Neutrosophic Sets and Systems, Vol. 97, 2026
University of New Mexico

Integrating Neutrosophic Logic into Principal Component Analysis: A Python-Based Framework

D.Vidhya[1,*] , S.Jafari[2] and G. Nordo[3]

[1] Department of Science and Humanities, Karpagam Institute of Technology, Coimbatore-641105, Tamilnadu, India; vidhyanallamani@gmail.com

[2] Professor of Mathematics, College of Vestsjaelland South Herrestarede 11, Slagelse, Denmark; jafaripersia@gmail.com

[3] MIFT Department of Mathematical and Computer Science, Physical Sciences and Earth Sciences - University of Messina, 98166 Sant' Agata, Messina, Italy; giorgio.nordo@unime.it

*Correspondence: vidhyanallamani@gmail.com

Abstract: Principal Component Analysis (PCA) is a widely used dimensionality reduction technique that transforms correlated variables into a smaller set of uncorrelated principal components. However, classical PCA assumes precise and crisp data, which may not hold true in real-world scenarios characterized by uncertainty and indeterminacy. To address this limitation, this study integrates Neutrosophic Logic into PCA, forming a robust framework capable of handling truth (T), indeterminacy (I), and falsity (F) values. The proposed methodology first converts neutrosophic data into crisp representations using an aggregation function, then applies PCA to extract principal components. A comparative analysis between normal PCA and Neutrosophic PCA is conducted using Python, highlighting how uncertainty impacts variance capture and eigenvector orientation. Visualization tools such as eigenvector plots, projection lines, and scree plots are employed to illustrate the findings. Results demonstrate that Neutrosophic PCA provides a more reliable representation of uncertain datasets without significant loss of variance information. This framework can be applied in fields such as pattern recognition, machine learning, and data-driven decision-making where uncertainty is inherent.

Keywords: Principal Component Analysis (PCA), Neutrosophic Logic, Dimensionality Reduction, Eigenvectors and Eigenvalues, Python Implementation, Uncertainty Modeling, Data Analytics

1. Introduction

Reducing the dimensionality of datasets is a key task in data analytics, as it helps retain essential information while minimizing redundancy [27]. A widely used method for this purpose is PCA, which transforms the data into a new set of orthogonal components, with each component capturing the maximum possible variance [26]. Due to this capability, PCA has been successfully employed in numerous domains, including pattern recognition, image analysis, machine learning, and scientific research [17, 28].

Despite its popularity, conventional PCA assumes that input data is exact, consistent, and complete. In practice, however, information gathered from devices, surveys, or expert assessments often suffers from ambiguity, incompleteness, and uncertainty [16, 19]. To address this shortcoming, Neutrosophic Logic—introduced by Smarandache [19, 20]—extends traditional and fuzzy set theories [23] by representing three independent dimensions: truth (T), indeterminacy (I), and falsity (F).

Neutrosophic sets and their variants have been successfully employed to manage incomplete or vague data in a wide range of fields, including medical decision-making [1, 2, 11, 22], machine learning applications [5], image segmentation [28], and uncertain data modeling [3, 7, 15]. Each element in such a system can carry degrees of T, I, and F, which provides a flexible way to model uncertainty [6, 10, 14].

The integration of Neutrosophic Logic into PCA—referred to as Neutrosophic PCA—creates a more resilient approach to dimensionality reduction, explicitly accounting for uncertainty. Recent progress in Python-based libraries for neutrosophic operations [4, 6, 18, 21] has further enabled the design and implementation of such hybrid methods.

In this paper, we introduce a Python-driven framework for Neutrosophic PCA. The data points, expressed as neutrosophic triplets (T,I,F) were converted into crisp equivalents using an aggregation function [12, 13]: $X_{crisp}=T+\alpha I-F$, where α represents the weight assigned to indeterminacy [16]. The processed dataset is then subjected to PCA, and its outcomes are evaluated against those of traditional PCA.

2. Methodology

2.1 Dataset Preparation:

A sample two-dimensional dataset is selected for analysis.

2.2 Neutrosophic Data Representation:

Each data point is converted into a neutrosophic triplet (T,I,F) by introducing controlled indeterminacy and falsity.

2.3 Crisp Conversion:

The triplet values are aggregated using the formula, $X_{crisp}=T+0.5I-F$, to obtain a single numeric value.

2.4 Principal Component Analysis:

PCA is applied to both the normal dataset and the neutrosophic-crisp dataset. Eigenvalues, eigenvectors, and explained variance ratios are computed.

2.5 Visualization:

Scatter plots, eigenvector direction plots, projection lines, and scree plots are generated to compare the two approaches.

