left <= 1
        assert 0 <= falsity_pos_right <= 1
        assert -1 <= truth_neg_left <= 0
        assert -1 <= truth_neg_right <= 0
        assert -1 <= indeterminacy_neg_left <= 0
        assert -1 <= indeterminacy_neg_right <= 0
        assert -1 <= falsity_neg_left <= 0
        assert -1 <= falsity_neg        _right <= 0

        self.parent = parent
        self.truth_pos_left = truth_pos_left
        self.truth_pos_right = truth_pos_right
        self.indeterminacy_pos_left = indeterminacy_pos_left
        self.indeterminacy_pos_right = indeterminacy_pos_right
        self.falsity_pos_left = falsity_pos_left
        self.falsity_pos_right = falsity_pos_right
        self.truth_neg_left = truth_neg_left
        self.truth_neg_right = truth_neg_right
        self.indeterminacy_neg_left = indeterminacy_neg_left
        self.indeterminacy_neg_right = indeterminacy_neg_right
        self.falsity_neg_left = falsity_neg_left
        self.falsity_neg_right = falsity_neg_right
```

The ___str___ function in Python is overrided as shown below to display the string representation of Interval Valued Bipolar Neutrosophic Element.

```
def ___str___(self):
    return (
        f"<{self.parent},"
        f"[{self.truth_pos_left}, {self.truth_pos_right}],"
        f"[{self.indeterminacy_pos_left}, {self.indeterminacy_pos_right}],"
        f"[{self.falsity_pos_left}, {self.falsity_pos_right}],"
        f"[{self.truth_neg_left}, {self.truth_neg_right}],"
        f"[{self.indeterminacy_neg_left}, {self.indeterminacy_neg_right}],"
        f"[{self.falsity_neg_left}, {self.falsity_neg_right}]>"
    )
```

Definition 0.4. [8] *Let*

$$A_1 = \left\langle \left[T_{1L}^+(x), T_{1R}^+(x)\right], \left[I_{1L}^+(x), I_{1R}^+(x)\right], \left[F_{1L}^+(x), F_{1R}^+(x)\right], \left[T_{1L}^-(x), T_{1R}^-(x)\right], \left[I_{1L}^-(x), I_{1R}^-(x)\right], \left[F_{1L}^-(x), F_{1R}^-(x)\right] \right\rangle$$

and

$$A_2 = \left\langle \left[T_{2L}^+(x), T_{2R}^+(x)\right], \left[I_{2L}^+(x), I_{2R}^+(x)\right], \left[F_{2L}^+(x), F_{2R}^+(x)\right], \left[T_{2L}^-(x), T_{2R}^-(x)\right], \left[I_{2L}^-(x), I_{2R}^-(x)\right], \left[F_{2L}^-(x), F_{2R}^-(x)\right] \right\rangle$$

be two interval valued bipolar neutrosophic elements.

For example, let us consider two Interval Valued Bipolar Neutrosophic Elements A_1 and A_2.

$$A_1 = \langle [0.5, 0.6], [0.2, 0.5], [0.1, 0.7], [-0.2, -0.1], [-0.6, -0.2], [-0.4, -0.3] \rangle$$

and

$$A_2 = \langle [0.3, 0.4], [0.1, 0.6], [0.5, 0.7], [-0.5, -0.1], [-0.8, -0.7], [-0.9, -0.8] \rangle$$

[8]1. Then the complement of A_1 is denoted by A_1^c and is defined by

$$T_{1L^C}^+(x) = \{1^+\} - T_{1R}^+(x) \quad I_{1L^C}^+(x) = \{1^+\} - I_{1R}^+(x) \quad F_{1L^C}^+(x) = \{1^+\} - F_{1R}^+(x)$$

$$T_{1R^C}^+(x) = \{1^+\} - T_{1L}^+(x) \quad I_{1R^C}^+(x) = \{1^+\} - I_{1L}^+(x) \quad F_{1R^C}^+(x) = \{1^+\} - F_{1L}^+(x)$$

$$T_{1L^C}^-(x) = \{1^-\} - T_{1R}^-(x) \quad I_{1L^C}^-(x) = \{1^-\} - I_{1R}^-(x) \quad F_{1L^C}^-(x) = \{1^-\} - F_{1R}^-(x)$$

$$T_{1R^C}^-(x) = \{1^-\} - T_{1L}^-(x) \quad I_{1R^C}^-(x) = \{1^-\} - I_{1L}^-(x) \quad F_{1R^C}^-(x) = \{1^-\} - F_{1L}^-(x)$$

for all $x \in X$.

The following is the code that computes the complement of Interval Valued Bipolar Neutrosophic Element.

```
def complement(self):
    return IntervalValuedBipolarNeutrosophicElement(
        self.parent,
        1 - self.truth_pos_right,
        1 - self.truth_pos_left,
```

```
            1 - self.indeterminacy_pos_right,
            1 - self.indeterminacy_pos_left,
            1 - self.falsity_pos_right,
            1 - self.falsity_pos_left,
            -1 - self.truth_neg_right,
            -1 - self.truth_neg_left,
            -1 - self.indeterminacy_neg_right,
            -1 - self.indeterminacy_neg_left,
            -1 - self.falsity_neg_right,
            -1 - self.falsity_neg_left,
        )
```

For our example, the complement of A_1, denoted by A_1^c is:

$$A_1^c = \langle [0.4, 0.9], [0.5, 0.9], [0.6, 0.9], \\ [-0.8, -0.5], [-0.7, -0.3], [-0.8, -0.6] \rangle$$

[8]2. Two Interval Valued Bipolar Neutrosophic Elements are said to be equal $A_1 = A_2$ if and only if

$T^+_{1L}(x) = T^+_{2L}(x) \quad T^+_{1R}(x) = T^+_{2R}(x) \quad I^+_{1L}(x) = I^+_{2L}(x) \quad I^+_{1R}(x) = I^+_{2R}(x)$

$F^+_{1L}(x) = F^+_{2L}(x) \quad F^+_{1R}(x) = F^+_{2R}(x) \quad T^-_{1L}(x) = T^-_{2L}(x) \quad T^-_{1R}(x) = T^-_{2R}(x)$

$I^-_{1L}(x) = I^-_{2L}(x) \quad I^-_{1R}(x) = I^-_{2R}(x) \quad F^-_{1L}(x) = F^-_{2L}(x) \quad F^-_{1R}(x) = F^-_{2R}(x)$

for all $x \in X$.

The following is the code that checks the equality of two IVBN elements. This is done by overriding the __eq__ function in Python.

```
    def __eq__(self, other):
        if (
            self.parent == other.parent
            and self.truth_pos_left == other.truth_pos_left
            and self.truth_pos_right == other.truth_pos_right
            and self.indeterminacy_pos_left == other.indeterminacy_pos_left
            and self.indeterminacy_pos_right == other.indeterminacy_pos_right
            and self.falsity_pos_left == other.falsity_pos_left
            and self.falsity_pos_right == other.falsity_pos_right
            and self.truth_neg_left == other.truth_neg_left
            and self.truth_neg_right == other.truth_neg_right
            and self.indeterminacy_neg_left == other.indeterminacy_neg_left
            and self.indeterminacy_neg_right == other.indeterminacy_neg_right
            and self.falsity_neg_left == other.falsity_neg_left
            and self.falsity_neg_right == other.falsity_neg_right
        ):
            return True

        return False
```

For our example, the two IVBNS elements A_1 and A_2 do not fulfill the requirements for equality, so they are not equal. $(A_1 \neq A_2)$

[8]3. A IVBN element A_1 is said to be a subset of another IVBN element A_2 if and only if

$$T_{1L}^{+}(x) \leq T_{2L}^{+}(x) \quad T_{1R}^{+}(x) \leq T_{2R}^{+}(x) \quad I_{1L}^{+}(x) \geq I_{2L}^{+}(x) \quad I_{1R}^{+}(x) \geq I_{2R}^{+}(x)$$

$$F_{1L}^{+}(x) \geq F_{2L}^{+}(x) \quad F_{1R}^{+}(x) \geq F_{2R}^{+}(x) \quad T_{1L}^{-}(x) \leq T_{2L}^{-}(x) \quad T_{1R}^{-}(x) \leq T_{2R}^{-}(x)$$

$$I_{1L}^{-}(x) \geq I_{2L}^{-}(x) \quad I_{1R}^{-}(x) \geq I_{2R}^{-}(x) \quad F_{1L}^{-}(x) \geq F_{2L}^{-}(x) \quad F_{1R}^{-}(x) \geq F_{2R}^{-}(x)$$

for all $x \in X$.

The following code checks whether an IVBN element is a subset of the other.

