Centro de Investigación en Matemáticas, A.C.

Reconocimiento de Patrones

Tarea 6. Ejercicio 2. Inciso a)

CIMAT

Importamos las librerías que utilizaremos:

In [1]:
```
import nltk  # Importing the NLTK library, a powerful toolkit for natural language
import string  # Importing the string library for working with strings in Python.
import re  # Importing the regular expression library for pattern matching.
import os  # Importing the os library for interacting with the operating system.
import numpy as np  # Importing NumPy for numerical operations, especially working
import pandas as pd  # Importing Pandas for data manipulation and analysis, providi
import seaborn as sns  # Importing Seaborn for statistical data visualization based
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # Required for 3D plotting
from sklearn.preprocessing import StandardScaler  # Importing StandardScaler from s
from sklearn.decomposition import PCA  # Importing Principal Component Analysis (PC
from collections import Counter  # Importing Counter from the collections library t
from nltk.corpus import stopwords  # Importing stopwords from the NLTK corpus, whic
from nltk.stem import SnowballStemmer  # Importing SnowballStemmer from NLTK for st
from sklearn.feature_extraction.text import CountVectorizer  # Importing CountVecto
from sklearn.model_selection import train_test_split  # Importing train_test_split
from sklearn.preprocessing import LabelEncoder  # Importing LabelEncoder from sciki
from sklearn.svm import SVC  # Importing Support Vector Classifier (SVC) from sciki
from sklearn.metrics import classification_report, balanced_accuracy_score  # Impor
from sklearn.linear_model import LogisticRegression  # Importing LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree  # Importing DecisionTre
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import Huber
import matplotlib.colors as mcolors
from sklearn import manifold
```

In [2]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[2]: True

Reutilizamos las mismas funciones que en la tarea 5:

In [3]:
```
# Define preprocessing function
def preprocess(text, stemmer, stop_words):
    # Lowercase the input text to ensure consistent processing.
    text = text.lower()
    # Remove punctuation and numbers from the text.
    # This step helps to remove noise and improves the quality of the processed tex
    text = re.sub(r'[\d]+', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Tokenize the text into individual words or tokens.
    tokens = text.split()
    # Remove stop words and stem the remaining tokens.
    # Stop words are common words (e.g., "the", "a", "is") that are typically remov
    # Stemming reduces words to their root form.
    tokens = [stemmer.stem(word) for word in tokens if word not in stop_words and l
    return ' '.join(tokens)
```

In [4]:
```
def split_into_chapters(text):
    # Split text into chapters based on "Chapter x:" headings
    chapters = []
    current_chapter = []
    for line in text.splitlines():
        # Check if the line is a chapter heading (e.g., "Chapter 1:")
        if re.match(r'^\s*Chapter\s+\d+\s*:', line, re.IGNORECASE):
            # If there's a current chapter, add it to the list of chapters
            if current_chapter:
                chapters.append('\n'.join(current_chapter))
                # Reset the current chapter
                current_chapter = []
        # Add the current line to the current chapter
        current_chapter.append(line)
    # Add the last chapter if there's any remaining content
    if current_chapter:
        chapters.append('\n'.join(current_chapter))
    return chapters
```

In [5]:
```
def process_books(directory, k, language):
    """
    Processes a directory of text files, splitting into chapters,
    applying preprocessing, retaining only the k most common words,
    and returning a CountVectorizer object.

    Args:
        directory: Path to the directory with .txt files.
        k: Number of most common words to retain.
        language: Language for stemming and stopwords.

    Returns:
        A tuple (X, vectorizer, chapter_ids) where:
        - X is the matrix representation (chapters x words),
        - vectorizer is the fitted CountVectorizer,
        - chapter_ids is a list like ["book1_chapter1", "book1_chapter2", ...]
    """
```

```
    all_texts = []
    chapter_ids = []

    stemmer = SnowballStemmer(language)
    stop_words = set(stopwords.words(language))

    for filename in os.listdir(directory):
        if filename.endswith(".txt"):
            filepath = os.path.join(directory, filename)
            with open(filepath, 'r', encoding='utf-8') as f:
                text = f.read()
                chapters = split_into_chapters(text)
                for idx, chapter in enumerate(chapters, 1):
                    processed_chapter = preprocess(chapter, stemmer, stop_words)
                    all_texts.append(processed_chapter)
                    chapter_ids.append(f"{filename[:-4]}_chapter{idx}")

    # Build vocabulary of the k most common words
    all_word_list = [word for text in all_texts for word in text.split()]
    most_common_words = [word for word, _ in Counter(all_word_list).most_common(k)]

    # Vectorize
    vectorizer = CountVectorizer(min_df=1, vocabulary=most_common_words)
    X = vectorizer.fit_transform(all_texts)

    return X, vectorizer, chapter_ids
```

In [6]:
```
X, vectorizer, books = process_books('./books',100,"english")
```

In [7]:
```
# Load titles
titulos_df = pd.read_csv('./titulos.csv')

# Create mapping: filename -> title, author
book_mapping = dict(zip(titulos_df['title'], titulos_df['author']))

# Build a DataFrame for chapters
chapter_df = pd.DataFrame({
    'chapter_id': books,   # e.g., 'book1_chapter1'
    'text': list(X.toarray())
})

# Extract book name from chapter_id
chapter_df['book_name'] = chapter_df['chapter_id'].apply(lambda x: re.sub(r'_chapte

# Now map title and author
chapter_df['title'] = chapter_df['book_name']
chapter_df['author'] = chapter_df['title'].map(book_mapping)

# Expand the X matrix into columns
X_dense = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())
```