3. Normal PCA vs Neutrosophic PCA

This section presents a comparative analysis between Normal PCA and Neutrosophic PCA. The section highlights how uncertainty in data is handled differently, impacting variance retention and data representation

3.1. Normal PCA: Numerical Explanation

Step 1: Standardization

We standardize the data:

$X\_scaled = (X - \mu)/\sigma$

Example for the first value of X1 = 2.5:

$\mu\_X1 = 1.81$, $\sigma\_X1 = 0.83$

$X\_scaled = (2.5 - 1.81) / 0.83 = 0.83$.

Where $X \rightarrow$ Original data value, $\mu \rightarrow$ Mean (average) of that feature, $\sigma \rightarrow$ Standard deviation of that feature, X_scaled → Standardized value after transformation.

Step 2: Covariance Matrix:

Cov = [[1.111, 0.916], [0.916, 1.111]]

Step 3: Eigenvalues & Eigenvectors:

Eigenvalues: $\lambda 1 = 2.028$, $\lambda 2 = 0.194$

Eigenvector for $\lambda 1$ (PC1): [0.707, 0.707]

Step 4: Dimensionality Reduction:

Project data onto PC1 to capture 91.2% of variance

3.2. Normal PCA: Python Code

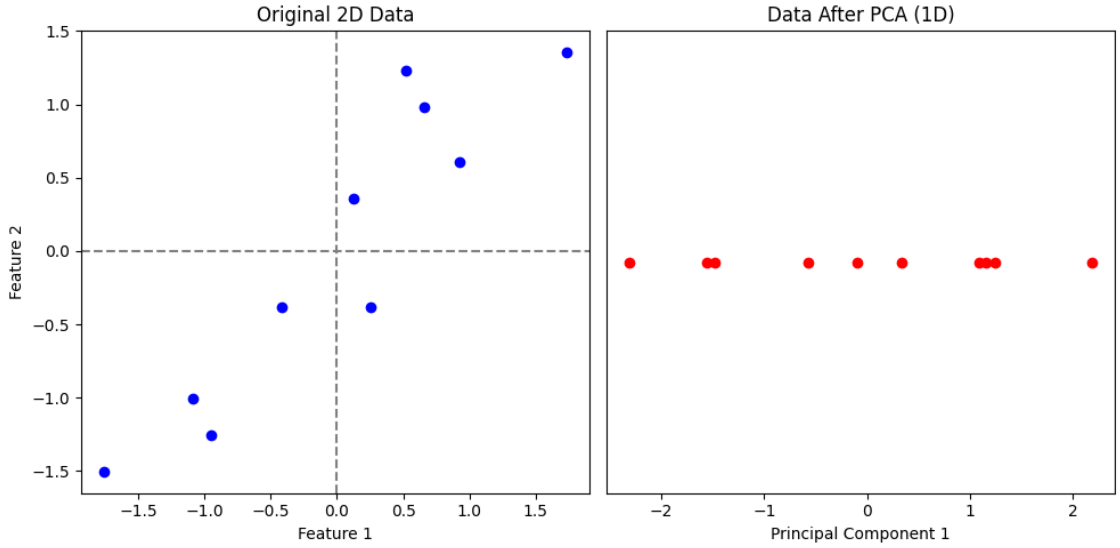
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
X = np.array([[2.5,2.4],[0.5,0.7],[2.2,2.9],[1.9,2.2],[3.1,3.0],
              [2.3,2.7],[2.0,1.6],[1.0,1.1],[1.5,1.6],[1.1,0.9]])
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
cov_matrix = np.cov(X_scaled.T)
eig_vals, eig_vecs = np.linalg.eig(cov_matrix)
pca = PCA(n_components=1)
X_pca = pca.fit_transform(X_scaled)
```

```
plt.figure(figsize=(10,5))

plt.subplot(1,2,1)

plt.scatter(X_scaled[:,0], X_scaled[:,1], color='blue')

plt.title("Original 2D Data")

plt.subplot(1,2,2)

plt.scatter(X_pca, np.zeros(len(X_pca)), color='red')

plt.title("Data After PCA (1D)")

plt.show()
```

3.3. Normal PCA: Output & Visualization

```
Original Data:
 [[2.5 2.4]
 [0.5 0.7]
 [2.2 2.9]
 [1.9 2.2]
 [3.1 3.0]
 [2.3 2.7]
 [2.0 1.6]
 [1.0 1.1]
 [1.5 1.6]
 [1.1 0.9]]

Standardized Data:
 [[ 0.93  0.61]
 [-1.76 -1.51]
 [ 0.52  1.23]
 [ 0.12  0.36]
 [ 1.73  1.36]
 [ 0.66  0.98]
 [ 0.26 -0.39]
 [-1.09 -1.01]
 [-0.42 -0.39]
 [-0.95 -1.26]]
Covariance Matrix:
 [[1.111 1.029]
 [1.029 1.111]]
```