```
def is_subset(self, other):
    if (
        self.parent == other.parent
        and self.truth_pos_left <= other.truth_pos_left
        and self.truth_pos_right <= other.truth_pos_right
        and self.indeterminacy_pos_left >= other.indeterminacy_pos_left
        and self.indeterminacy_pos_right >= other.indeterminacy_pos_right
        and self.falsity_pos_left >= other.falsity_pos_left
        and self.falsity_pos_right >= other.falsity_pos_right
        and self.truth_neg_left <= other.truth_neg_left
        and self.truth_neg_right <= other.truth_neg_right
        and self.indeterminacy_neg_left >= other.indeterminacy_neg_left
        and self.indeterminacy_neg_right >= other.indeterminacy_neg_right
        and self.falsity_neg_left >= other.falsity_neg_left
        and self.falsity_neg_right >= other.falsity_neg_right
    ):
        return True

    return False
```

For our example, the two IVBNS elements A_1 and A_2 do not fulfill the requirements for subset property, so A_1 is not a subset of A_2 ($A_1 \nsubseteq A_2$)

[8]4. The union of two IVBN elements A_1 and A_2 is defined as

$$\begin{gathered}
A_1 \cup A_2(x) = \\
\left[\max\left\{T_{1L}^{+}(x), T_{2L}^{+}(x)\right\}, \max\left\{T_{1R}^{+}(x), T_{2R}^{+}(x)\right\}\right], \\
\left[\min\left\{I_{1L}^{+}(x), I_{2L}^{+}(x)\right\}, \min\left\{I_{1R}^{+}(x), I_{2R}^{+}(x)\right\}\right], \\
\left[\min\left\{F_{1L}^{+}(x), F_{2L}^{+}(x)\right\}, \min\left\{F_{1R}^{+}(x), F_{2R}^{+}(x)\right\}\right], \\
\left[\min\left\{T_{1L}^{-}(x), T_{2L}^{-}(x)\right\}, \min\left\{T_{1R}^{-}(x), T_{2R}^{-}(x)\right\}\right], \\
\left[\max\left\{I_{1L}^{-}(x), I_{2L}^{-}(x)\right\}, \max\left\{I_{1R}^{-}(x), I_{2R}^{-}(x)\right\}\right], \\
\left[\max\left\{F_{1L}^{-}(x), F_{2L}^{-}(x)\right\}, \max\left\{F_{1R}^{-}(x), F_{2R}^{-}(x)\right\}\right]
\end{gathered}$$

for all $x \in X$.

The following code finds the union on two IVBN elements A_1 and A_2

```
def union(self, other):
    return IntervalValuedBipolarNeutrosophicElement(
        self.parent,
        max(self.truth_pos_left, other.truth_pos_left),
```

```
        max(self.truth_pos_right, other.truth_pos_right),
        min(self.indeterminacy_pos_left, other.indeterminacy_pos_left),
        min(self.indeterminacy_pos_right, other.indeterminacy_pos_right),
        min(self.falsity_pos_left, other.falsity_pos_left),
        min(self.falsity_pos_right, other.falsity_pos_right),
        min(self.truth_neg_left, other.truth_neg_left),
        min(self.truth_neg_right, other.truth_neg_right),
        max(self.indeterminacy_neg_left, other.indeterminacy_neg_left),
        max(self.indeterminacy_neg_right, other.indeterminacy_neg_right),
        max(self.falsity_neg_left, other.falsity_neg_left),
        max(self.falsity_neg_right, other.falsity_neg_right),
    )
```

For our example, the union of the two IVBNS elements A_1 and A_2, denoted by $A_1 \cup A_2$ is:

$$\begin{aligned} A_1 \cup A_2 = \langle & [0.3, 0.6], [0.1, 0.5], [0.1, 0.4], \\ & [-0.5, -0.2], [-0.7, -0.3], [-0.4, -0.2] \rangle \end{aligned}$$

[8]5. The intersection of two IVBN elements A_1 and A_2 is defined as

$$\begin{gathered} A_1 \cap A_2(x) = \\ \left[\min\left\{T^+_{1L}(x), T^+_{2L}(x)\right\}, \min\left\{T^+_{1R}(x), T^+_{2R}(x)\right\}\right], \\ \left[\max\left\{I^+_{1L}(x), I^+_{2L}(x)\right\}, \max\left\{I^+_{1R}(x), I^+_{2R}(x)\right\}\right], \\ \left[\max\left\{F^+_{1L}(x), F^+_{2L}(x)\right\}, \max\left\{F^+_{1R}(x), F^+_{2R}(x)\right\}\right], \\ \left[\max\left\{T^-_{1L}(x), T^-_{2L}(x)\right\}, \max\left\{T^-_{1R}(x), T^-_{2R}(x)\right\}\right], \\ \left[\min\left\{I^-_{1L}(x), I^-_{2L}(x)\right\}, \min\left\{I^-_{1R}(x), I^-_{2R}(x)\right\}\right], \\ \left[\min\left\{F^-_{1L}(x), F^-_{2L}(x)\right\}, \min\left\{F^-_{1R}(x), F^-_{2R}(x)\right\}\right] \end{gathered}$$

for all $x \in X$.

The following code finds the intersection between two IVBN elements A_1 and A_2

```
def intersection(self, other):
    return IntervalValuedBipolarNeutrosophicElement(
        self.parent,
        min(self.truth_pos_left, other.truth_pos_left),
        min(self.truth_pos_right, other.truth_pos_right),
        max(self.indeterminacy_pos_left, other.indeterminacy_pos_left),
        max(self.indeterminacy_pos_right, other.indeterminacy_pos_right),
        max(self.falsity_pos_left, other.falsity_pos_left),
        max(self.falsity_pos_right, other.falsity_pos_right),
        max(self.truth_neg_left, other.truth_neg_left),
        max(self.truth_neg_right, other.truth_neg_right),
        min(self.indeterminacy_neg_left, other.indeterminacy_neg_left),
        min(self.indeterminacy_neg_right, other.indeterminacy_neg_right),
        min(self.falsity_neg_left, other.falsity_neg_left),
        min(self.falsity_neg_right, other.falsity_neg_right),
    )
```

For our example, the intersection of the two IVBNS elements A_1 and A_2, denoted by $A_1 \cap A_2$ is:

$$A_1 \cap A_2 = \langle [0.1, 0.4], [0.1, 0.6], [0.5, 0.7], \\ [-0.5, -0.1], [-0.8, -0.7], [-0.9, -0.8] \rangle$$

[8]6. The sum of two IVBN elements A_1 and A_2 is defined as

$$A_1 + A_2 = \\ \left\langle \left[T^+_{1L} + T^+_{2L} - T^+_{1L}.T^+_{2L}, T^+_{1R} + T^+_{2R} - T^+_{1R}.T^+_{2R}\right], \left[I^+_{1L}I^+_{2L}, I^+_{1R}I^+_{2R}\right], \\ \left[F^+_{1L}F^+_{2L}, F^+_{1R}F^+_{2R}\right], \left[-T^-_{1L}.T^-_{2L}, -T^-_{1R}T^-_{2R}\right], \\ \left[-\left(-I^-_{1L} - I^-_{2L} - I^-_{1L}.I^-_{2L}\right), -\left(-I^-_{1R} - I^-_{2R} - I^-_{1R}.I^-_{2R}\right)\right], \\ \left[-\left(-F^-_{1L} - F^-_{2L} - F^-_{1L}.F^-_{2L}\right), -\left(-F^-_{1R} - F^-_{2R} - F^-_{1R}.F^-_{2R}\right)\right]\right\rangle$$

The following code finds the sum of two IVBN elements A_1 and A_2. This is done by overriding __add__ function in Python.

```
def __add__(self, other):
    return IntervalValuedBipolarNeutrosophicElement(
        self.parent,
        self.truth_pos_left
        + other.truth_pos_left
        - (self.truth_pos_left * other.truth_pos_left),
        self.truth_pos_right
        + other.truth_pos_right
        - (self.truth_pos_right * other.truth_pos_right),
        self.indeterminacy_pos_left * other.indeterminacy_pos_left,
        self.indeterminacy_pos_right * other.indeterminacy_pos_right,
        self.falsity_pos_left * other.falsity_pos_left,
        self.falsity_pos_right * other.falsity_pos_right,
        -1 * self.truth_neg_left * other.truth_neg_left,
        -1 * self.truth_neg_right * other.truth_neg_right,
        -1
        * (
            (-1 * self.indeterminacy_neg_left)
            - other.indeterminacy_neg_left
            - (self.indeterminacy_neg_left * other.indeterminacy_neg_left)
        ),
        -1
        * (
            (-1 * self.indeterminacy_neg_right)
            - other.indeterminacy_neg_right
            - (self.indeterminacy_neg_right * other.indeterminacy_neg_right)
        ),
        -1
        * (
            (-1 * self.falsity_neg_left)
            - other.falsity_neg_left
            - (self.falsity_neg_left * other.falsity_neg_left)
        ),
        -1
```

```
        * (
            (-1 * self.falsity_neg_right)
            - other.falsity_neg_right
            - (self.falsity_neg_right * other.falsity_neg_right)
        ),
    )
```

For our example, the sum of the two IVBNS elements A_1 and A_2, denoted by $A_1 + A_2$ is:

$$A_1 + A_2 = \langle [0.37, 0.76], [0.01, 0.3], [0.05, 0.28], \\ [-0.25, -0.02], [-0.94, -0.79], [-0.94, -0.84] \rangle$$

[8]7. The product of two IVBN elements A_1 and A_2 is defined as

$$A_1.A_2 =$$

$$\langle [T_{1L}^+ T_{2L}^+, T_{1R}^+ T_{2R}^+], [I_{1L}^+ + I_{2L}^+ - I_{1L}^+ I_{2L}^+, I_{1R}^+ + I_{2R}^+ - I_{1R}^+ I_{2R}^+], \\ [F_{1L}^+ + F_{2L}^+ - F_{1L}^+ F_{2L}^+, F_{1R}^+ + F_{2R}^+ - F_{1R}^+ F_{2R}^+], \\ [-(-T_{1L}^- - T_{2L}^- - T_{1L}^- T_{2L}^-), -(-T_{1R}^- - T_{2R}^- - T_{1R}^- T_{2R}^-)], \\ [-(I_{1L}^- I_{2L}^-), -(I_{1R}^- I_{2R}^-)], \ [-(F_{1L}^- F_{2L}^-), -(F_{1R}^- F_{2R}^-)] \rangle$$

The following code finds the product of two IVBN elements A_1 and A_2. This is done by overriding ___mul___ function in Python.