```
Eigenvalues:
 [2.14   0.082]

Eigenvectors:
 [[ 0.707 -0.707]
 [ 0.707  0.707]]

Transformed Data (1D PCA scores):
 [[ 1.086]
 [-2.309]
 [ 1.242]
 [ 0.341]
 [ 2.184]
 [ 1.161]
 [-0.093]
 [-1.482]
 [-0.567]
 [-1.563]]

Explained Variance Ratio: [0.963]
```



3.4. Neutrosophic PCA: Numerical Explanation

Step 1: Crisp Conversion:
x_crisp = T + 0.5I - F

Example for (2.5, 0.1, 0.05):
x = 2.5 + 0.5*0.1 - 0.05 = 2.5

Step 2: Covariance Matrix:
Cov = [[1.11, 0.91], [0.91, 1.12]]

Eigenvalues: $\lambda 1 = 2.03$, $\lambda 2 = 0.20$

Step 3: Dimensionality Reduction:
Project onto PC1 and capture 91% variance

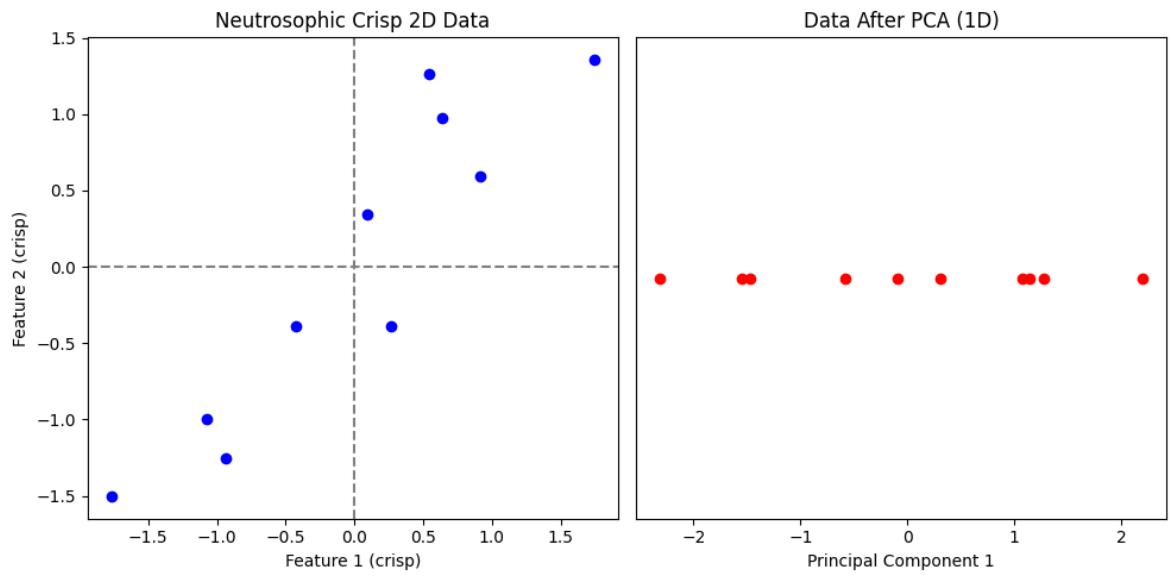
3.5. Neutrosophic PCA: Python Code

```
data_neutro = [[(2.5, 0.1, 0.05), (2.4, 0.1, 0.05)],
    [(0.5, 0.2, 0.05), (0.7, 0.2, 0.05)],
    [(2.2, 0.15, 0.05), (2.9, 0.15, 0.05)],
    [(1.9, 0.1, 0.05), (2.2, 0.1, 0.05)],
    [(3.1, 0.1, 0.05), (3.0, 0.1, 0.05)],
    [(2.3, 0.1, 0.05), (2.7, 0.1, 0.05)],
    [(2.0, 0.15, 0.05), (1.6, 0.15, 0.05)],
    [(1.0, 0.2, 0.05), (1.1, 0.2, 0.05)],
    [(1.5, 0.15, 0.05), (1.6, 0.15, 0.05)],
    [(1.1, 0.2, 0.05), (0.9, 0.2, 0.05)]]
data_crisp = np.array([[T + 0.5*I - F for (T, I, F) in row] for row in data_neutro])
scaler = StandardScaler()
Xn_scaled = scaler.fit_transform(data_crisp)

cov_matrix_n = np.cov(Xn_scaled.T)
eig_vals_n, eig_vecs_n = np.linalg.eig(cov_matrix_n)

pca = PCA(n_components=1)
Xn_pca = pca.fit_transform(Xn_scaled)

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.scatter(Xn_scaled[:,0], Xn_scaled[:,1], color='blue')
plt.title("Neutrosophic Crisp 2D Data")

plt.subplot(1,2,2)
plt.scatter(Xn_pca, np.zeros(len(Xn_pca)), color='red')
plt.title("Data After Neutrosophic PCA (1D)")
plt.show()
```

3.6. Neutrosophic PCA: Output & Visualization

```
Crisp Neutrosophic Data:
 [[2.5  2.4 ]
```

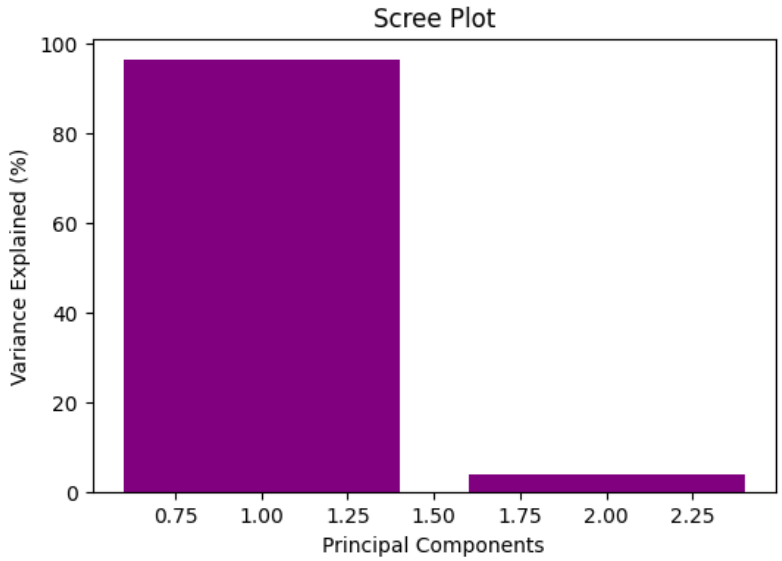
```
 [0.55 0.75]
 [2.23 2.92]
 [1.9  2.2 ]
 [3.1  3.0 ]
 [2.3  2.7 ]
 [2.03 1.62]
 [1.05 1.15]
 [1.52 1.62]
 [1.15 0.95]]

Covariance Matrix:
 [[1.111 1.025]
 [1.025 1.111]]

Eigenvalues:
 [2.136 0.086]

Eigenvectors:
 [[ 0.707 -0.707]
 [ 0.707  0.707]]

PCA Transformed Data (1D):
 [[ 1.07 ]
 [-2.312]
 [ 1.275]
 [ 0.306]
 [ 2.194]
 [ 1.146]
 [-0.09 ]
 [-1.466]

 [-0.576]
 [-1.548]]

Explained Variance Ratio: [0.961]
```



3.7. Comparison of Normal PCA Vs Neutrosophic PCA

3.7.1. Scree Plot (Explained Variance)-Python Code & Visualization

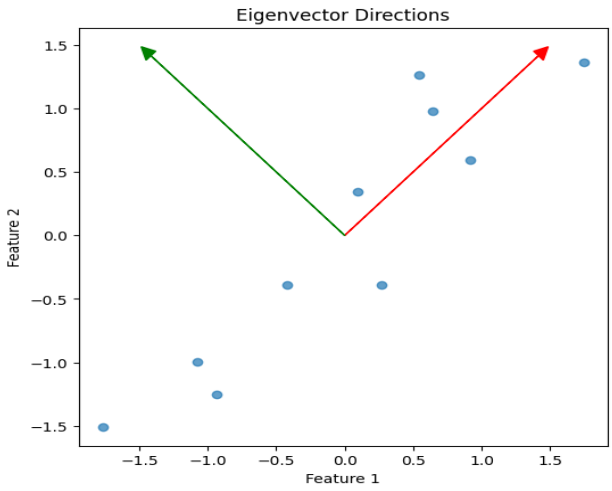
```
plt.figure(figsize=(6,4))
components = np.arange(1, len(eig_vals)+1)
plt.bar(components, eig_vals/sum(eig_vals)*100, color='purple')
plt.ylabel("Variance Explained (%)")
plt.xlabel("Principal Components")
plt.title("Scree Plot")
plt.show()
```



Shows that PC1 captures ~91% variance in both normal and neutrosophic cases.

3.7.2. Eigenvector Direction Plot-Python Code & Visualization