```
def __mul__(self, other):
    return IntervalValuedBipolarNeutrosophicElement(
        self.parent,
        self.truth_pos_left * other.truth_pos_left,
        self.truth_pos_right * other.truth_pos_right,
        self.indeterminacy_pos_left
        + other.indeterminacy_pos_left
        - (self.indeterminacy_pos_left * other.indeterminacy_pos_left),
        self.indeterminacy_pos_right
        + other.indeterminacy_pos_right
        - (self.indeterminacy_pos_right * other.indeterminacy_pos_right),
        self.falsity_pos_left
        + other.falsity_pos_left
        - (self.falsity_pos_left * other.falsity_pos_left),
        self.falsity_pos_right
        + other.falsity_pos_right
        - (self.falsity_pos_right * other.falsity_pos_right),
        -1
        * (
            (-1 * self.truth_neg_left)
            - other.truth_neg_left
            - (self.truth_neg_left * other.truth_neg_left)
        ),
        -1
        * (
            (-1 * self.truth_neg_right)
            - other.truth_neg_right
            - (self.truth_neg_right * other.truth_neg_right)
        ),
```

```
            -1 * self.indeterminacy_neg_left * other.indeterminacy_neg_left,
            -1 * self.indeterminacy_neg_right * other.indeterminacy_neg_right,
            -1 * self.falsity_neg_left * other.falsity_neg_left,
            -1 * self.falsity_neg_right * other.falsity_neg_right,
        )
```

For our example, the product of the two IVBNS elements A_1 and A_2, denoted by $A_1.A_2$ is:

$$A_1.A_2 = \langle [0.03, 0.24], [0.19, 0.8], [0.55, 0.82], \\ [-0.75, -0.28], [-0.56, -0.21], [-0.36, -0.16] \rangle$$

[8]8. The product of an IVBN element A_1 and an integer scalar λ, $\lambda > 0$ is defined as

$$\lambda A_1 = \\ \left\langle \left[1-\left(1-T_L^+\right)^\lambda, 1-\left(1-T_R^+\right)^\lambda\right], \left[\left(I_L^+\right)^\lambda, \left(I_R^+\right)^\lambda\right], \left[\left(F_L^+\right)^\lambda, \left(F_R^+\right)^\lambda\right], \right. \\ \left[-\left(-T_L^-\right)^\lambda, -\left(-T_R^-\right)^\lambda\right], \left[-\left(-I_L^-\right)^\lambda, -\left(-I_R^-\right)^\lambda\right], \\ \left. \left[-\left(1-\left(1-\left(-F_L^-\right)\right)^\lambda\right), -\left(1-\left(1-\left(-F_R^-\right)\right)^\lambda\right)\right] \right\rangle$$

The following code finds the product of an IVBN element A_1 and an integer λ. This is done by overriding __mul__ function in Python.

```
def __mul__(self, other):
    return IntervalValuedBipolarNeutrosophicElement(
            self.parent,
            1 - pow(1 - self.truth_pos_left, other),
            1 - pow(1 - self.truth_pos_right, other),
            pow(self.indeterminacy_pos_left, other),
            pow(self.indeterminacy_pos_right, other),
            pow(self.falsity_pos_left, other),
            pow(self.falsity_pos_right, other),
            -1 * pow(-1 * self.truth_neg_left, other),
            -1 * pow(-1 * self.truth_neg_right, other),
            -1 * pow(-1 * self.indeterminacy_neg_left, other),
            -1 * pow(-1 * self.indeterminacy_neg_right, other),
            -1 * (1 - pow(1 - (-1 * self.falsity_neg_left), other)),
            -1 * (1 - pow(1 - (-1 * self.falsity_neg_right), other)),
        )
```

For our example, let us consider the scalar λ as 3. The product of a scalar and a IVBNS, denoted by $3A_1$ is:

$$3A_1 = \langle [0.271, 0.936], [0.001, 0.125], [0.001, 0.064], \\ [-0.125, -0.008], [-0.343, -0.027], [-0.784, -0.488] \rangle$$

Now, combining both the code of the same function so that it performs multiplication operation accordingly, based on the type of the multiplier.

```
def __mul__(self, other):
    if isinstance(other, int):
        return IntervalValuedBipolarNeutrosophicElement(
            self.parent,
            1 - pow(1 - self.truth_pos_left, other),
```

```
                1 - pow(1 - self.truth_pos_right, other),
                pow(self.indeterminacy_pos_left, other),
                pow(self.indeterminacy_pos_right, other),
                pow(self.falsity_pos_left, other),
                pow(self.falsity_pos_right, other),
                -1 * pow(-1 * self.truth_neg_left, other),
                -1 * pow(-1 * self.truth_neg_right, other),
                -1 * pow(-1 * self.indeterminacy_neg_left, other),
                -1 * pow(-1 * self.indeterminacy_neg_right, other),
                -1 * (1 - pow(1 - (-1 * self.falsity_neg_left), other)),
                -1 * (1 - pow(1 - (-1 * self.falsity_neg_right), other)),
            )
        else:
            return IntervalValuedBipolarNeutrosophicElement(
                self.parent,
                self.truth_pos_left * other.truth_pos_left,
                self.truth_pos_right * other.truth_pos_right,
                self.indeterminacy_pos_left
                + other.indeterminacy_pos_left
                - (self.indeterminacy_pos_left * other.indeterminacy_pos_left),
                self.indeterminacy_pos_right
                + other.indeterminacy_pos_right
                - (self.indeterminacy_pos_right * other.indeterminacy_pos_right),
                self.falsity_pos_left
                + other.falsity_pos_left
                - (self.falsity_pos_left * other.falsity_pos_left),
                self.falsity_pos_right
                + other.falsity_pos_right
                - (self.falsity_pos_right * other.falsity_pos_right),
                -1
                * (
                    (-1 * self.truth_neg_left)
                    - other.truth_neg_left
                    - (self.truth_neg_left * other.truth_neg_left)
                ),
                -1
                * (
                    (-1 * self.truth_neg_right)
                    - other.truth_neg_right
                    - (self.truth_neg_right * other.truth_neg_right)
                ),
                -1 * self.indeterminacy_neg_left * other.indeterminacy_neg_left,
                -1 * self.indeterminacy_neg_right * other.indeterminacy_neg_right,
                -1 * self.falsity_neg_left * other.falsity_neg_left,
                -1 * self.falsity_neg_right * other.falsity_neg_right,
            )
```

[8]9. The score function $S(A_1)$ is defined as

$$S(A_1) =$$

$$\frac{1}{12}\left(T_L^+ + T_R^+ + 1 - I_L^+ + 1 - I_R^+ + 1 - F_L^+ + 1\right.$$

$$\left. - F_R^+ + 1 + T_L^- + 1 + T_R^- - I_L^- - I_R^- - F_L^- - F_R^-\right)$$

The following code finds the score function of the IVBN element A_1

```
def score(self):
    return (1 / 12) * (
        self.truth_pos_left
        + self.truth_pos_right
        + 1
        - self.indeterminacy_pos_left
        + 1
        - self.indeterminacy_pos_right
        + 1
        - self.falsity_pos_left
        + 1
        - self.falsity_pos_right
        + 1
        + self.truth_neg_left
        + 1
        + self.truth_neg_right
        - self.indeterminacy_neg_left
        - self.indeterminacy_neg_right
        - self.falsity_neg_left
        - self.falsity_neg_right
    )
```

For our example, the score value of A_1, denoted by $S(A_1)$ is: 0.5417

[8]10. The accuracy function $A(A_1)$ is defined as

$$A(A_1) = T_L^+ + T_R^+ F_L^+ F_R^+ + T_L^- + T_R^- F_L^- F_R^-$$

The following code finds the accuracy function of the IVBN element A_1

```
def accuracy(self):
    return (
        self.truth_pos_left
        + self.truth_pos_right
        - self.falsity_pos_left
        - self.falsity_pos_right
        + self.truth_neg_left
        + self.truth_neg_right
        - self.falsity_neg_left
        - self.falsity_neg_right
    )
```

For our example, the accuracy value of A_1, denoted by $A(A_1)$ is: 0.1

[8]11. The certainity function $C(A_1)$ is defined as

$$C(A_1) = T_L^+ + T_R^+ F_L^- F_R^-$$

The following code finds the certiainity function of the IVBN element A_1

```
    def certainity(self):
        return (
            self.truth_pos_left
            + self.truth_pos_right
            - self.falsity_neg_leftx
            - self.falsity_neg_right
        )
```

For our example, the certainity value of A_1, denoted by $C(A_1)$ is: 1.3

Now, the Python code for Interval Valued Bipolar Neutrosophic Sets, which is a collection of several Interval Valued Bipolar Neutrosophic Elements is given below:

```
class IntervalValuedBipolarNeutrosophicSet():
    def __init__(self):
        self.items = []
        self.len = 0

    def add_element(self, elt):
        if not isinstance(elt, IntervalValuedBipolarNeutrosophicElement):
            raise TypeError("Please provide the element as Interval Valued Bipolar Neutroso
        else:
            self.items.append(elt)
            self.len += 1

    def complement(self):
        complement_set = IntervalValuedBipolarNeutrosophicSet()
        for elt in self.items:
            complement_set.add_element(elt.complement())
        return complement_set

    def __eq__(self, other):

        if self.len != other.len:
            return False

        flag = False
        for i in range(self.len):
            if self.items[i] == other.items[i]:
                flag = True

            if not flag:
                return False

        return True

    def is_subset(self, other):

        if self.len != other.len:
            return False
```

```
        flag = False
        for i in range(self.len):
            if self.items[i].is_subset(other.items[i]):
                flag = True

            if not flag:
                return False

        return True

    def union(self, other):

        if self.len != other.len:
            raise TypeError("Sets are not of the same size!")

        union_set = IntervalValuedBipolarNeutrosophicSet()

        for i in range(self.len):
            union_set.add_element(self.items[i].union(other.items[i]))

        return union_set

    def intersection(self, other):
        if self.len != other.len:
            raise TypeError("Sets are not of the same size!")

        intersection_set = IntervalValuedBipolarNeutrosophicSet()

        for i in range(self.len):
            intersection_set.add_element(self.items[i].intersection(other.items[i]))

        return intersection_set

    def __add__(self, other):
        if self.len != other.len:
            raise TypeError("Sets are not of the same size!")

        sum_set = IntervalValuedBipolarNeutrosophicSet()

        for i in range(self.len):
            sum_set.add_element(self.items[i] + other.items[i])

        return sum_set

    def __mul__(self, other):
        product_set = IntervalValuedBipolarNeutrosophicSet()
        if isinstance(other, int):
            for i in range(self.len):
                product_set.add_element(other * self.items[i])
```

```
        elif isinstance(other, IntervalValuedBipolarNeutrosophicSet):
            for i in range(self.len):
                product_set.add_element(self.items[i] * other.items[i])

        return product_set

    def score(self):
        score_result = []

        for elt in self.items:
            score_result.append(elt.score())

        return score_result

    def accuracy(self):
        accuracy_result = []

        for elt in self.items:
            accuracy_result.append(elt.accuracy())

        return accuracy_result

    def certainity(self):
        certainity_result = []

        for elt in self.items:
            certainity_result.append(elt.certainity())

        return certainity_result

    def __str__(self):
        string_to_be_returned = "{\n"
        for i in range(self.len):
            if i+1 >= self.len:
                string_to_be_returned += "\t" + str(self.items[i]) + "\n"
            else:
                string_to_be_returned += "\t" + str(self.items[i]) + ",\n"

        string_to_be_returned += "}"