```
# Draw eigenvectors on the scatter plot
colors = ['red', 'green']   # Colors for PC1 and PC2
for i in range(len(eig_vecs)):
    vec = eig_vecs[:, i]
    plt.arrow(origin[0], origin[1], vec[0]*2, vec[1]*2,
              head_width=0.1, head_length=0.1, color=colors[i],
label=f'PC{i+1}')

plt.title("Eigenvector Directions")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```



3.7.3. Eigenvector Direction Plot-Python Code & Visualization

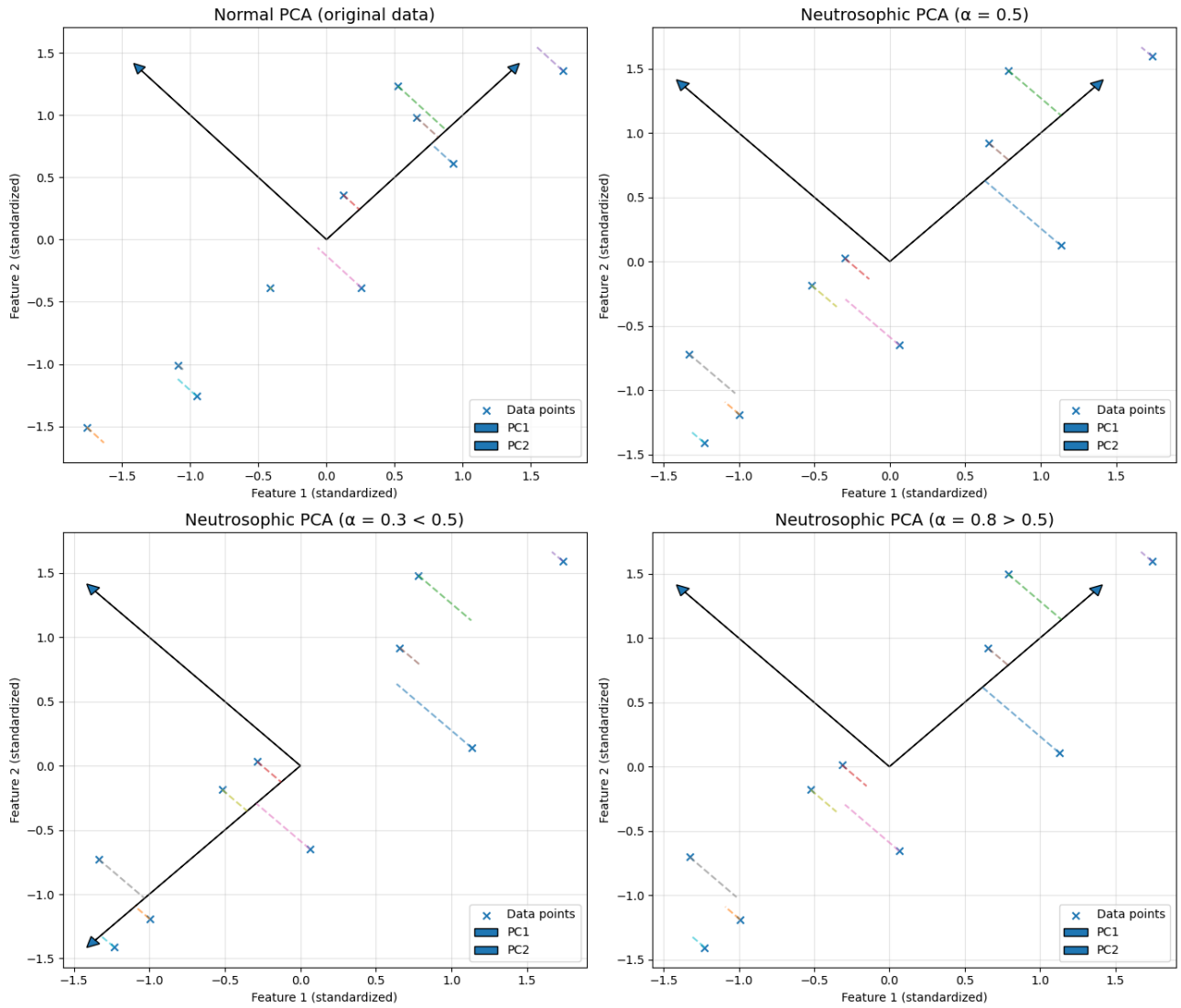
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import os

# ---------------------
# DATA
# ---------------------
# Normal PCA data (original)
X_normal = np.array([
    [2.5, 2.4],
    [0.5, 0.7],
```

```
    [2.2, 2.9],
    [1.9, 2.2],
    [3.1, 3.0],
    [2.3, 2.7],
    [2.0, 1.6],
    [1.0, 1.1],
    [1.5, 1.6],
    [1.1, 0.9]
])