        return string_to_be_returned
```

3|Python Tool for IVBNS

On exploring the comparative landscape of toolboxes dedicated to Interval Valued Bipolar Neutrosophic Sets (IVBNS) analysis and processing, the only framework available in the market is the Matlab platform. The absence of a wider range of comparable tools underscores the novelty and potential significance of the Python toolbox developed in this study. In terms of cost and accessibility, Matlab is proprietary with license costs whereas the Python toolbox built is open-source and it allows the users to add and modify the source at free cost. Secondly concerning size and resource utilization Python toolbox runs effectively even on systems with limited computational resources and the proposed IVBNS framework involves no specific modules being imported and the implementation is written in simple methods hence the storage space is less than 17KB whereas MATLAB is more resource-intensive, potentially impacting performance on lower-end machines. Thirdly the proposed toolbox offers a user-friendly GUI, simplifying interaction and reducing the learning curve for users unfamiliar with programming languages. In contrast, the existing MATLAB toolbox relies on a legacy interface, tailored for users with programming expertise to navigate and utilize its functionalities effectively. A snapshot of the User Interface design of the proposed work is depicted in Figure 1 below.

In technical terms of unique features, the existing MATLAB toolbox[20] offers a valuable set of IVBNS operations, whereas the Python toolbox expands upon this functionality by including additional and score operations for usage in real-time applications.

The Python code for the IVBNS Toolbox, encompassing all functionalities and scripts necessary to run the toolbox, is publicly available on GitHub for reproducibility and transparency. The code repository can be accessed at https://github.com/praneshkumar-12/IVBNS-Toolbox

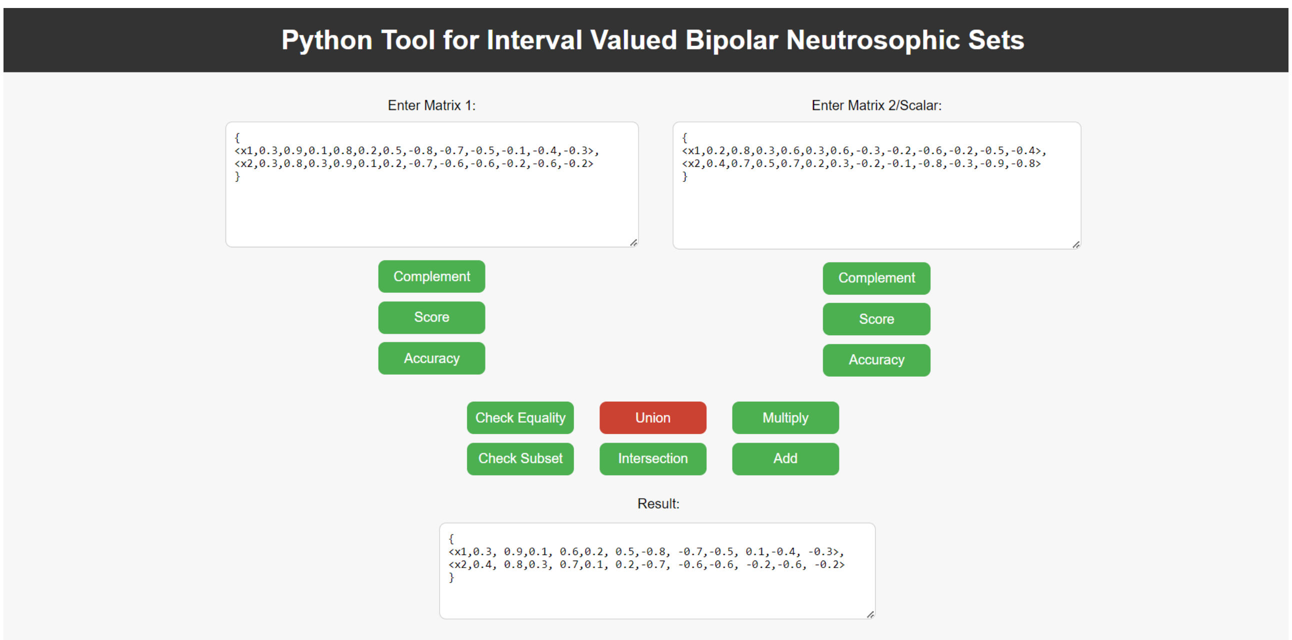

FIGURE 1. Example illustrating the IVBNS Toolbox Interface and its functionality

Design Choices:

Front-End Implementation: Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) were employed for the front end due to their ubiquity and ease of development. This selection facilitates the creation of a user-friendly interface and enables straightforward future modifications to the visual design.

Back-End Development: Python is chosen for the back-end due to its readability, extensive libraries, and expressive syntax. These characteristics promote the development of concise and maintainable code, facilitating the implementation of complex data structures and operations required for IVBNS manipulation.

Object-Oriented Design: An Abstract Data Type (ADT) encapsulating the representation and operations of IVBNS sets was implemented. This approach promotes data integrity, and reusability of the core logic, and enforces constraints specific to IVBNS, ensuring accurate and reliable computations.

Client-Server Architecture: The application adheres to a client-server architecture, separating the presentation layer (front-end) from the business logic layer (back-end). This promotes modularity, and scalability, and simplifies maintenance of the application.

Data Structures:

The abstract data type (ADT) employed for representing Interval Valued Bipolar Neutrosophic Sets (IVBNS) utilizes lists and arrays to store each set's values and requisite attributes efficiently. These data structures facilitate systematic organization and manipulation of IVBNS elements, enabling the seamless execution of operations such as addition and subtraction. Furthermore, the ADT leverages dictionaries or key-value pairs to store associated attributes such as membership functions, truth values, and indeterminacy values for individual IVBNS elements. This structured approach ensures the preservation of IVBNS properties and facilitates precise computations within the system.

Architecture Diagram:

This section presents a high-level overview of the system architecture employed in our work. We utilize a toolbox analogy, where each component within the architecture represents a specific process with a designated function. By understanding the architecture, you gain insight into how these tools work together to achieve the overall functionality of this toolbox. The architecture diagram is depicted in Figure 2.

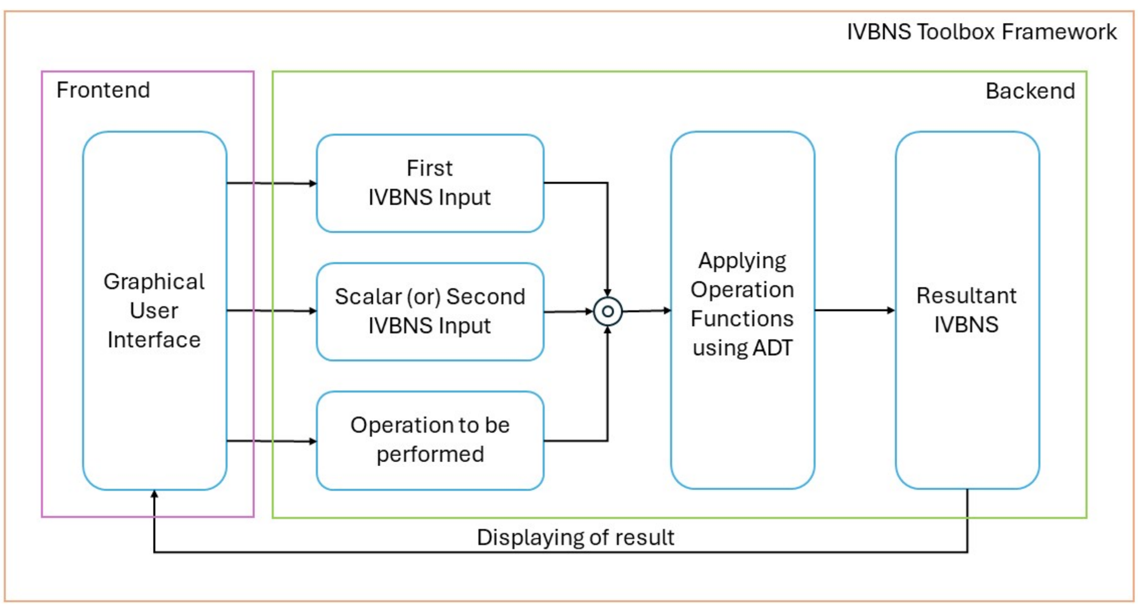

Figure 2. Architecture diagram of the IVBNS Toolbox

Technical Comparison with Existing Frameworks:

In technical aspects the existing MATLAB toolbox[20] offers a valuable set of IVBNS operations, whereas the Python toolbox expands upon this functionality. The proposed work introduces additional operations not currently available in the MATLAB toolbox. A comprehensive evaluation of the operations included in the

MATLAB toolbox is planned for future work. This comparative analysis will provide valuable insights into the strengths and potential limitations of both toolboxes.

By offering a free, user-friendly interface, efficient resource utilization, and an extended set of operations, the Python toolbox positions itself as a valuable alternative to existing MATLAB-based solutions for IVBNS manipulation.

A thorough evaluation of the software has been done. All the inputs have been tested as well as compared with manual calculations. Various test cases have been tested, including various formats of IVBNS, numbers out of range, given input is not in the form of IVBNS and so on. All the test cases have been passed and is reliable and robust. Sample test cases have been illustrated from figures Figure 3 to Figure 10.

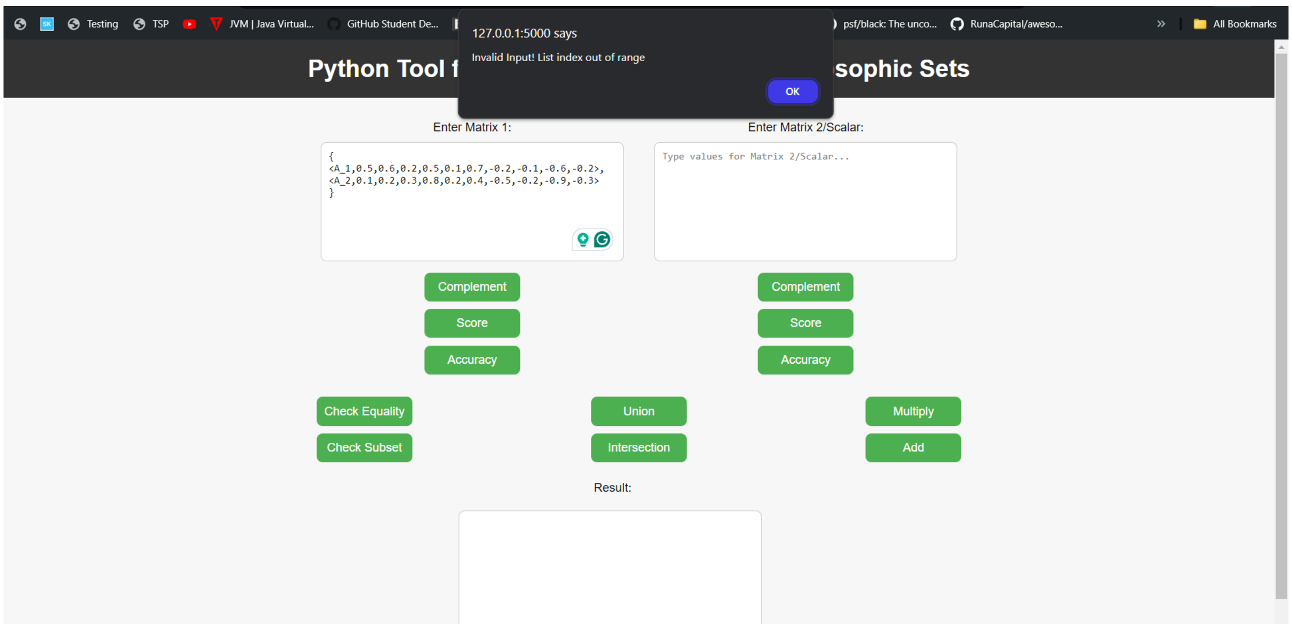

FIGURE 3. A Sample Test Case for Invalid Input

4|Numerical Result

To illustrate the system, we use set C_1 and set C_2 as the following. They contain random IVBNS numbers. The results have been tested and compared with manual solutions. We make a detailed tracing to make sure the accuracy of the results.

$$C_1 = \left\{ \begin{array}{l} \langle A_1, [0.5, 0.6], [0.2, 0.5], [0.1, 0.7], \ [-0.2, -0.1], [-0.6, -0.2], [-0.4, -0.3] \rangle, \\ \langle A_2, [0.1, 0.2], [0.3, 0.8], [0.2, 0.4], [-0.5, -0.2], [-0.9, -0.3], [-0.6, -0.1] \rangle \end{array} \right\}$$

$$C_2 = \left\{ \begin{array}{l} \langle A_1, [0.3, 0.9], [0.1, 0.8], [0.2, 0.5], \ [-0.8, -0.7], [-0.5, -0.1], [-0.4, -0.1] \rangle, \\ \langle A_2, [0.2, 0.8], [0.1, 0.4], [0.3, 0.4], [-0.5, -0.1], [-0.9, -0.4], [-0.3, -0.1] \rangle \end{array} \right\}$$

The complement of $C_1 = (C_1)^c$ is calculated as:

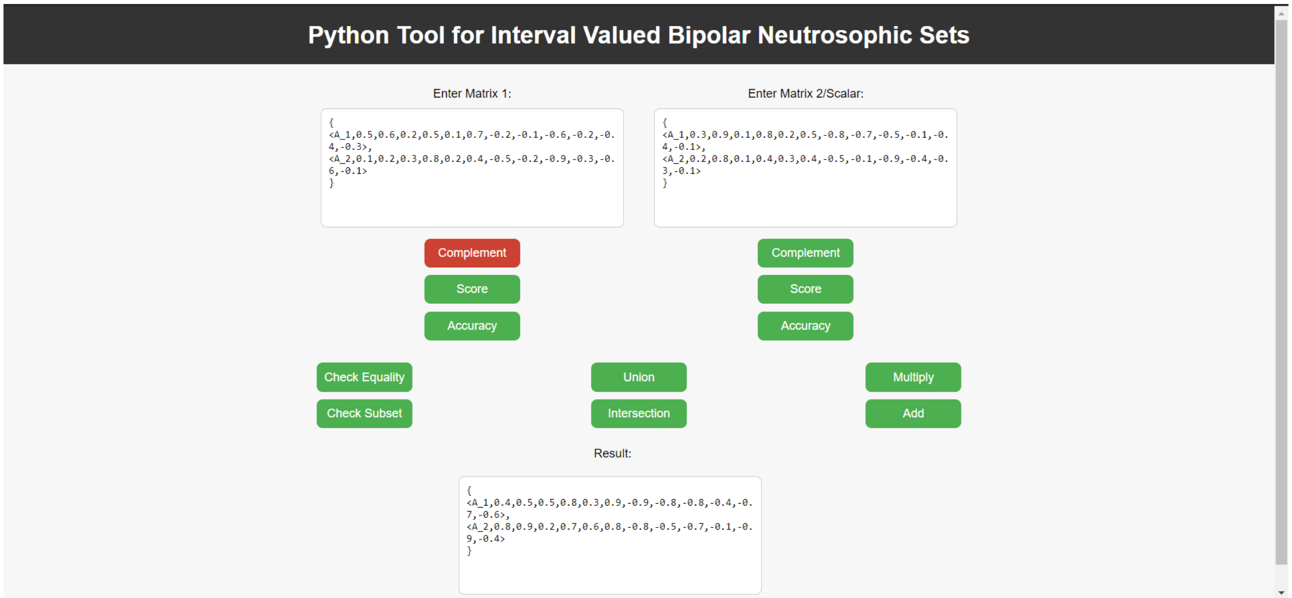

FIGURE 4. Sample Test Case for Complement

The score of C_1 is calculated as:

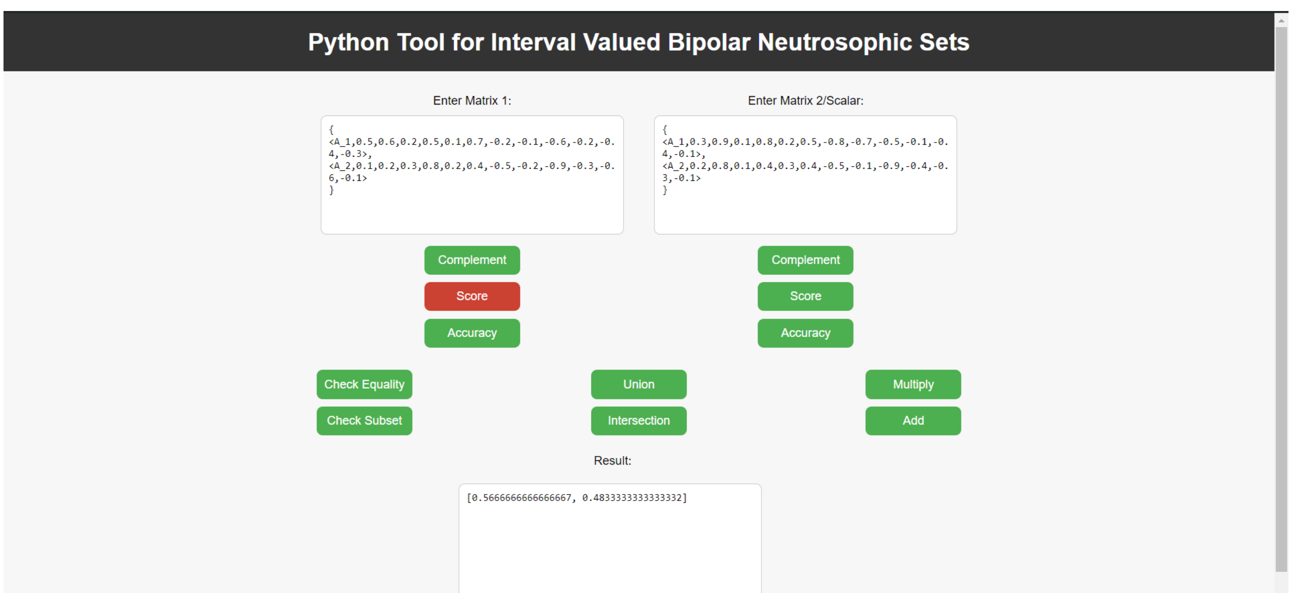

FIGURE 5. Sample Test Case for Score

The accuracy of C_1 is calculated as:

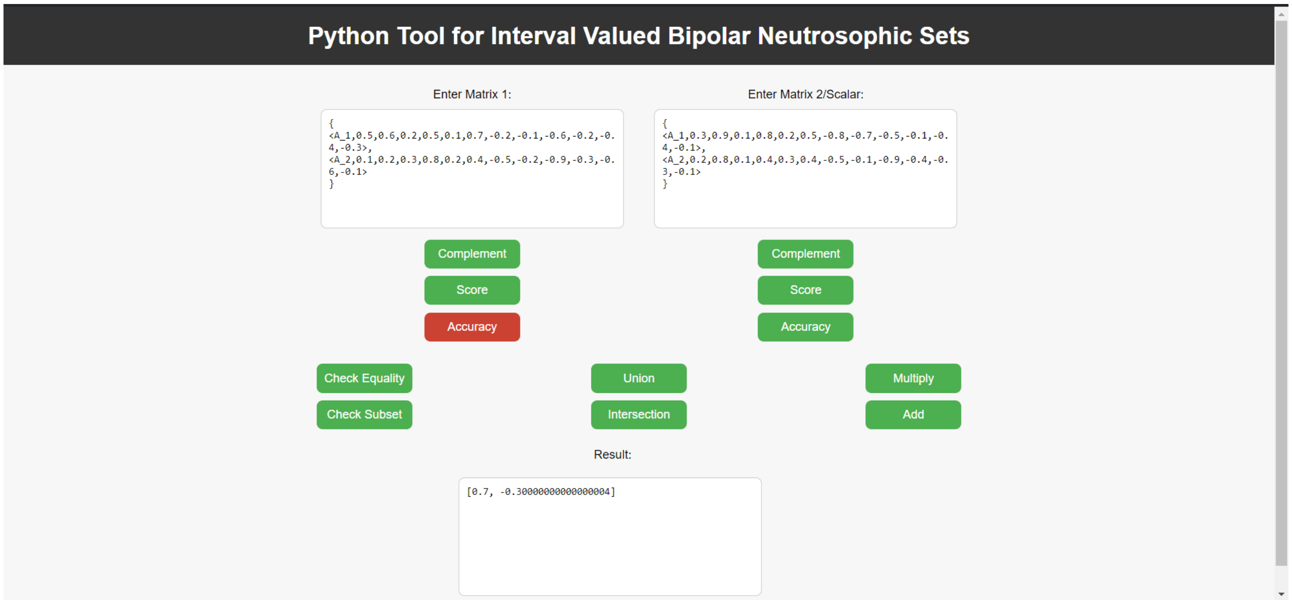

FIGURE 6. Sample Test Case for Accuracy

The union of C_1 and C_2, $C_1 \cup C_2$ is calculated as:

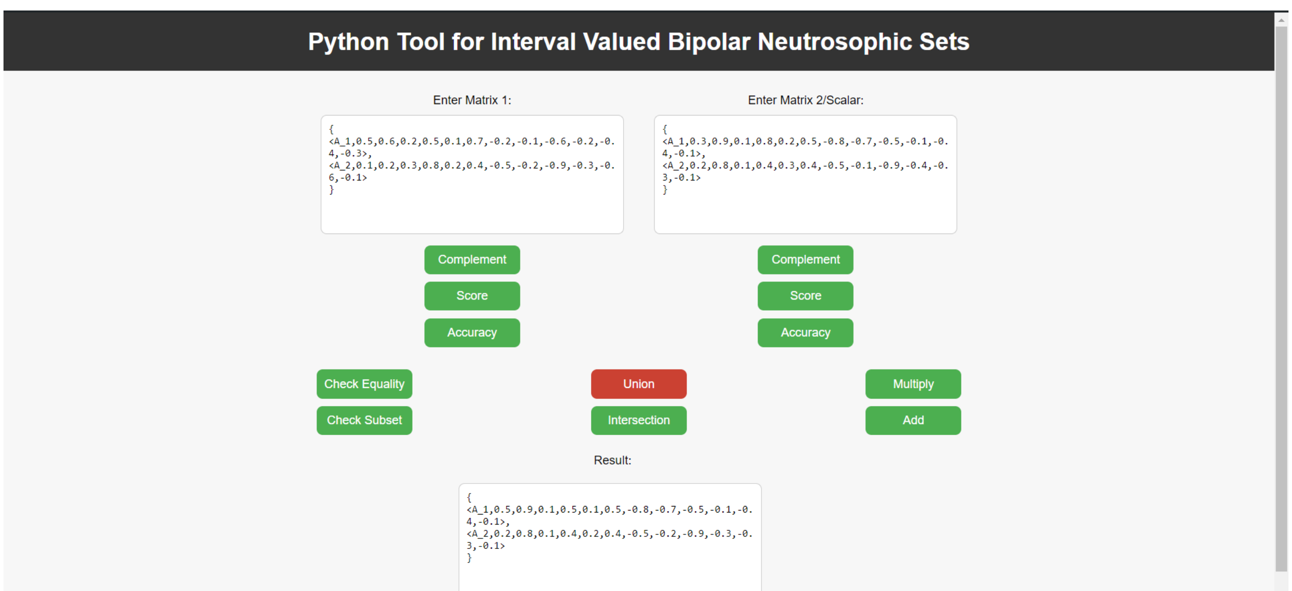

FIGURE 7. Sample Test Case for Union

The intersection of C_1 and C_2, $C_1 \cap C_2$ is calculated as:

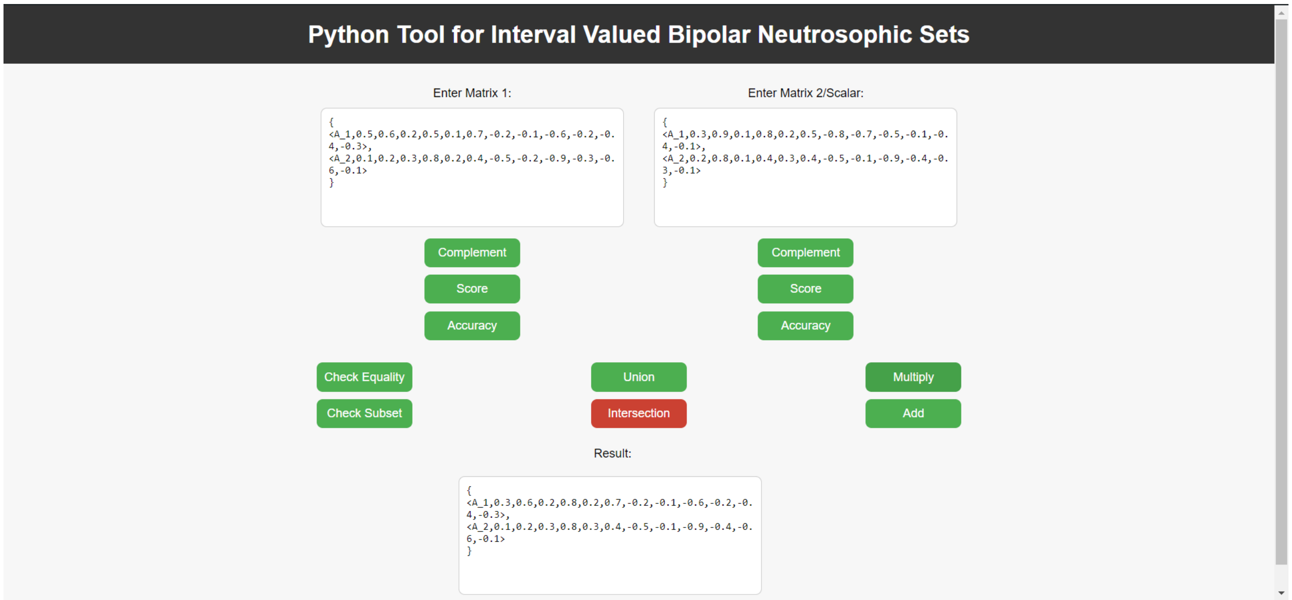

FIGURE 8. Sample Test Case for Intersection

The product of C_1 and C_2, $C_1 \times C_2$ is calculated as:

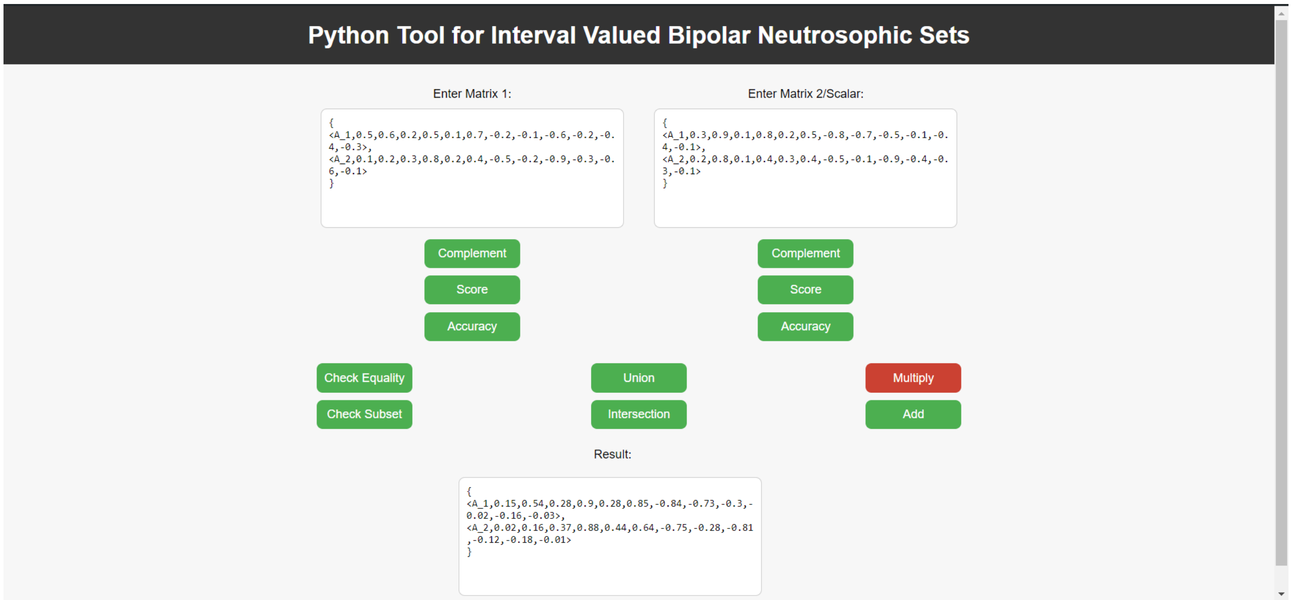

FIGURE 9. Sample Test Case for Multiplication

The below operation checks if C_1 is subset of C_2, $C_1 \subset C_2$:

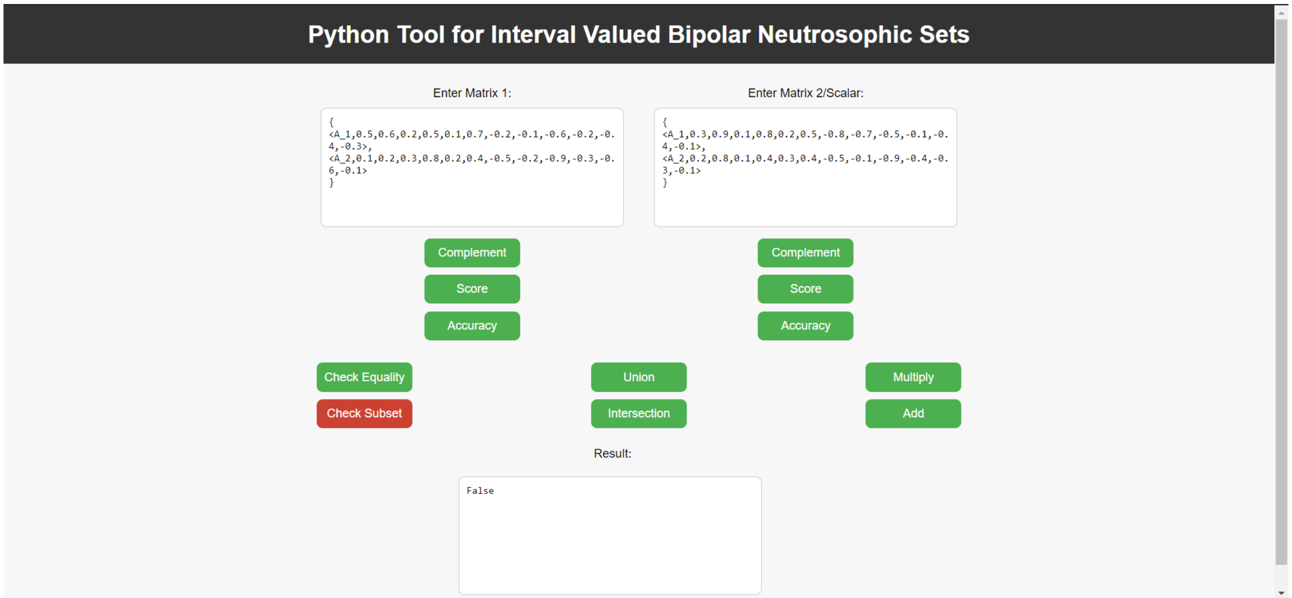

FIGURE 10. Sample Test Case for Subset

5|Conclusion

Let us consider a decision making problem adapted from Ye [13]. There is an investment company, which wants to invest a sum of money in the best option. There is a panel with the set of the four alternatives is denoted by C_1 = car company, C_2 = food company and C_3 = computer company.

The investment company must take a decision according to the set of the two attributes for each company and is denoted by A_1 = growth and A_2 = risk. Then according to the algorithm, we have:

Step 1:

Construct the required decision IVBNS.

Company 1.

$$C_1 = \left\{ \begin{array}{l} \langle A_1, [0.5, 0.6], [0.2, 0.5], [0.1, 0.7], \ [-0.2, -0.1], [-0.6, -0.2], [-0.4, -0.3] \rangle, \\ \langle A_2, [0.1, 0.2], [0.3, 0.8], [0.2, 0.4], [-0.5, -0.2], [-0.9, -0.3], [-0.6, -0.1] \rangle \end{array} \right\}$$

Company 2.

$$C_2 = \left\{ \begin{array}{l} \langle A_1, [0.3, 0.9], [0.1, 0.8], [0.2, 0.5], \ [-0.8, -0.7], [-0.5, -0.1], [-0.4, -0.1] \rangle, \\ \langle A_2, [0.2, 0.8], [0.1, 0.4], [0.3, 0.4], [-0.5, -0.1], [-0.9, -0.4], [-0.3, -0.1] \rangle \end{array} \right\}$$

Company 3.

$$C_3 = \left\{ \begin{array}{l} \langle A_1, [0.1, 0.6], [0.1, 0.5], [0.1, 0.4], \ [-0.5, -0.2], [-0.7, -0.3], [-0.4, -0.2] \rangle, \\ \langle A_2, [0.3, 0.4], [0.1, 0.6], [0.5, 0.7], [-0.5, -0.1], [-0.8, -0.7], [-0.9, -0.8] \rangle \end{array} \right\}$$

Step 2:

Compute the score value for each element in the IVBNS. Then find out the ratio between them.

$$S(C_1[A_1]) = 0.57$$
$$S(C_1[A_2]) = 0.48$$

Calculating the growth risk ratio,

$$S(C_1) = \frac{0.57}{0.48} = 1.1875$$

$$S(C_2[A_1]) = 0.43$$
$$S(C_2[A_2]) = 0.575$$

Calculating the growth risk ratio,

$$S(C_2) = \frac{0.43}{0.575} = 0.748$$

$$S(C_3[A_1]) = 0.542$$
$$S(C_3[A_2]) = 0.617$$

Calculating the growth risk ratio,

$$S(C_3) = \frac{0.542}{0.617} = 0.878$$

Step 3:

Rank all the systems according to the score value as: $S(C_1) > S(C_3) > S(C_2)$ and thus, C_1 is the most desirable alternative to invest in.