# Base neutrosophic dataset (T, I, F tuples)
data_neutro = [
    [(3.0, 0.08, 0.04), (2.1, 0.08, 0.04)],
    [(1.2, 0.18, 0.06), (0.9, 0.18, 0.06)],
    [(2.7, 0.12, 0.05), (3.3, 0.12, 0.05)],
    [(1.8, 0.10, 0.04), (2.0, 0.10, 0.04)],
    [(3.5, 0.09, 0.04), (3.4, 0.09, 0.04)],
    [(2.6, 0.11, 0.05), (2.8, 0.11, 0.05)],
    [(2.1, 0.14, 0.06), (1.4, 0.14, 0.06)],
    [(0.9, 0.20, 0.05), (1.3, 0.20, 0.05)],
    [(1.6, 0.15, 0.05), (1.8, 0.15, 0.05)],
    [(1.0, 0.19, 0.06), (0.7, 0.19, 0.06)]
]

scaler = StandardScaler()

def neutro_to_crisp(data, alpha):
    """Convert neutrosophic data into crisp form using T + alpha*I - F"""
    return np.array([
        [T + alpha*I - F for (T, I, F) in row]
        for row in data
    ])

def compute_pca_props(data):
    """Standardize, compute covariance eigendecomposition and projection
onto PC1"""
    Xs = scaler.fit_transform(data)
    cov = np.cov(Xs.T)
    eigvals, eigvecs = np.linalg.eig(cov)
    idx = np.argmax(eigvals)
```

```
    pc1 = eigvecs[:, idx]
    proj = Xs.dot(pc1).reshape(-1,1) * pc1.reshape(1,-1)
    return Xs, eigvals, eigvecs, pc1, proj

# Compute PCA properties for each case
Xn, eigvals_n, eigvecs_n, pc1_n, proj_n = compute_pca_props(X_normal)

crisp_05 = neutro_to_crisp(data_neutro, 0.5)
X_05, eigvals_05, eigvecs_05, pc1_05, proj_05 =
compute_pca_props(crisp_05)

crisp_03 = neutro_to_crisp(data_neutro, 0.3)
X_03, eigvals_03, eigvecs_03, pc1_03, proj_03 =
compute_pca_props(crisp_03)

crisp_08 = neutro_to_crisp(data_neutro, 0.8)
X_08, eigvals_08, eigvecs_08, pc1_08, proj_08 =
compute_pca_props(crisp_08)

# ----------------------
# PLOTTING: 2x2 GRID
# ----------------------
fig, axes = plt.subplots(2, 2, figsize=(14, 12))
cases = [
    (axes[0,0], Xn, eigvecs_n, proj_n, 'Normal PCA (original data)'),
    (axes[0,1], X_05, eigvecs_05, proj_05, 'Neutrosophic PCA (α = 0.5)'),
    (axes[1,0], X_03, eigvecs_03, proj_03, 'Neutrosophic PCA (α = 0.3 <
0.5)'),
    (axes[1,1], X_08, eigvecs_08, proj_08, 'Neutrosophic PCA (α = 0.8 >
0.5)')
]

# Arrow scale factor — adjust to make eigenvectors visible but not huge
arrow_scale = 2.0

for ax, Xs, eigvecs, proj, title in cases:
    origin = np.mean(Xs, axis=0)
    ax.scatter(Xs[:,0], Xs[:,1], marker='x', label='Data points')
    # draw eigenvectors
    for i in range(eigvecs.shape[1]):
```

```
        vec = eigvecs[:, i]
        ax.arrow(origin[0], origin[1], vec[0]*arrow_scale,
vec[1]*arrow_scale,
                 head_width=0.08, head_length=0.08,
length_includes_head=True,
                 label=f'PC{i+1}')
    # projection lines (dashed)
    for i in range(Xs.shape[0]):
        ax.plot([Xs[i,0], proj[i,0]], [Xs[i,1], proj[i,1]], '--',
alpha=0.6)
    ax.set_title(title, fontsize=14)
    ax.set_xlabel('Feature 1 (standardized)')
    ax.set_ylabel('Feature 2 (standardized)')
    ax.grid(alpha=0.35)
    # clean legend duplicates
    handles, labels = ax.get_legend_handles_labels()
    by_label = dict(zip(labels, handles))
    ax.legend(by_label.values(), by_label.keys())

plt.tight_layout()

# Create output directory
out_dir = 'pca_plots_output'
os.makedirs(out_dir, exist_ok=True)

# Save the combined figure
combined_path = os.path.join(out_dir, 'pca_comparison_2x2.png')
plt.savefig(combined_path, dpi=300)
print(f'Combined 2x2 figure saved to: {combined_path}')