This research proposes a scalable approach, similar to [19], for selecting the optimal private-sector partner in large-scale projects, considering risk factors within a neutrosophic environment.

6|Conclusion

The development of the Interval Valued Bipolar Neutrosophic Sets (IVBNS) framework marks a significant advancement in the field of imprecise information processing. This novel framework, coupled with the accompanying Python toolbox, provides a comprehensive and user-friendly platform for effectively modeling and analyzing imprecise and inconsistent information prevalent in real-world applications. The IVBNS framework's inherent ability to handle fuzziness and uncertainties makes it particularly well-suited for image processing and segmentation tasks, particularly in medical imaging, where accurate segmentation is crucial for diagnosis and treatment planning.

Moving forward, we envision the IVBNS framework's applicability expanding to encompass a broader range of image processing and segmentation tasks. Our immediate focus lies in enhancing the accuracy and robustness of image segmentation algorithms for medical imaging and other applications involving uncertain or imprecise data. Additionally, we aim to delve deeper into the theoretical underpinnings of IVBNS and explore its potential for solving complex problems in diverse domains, including decision-making under uncertainty, pattern recognition, and data analysis.[15] The IVBNS framework holds immense promise for revolutionizing the field of imprecise information processing and paving the way for groundbreaking research and innovation.

7|Future Directions

Future research directions in neutrosophic software development includes:

- Expanding the functionalities of existing toolboxes to encompass a wider range of neutrosophic set types, including IVBNSs.

- Integrating neutrosophic set processing capabilities into existing software frameworks used in specific application domains (e.g., decision-making, engineering analysis). In the image processing domain, the significance of bipolar NS values can be used to differentiate the background and foreground images . This idea can be incorporated in dealing with segmentation.
- Developing user-friendly interfaces for neutrosophic software tools to enhance accessibility and usability for researchers and practitioners with varying levels of expertise.

By addressing these directions, the field of neutrosophic computing can move towards more robust and user-centric software tools, ultimately facilitating the broader adoption of neutrosophic theories in real-world applications.

Acknowledgements and Funding

The authors would like to express their sincere gratitude to the editor as well as the reviewers for their valuable suggestions. The authors declare that no research funding was received for this research work.

Author Contributions

Pranesh Kumar S P: Methodology, Analysis, Programming, Writing - Original Draft
Sofia Jennifer J: Conceptualization, Supervision, Writing - Review & Editing
Kalaivani C: Methodology, Data interpretation, Writing - Review & Editing
Florentin S: Validation, Writing - Review & Editing.

Data Availability

All data and resources utilized in this research are detailed within the manuscript. Additionally, relevant online resources are hyperlinked within the text for further exploration.

Conflicts of Interest

The authors declare that they have no conflict of interest.

References

[1] Abdullah, W. (2024). A study of neutrosophic sets and deep learning models for breast cancer classification *Multicriteria Algorithms with Applications*, *3*, 50-59. https://doi.org/10.61356/j.mawa.2024.3237

[2] Atanassov, K.T. (1999). Interval valued intuitionistic fuzzy sets. *Intuitionistic Fuzzy Sets: Theory and Applications*, *35*, (pp. 139-177). Physica, Heidelberg. https://doi.org/10.1007/978-3-7908-1870-3_2

[3] Atanassov, K.T. (1999). Interval valued intuitionistic fuzzy sets. *Intuitionistic Fuzzy Sets: Studies in Fuzziness and Soft Computing*, *35*. Physica, Heidelberg. https://doi.org/10.1007/978-3-7908-1870-3_2

[4] Bhat, S. A. (2023). An enhanced AHP group decision making model employing neutrosophic trapezoidal numbers. *Journal of Operational and Strategic Analytics*, *1(2)*, 81-89. https://doi.org/10.56578/josa010205

[5] Bhattacharya, P., & Pramanik, P. (2014). Some concepts on interval valued refined neutrosophic sets and their applications *Notes on Intuitionistic Fuzzy Sets*, *20(2)*, 31-44.

[6] Veeramani, C., Venugopal. R., & Edalatpanah, S. A. (2022). Neutrosophic DEMATEL approach for financial ratio performance evaluation of the NASDAQ Exchange. *Neutrosophic Sets and Systems*, *51*, 766–782. https://doi.org/10.5281/zenodo.7135415

[7] Deli, I., Ali, M., & Smarandache, F. (2015). Bipolar neutrosophic sets and their application based on multi-criteria decision making problems. *International Conference on Advanced Mechatronic Systems (ICAMechS)*, (pp. 249-254). Beijing, China. https://doi.org/10.1109/ICAMechS.2015.7287068

[8] Deli, I., Şubaş, Y., Smarandache, F., & Ali, M. (2016). Interval valued bipolar neutrosophic sets and their application in pattern recognition. *arXiv:1601.01266 [math.GM]*. https://doi.org/10.48550/arXiv.1601.01266

[9] Kandasamy, I. et al., (2023). NCMPy: A modelling software for neutrosophic cognitive maps based on python package. *Neutrosophic Systems with Applications*, *13*, 1-22. https://doi.org/10.61356/j.nswa.2024.114

[10] El-Ghareeb., & Haitham, A. (2019). Novel open source python neutrosophic package. *Neutrosophic Sets and Systems*, *25(1)*. https://digitalrepository.unm.edu/nss_journal/vol25/iss1/11

[11] Eskandari, S. (2022). PyIT-MLFS: A Python based information theoretical multi label feature selection library. *International Journal of Research in Industrial Engineering*, *11(1)*, 9–15. https://doi.org/10.22105/riej.2022.308916.1252

[12] Hosseinzadeh, E., & Tayyebi, J. (2023). A compromise solution for the neutrosophic multi-objective linear programming problem and its application in transportation problem. *Journal of Applied Research on Industrial Engineering*, *10(1)*, 1-10. https://doi.org/10.22105/jarie.2022.328580.1451

[13] Ye, Jun. (2014). Vector similarity measures of simplified neutrosophic sets and their application in multicriteria decision making. *International Journal of Fuzzy Systems*, *16*, 204-211.

[14] Kumar, R., Edalatpanah, S. A., Jha, S., & Singh, R. (2019). A novel approach to solve gaussian valued neutrosophic shortest path problems. *International Journal of Engineering and Advanced Technology*, *8(3)*, 347 – 353.

[15] Nordo, G., Jafari, S., Mehmood, A., & Bhimraj, B. (2024). A Python framework for neutrosophic sets and mappings. *Soft and Neutrosophic Topology Lab*, *65*, 199-236.

[16] M. Sallam, K., & Mohamed, A. W. (2023). Single valued neutrosophic sets for assessment quality of suppliers under uncertainty environment. *Multicriteria Algorithms With Applications*, *1*, 1-10. https://doi.org/10.61356/j.mawa.2023.15861

[17] Mohanta, K. K., & Toragay, O. (2023). Enhanced performance evaluation through neutrosophic data envelopment analysis leveraging pentagonal neutrosophic numbers. *Journal of Operational and Strategic Analytics*, *1(2)*, 70-80. https://doi.org/10.56578/josa010204

[18] Nagarajan, D., Kanchana, A., Jacob, K. et al. (2023). A novel approach based on neutrosophic Bonferroni mean operator of trapezoidal and triangular neutrosophic interval environments in multi-attribute group decision making. *Sci Rep*, *13*, 10455. https://doi.org/10.1038/s41598-023-37497-z

[19] Saberhoseini, S. F., Edalatpanah, S. A., & Sorourkhah, A. (2022). Choosing the best private sector partner according to the risk factors in neutrosophic environment. *Big Data and Computing Visions*, *2(2)*, 61-68. https://doi.org/10.22105/bdcv.2022.334005.1075

[20] Broumi, S. (2024). Toolbox for the interval valued bipolar neutrosophic matrices. *MATLAB Central File Exchange*, https://www.mathworks.com/matlabcentral/fileexchange/71735-toolbox-for-the-interval-valued-bipolar-neutrosophic-matrice

[21] Smarandache, F. (2014). A unifying field in logics neutrosophic logic neutrosophy neutrosophic set neutrosophic probability and statistics. *American Research Press,Rehoboth*, http://dx.doi.org/10.6084/M9.FIGSHARE.1014204

[22] Jennifer, S. J., & Sharmila, S. T. (2023). A neutrosophic set approach on chest X-Rays for automatic lung infection detection *Information Technology and Control*, *52*, 37-52. https://doi.org/10.5755/j01.itc.52.1.31520

[23] Jennifer, S. J., & Sharmila, S. T. (2022). Neutrosophic approach for glaucoma detection in retinal images, *Proceedings of The Romanian Academy, Series A* , *23(4)*, (pp. 389 - 398). https://acad.ro/sectii2002/proceedings/doc2022-4/09-Sofia-Jennifer.pdf

[24] Jennifer, S. J., & Sharmila, S. T. (2022). Neutrosophic approach of segmentation on thermal images case study drowsy driving application. *NeuroQuantology: An Interdisciplinary Journal of Neuroscience and Quantum Physics*, *20(8)*, 1-10.

[25] Uluçay, V., Deli, I. & Şahin, M. (2018). Similarity measures of bipolar neutrosophic sets and their application to multiple criteria decision making *Neural Computing & Applications*, *29(12)*, 739–748. https://doi.org/10.1007/s00521-016-2479-1

[26] Wang, H., Smarandache, F., Zhang, Y., & Sunderraman, R. (2012). Single valued neutrosophic sets. *Infinite study*, *10*, 10-14.

[27] Zadeh L.A. (1965). Fuzzy sets, *Information and Control*, *8(3)*, 338-353. https://doi.org/10.1016/S0019-9958(65)90241-X