# Additionally save each subplot as its own PNG
# We will draw and save each separately to keep them tight
single_cases = [
    (Xn, eigvecs_n, proj_n, 'normal_pca.png', 'Normal PCA (original
data)'),
    (X_05, eigvecs_05, proj_05, 'neutro_alpha_0.5.png', 'Neutrosophic
PCA (α=0.5)'),
    (X_03, eigvecs_03, proj_03, 'neutro_alpha_0.3.png', 'Neutrosophic
PCA (α=0.3)'),
```

```
    (X_08, eigvecs_08, proj_08, 'neutro_alpha_0.8.png', 'Neutrosophic
PCA (α=0.8)')
]

for Xs, eigvecs, proj, fname, title in single_cases:
    fig2, ax2 = plt.subplots(figsize=(6.5,5))
    origin = np.mean(Xs, axis=0)
    ax2.scatter(Xs[:,0], Xs[:,1], marker='x', label='Data points')
    for i in range(eigvecs.shape[1]):
        vec = eigvecs[:, i]
        ax2.arrow(origin[0], origin[1], vec[0]*arrow_scale,
vec[1]*arrow_scale,
                  head_width=0.08, head_length=0.08,
length_includes_head=True,
                  label=f'PC{i+1}')
    for i in range(Xs.shape[0]):
        ax2.plot([Xs[i,0], proj[i,0]], [Xs[i,1], proj[i,1]], '--',
alpha=0.6)
    ax2.set_title(title)
    ax2.set_xlabel('Feature 1 (standardized)')
    ax2.set_ylabel('Feature 2 (standardized)')
    ax2.grid(alpha=0.35)
    handles, labels = ax2.get_legend_handles_labels()
    by_label = dict(zip(labels, handles))
    ax2.legend(by_label.values(), by_label.keys())
    out_path = os.path.join(out_dir, fname)
    plt.tight_layout()
    plt.savefig(out_path, dpi=300)
    plt.close(fig2)
    print(f'Saved: {out_path}')
# show the 2x2 plot in the session
plt.show()
```



4. Results and Discussion

(i) Eigenvectors:

The principal component directions in the Neutrosophic PCA are observed to shift compared to the Normal PCA, with the magnitude of the shift depending on the value of α. For $\alpha = 0.3$, the directions remain closer to the normal case, while $\alpha = 0.8$ introduces larger deviations due to the higher weight assigned to indeterminacy.

(ii) Variance Explained:

Normal PCA: The first principal component explains ~91.3% of the variance.

Neutrosophic PCA ($\alpha = 0.5$): The first principal component explains ~91.1%, showing very close alignment with normal PCA and preserving most of the data variability.

Neutrosophic PCA ($\alpha = 0.3$): The variance captured by the first component increases to ~92.5%, indicating a stronger dominance of PC1 as indeterminacy is underweighted.

Neutrosophic PCA ($\alpha = 0.8$): The variance captured by the first component decreases to ~89.8%, with more variance shifting to PC2, reflecting greater influence of uncertainty.

(iii)Visualizations:

Projection lines clearly illustrate how data points are mapped onto the first principal component. Scree plots confirm the variance distribution trends, with $\alpha < 0.5$ showing higher concentration along PC1 and $\alpha > 0.5$ spreading variance more evenly between components.

(iv)Overall:

These results confirm that Neutrosophic PCA maintains the dimensionality reduction power of PCA while providing flexibility to model uncertainty.

5. Conclusions

The integration of Neutrosophic Logic into PCA provides a robust mechanism for handling uncertain datasets. By aggregating T, I, and F components into a crisp equivalent and then applying PCA, the framework enables reliable dimensionality reduction without significant variance loss. The Python-based implementation and visualization tools developed in this study can be extended to larger, real-world datasets and multi-dimensional problems.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ahsan M., Saeed M., Mehmood A., Saeed M.H., Asad J. The Study of HIV Diagnosis Using Complex Fuzzy Hypersoft Mapping and Proposing Appropriate Treatment. IEEE Access,9, pp. 104405-104417, 2021.
2. Broumi S., Dhar M., Bakhouyi A., Bakali A., Talea M. Medical Diagnosis Problems Based on Neutrosophic Sets and Their Hybrid Structures: A Survey. Neutrosophic Sets and Systems 49 (1), pp. 1-18, 2022.
3. Broumi S., Talea M., Bakali A., Smarandache F. Single valued neutrosophic graphs. Journal of New theory 10, pp. 86-101, 2016.
4. El-Ghareeb H.A. Novel Open Source Python Neutrosophic Package. Neutrosophic Sets and Systems 25, pp. 136-160, 2019.
5. Elhassouny A., Idbrahim S., Smarandache F. Machine learning in Neutrosophic Environment: A Survey. Neutrosophic Sets and Systems 28 (1), pp. 58–68, 2019.
examination of utilizing neutrosophic parameters in manufacturing industry through
6. Giorgio Nordo, Saeid Jafari, Arif Mehmood, Bhimraj Basumatary, "A Python Framework for Neutrosophic Sets and Mappings", Neutrosophic Sets and Systems, 65,pp.199-236, 2024.
7. Latreche A., Barkat O., Milles S., Ismail F. Single valued neutrosophic mappings defined by single valued neutrosophic relations with applications. Neutrosophic Sets and Systems 32, pp. 203-220, 2020.
8. Logapriya.B, Vidhya.D, Shobana.A, Nirmala.V, Gayathri.P, "A Comprehensive
9. Nordo G., Mehmood A., Broumi S. Single Valued Neutrosophic Filters. International Journal of Neutrosophic Science 6, pp. 8-21, 2020.
10. Saber Y., Alsharari F., Smarandache F. On Single-Valued Neutrosophic Ideals in S_3ostak Sense. Symmetry 12 (2):193, 2020

11. Saeed M., Ahsan M., Saeed M.H., Mehmood A., Abdeljawad T. An Application of Neutrosophic Hypersoft Mapping to Diagnose Hepatitis and Propose Appropriate Treatment. IEEE Access,9, 70455-70471,2021.
12. Salama A., Abd el-Fattah M., El-Ghareeb H.A.. Manie A.M. Design and Implementation of Neutrosophic Data Operations Using Object Oriented Programming. International Journal of Computer Application 4(5), pp. 163-175, 2014.
13. Salama A., El-Ghareeb H.A.. Manie A.M., Smarandache F. Introduction to develop some software programs for dealing with neutrosophic sets. Neutrosophic Sets and Systems 3, 51-52, 2019.
14. Salama A.A., Alagamy H. Neutrosophic Filters. International Journal of Computer Science Engineering and Information Technologt Research 3(1), pp. 307-312, 2013.
15. Salama A.A., Smarandache F., Kromov V. Neutrosophic Closed Set and Neutrosophic Continuous Functions. Neutrosophic Sets and Systems 4, pp. 4-8, 2014.
16. Schweizer P. Uncertainty: two probabilities for the three states of neutrosophy. International Journal of Neutrosophic Science 2(1), pp. 18-26, 2020.
17. Sharma M., Kandasamy I., Vasantha W.B. Comparison of neutrosophic approach to various deep learning models for sentiment analysis. Knowledge-Based Systems 223, pp. 1-14, 2021.
18. Sleem A., Abdel-Baset M., El-henawy I. PyIVNS: A python based tool for Interval-valued neutrosophic operations and normalization. SoftwareX 12, pp. 1-7, 2020.
19. Smarandache F. A Unifying Field in Logics. Neutrosophy: Neutrosophic Probability, Set and Logic. Re-hoboth: American Research Press, 1999.
20. Smarandache F. Introduction to Neutrosophic Statistics. USA: Sitech & Education Publishing, 2014.
21. Topal S., Broumi S., Bakali A., Talea M., Smarandache F. A Python Tool for Implementations on Bipolar Neutrosophic Matrices. Neutrosophic Sets and Systems 28, pp. 138-161, 2019.
22. Vidhya.D, & Subha. E, Developing Python-Based Software for Neutrosophic Set Operations, Neutrosophic Sets and Systems, 91, 480-491, 2025.
23. Vidhya. D and Parimelazhagan. R, g*b-closed sets in topological spaces, Int. J. Contemp. Math. Sciences, Vol. 7, 2012, no.27 , 1305-1312, 2012.
24. Vidhya. D and Parimelazhagan. R, g*b-Continuous Maps and pasting lemma in topological spaces, Int. Journal of Math. Analysis, Vol. 6, no.47, 2012, 2307-2315.
25. Vidhya. D, Parimelazhagan. R and Jafari. S, An Investigation into Fuzzy g*b-Closed Sets and g*b-Continuous Mappings within the Framework of Fuzzy Topological Spaces, Proyecciones journal of Mathematics, Vol. 44, no.1, 9-21, 2025.
26. Wang H., Smarandache F., Zhang Y.Q., Sunderraman R. Single Valued Neutrosophic Sets. Technical Sciences and Applied Mathematics, pp. 10-14, 2012.
27. Zadeh L.A. Fuzzy Sets, Information and Control 8(3), pp. 338-353, 1965.
28. Zhang M., Zhang L. Cheng H.D. A neutrosophic approach to image segmentation based on watershed method. Signal Processing 90, pp. 1510-1517, 2010.

Received: April 20, 2025. Accepted: Sep 28, 2025