In [8]:
```
# Create a LabelEncoder object to convert categorical author names into numerical v
label_encoder = LabelEncoder()
# Fit the LabelEncoder to the 'author' column of the DataFrame and transform it int
# These numerical values are then stored in the 'y' variable.
```

```python
y = label_encoder.fit_transform(chapter_df['author'])
# Scale the data to have zero mean and unit variance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_dense)
```

Ahora, creamos un autoencoder por una red neuronal de Keras que consiste en tres capas de codificación y tres de decodificación, lo que podemos ver a continuación:

In [9]:
```python
# Build the autoencoder
input_dim = X_scaled.shape[1]  # should be 100

# Input layer
input_layer = Input(shape=(input_dim,))

# Encoder
encode1 = Dense(64, activation='relu')(input_layer)
encode2 = Dense(32, activation='relu')(encode1)
reduced = Dense(2, activation='linear')(encode2)  # final 2D representation

# Decoder
decode1 = Dense(32, activation='relu')(reduced)
decode2 = Dense(64, activation='relu')(decode1)
decoded = Dense(input_dim, activation='linear')(decode2)

# Full autoencoder model
autoencoder = Model(inputs=input_layer, outputs=decoded, name="Autoencoder")

# Encoder model (for 2D representation)
encoder = Model(inputs=input_layer, outputs=reduced)

# Compile and train
autoencoder.compile(optimizer=Adam(learning_rate=1e-3,beta_1=0.5), loss=Huber(delta

autoencoder.summary()
```

```
Model: "Autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 100)]             0

 dense (Dense)               (None, 64)                6464

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 2)                 66

 dense_3 (Dense)             (None, 32)                96

 dense_4 (Dense)             (None, 64)                2112

 dense_5 (Dense)             (None, 100)               6500

=================================================================
Total params: 17318 (67.65 KB)
Trainable params: 17318 (67.65 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Entrenamos la red neuronal con los datos escalados:

In [10]:
```
autoencoder.fit(
    X_scaled,
    X_scaled,
    epochs=100,
    batch_size=32,
    shuffle=True,
    validation_split=0.1,
    verbose=1
)
```

```
Epoch 1/100
17/17 [==============================] - 3s 16ms/step - loss: 0.2529 - val_loss: 0.1
916
Epoch 2/100
17/17 [==============================] - 0s 8ms/step - loss: 0.2395 - val_loss: 0.17
66
Epoch 3/100
17/17 [==============================] - 0s 8ms/step - loss: 0.2212 - val_loss: 0.17
30
Epoch 4/100
17/17 [==============================] - 0s 8ms/step - loss: 0.2061 - val_loss: 0.17
11
Epoch 5/100
17/17 [==============================] - 0s 8ms/step - loss: 0.2002 - val_loss: 0.17
08
Epoch 6/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1978 - val_loss: 0.17
12
Epoch 7/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1997 - val_loss: 0.17
03
Epoch 8/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1943 - val_loss: 0.16
98
Epoch 9/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1921 - val_loss: 0.16
95
Epoch 10/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1957 - val_loss: 0.16
86
Epoch 11/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1911 - val_loss: 0.16
88
Epoch 12/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1909 - val_loss: 0.16
81
Epoch 13/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1878 - val_loss: 0.16
84
Epoch 14/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1865 - val_loss: 0.16
78
Epoch 15/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1861 - val_loss: 0.16
81
Epoch 16/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1846 - val_loss: 0.16
81
Epoch 17/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1834 - val_loss: 0.16
81
Epoch 18/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1836 - val_loss: 0.16
81
Epoch 19/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1852 - val_loss: 0.16
```

```
81
Epoch 20/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1836 - val_loss: 0.16
83
Epoch 21/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1824 - val_loss: 0.16
88
Epoch 22/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1810 - val_loss: 0.16
85
Epoch 23/100
17/17 [==============================] - 0s 9ms/step - loss: 0.1818 - val_loss: 0.16
84
Epoch 24/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1812 - val_loss: 0.16
86
Epoch 25/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1792 - val_loss: 0.16
84
Epoch 26/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1810 - val_loss: 0.16
88
Epoch 27/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1810 - val_loss: 0.16
88
Epoch 28/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1771 - val_loss: 0.16
91
Epoch 29/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1783 - val_loss: 0.16
86
Epoch 30/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1801 - val_loss: 0.16
95
Epoch 31/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1778 - val_loss: 0.16
87
Epoch 32/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1800 - val_loss: 0.16
93
Epoch 33/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1780 - val_loss: 0.17
03
Epoch 34/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1767 - val_loss: 0.17
01
Epoch 35/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1786 - val_loss: 0.16
96
Epoch 36/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1786 - val_loss: 0.17
04
Epoch 37/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1741 - val_loss: 0.16
97
Epoch 38/100
```

```
17/17 [==============================] - 0s 6ms/step - loss: 0.1749 - val_loss: 0.16
97
Epoch 39/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1765 - val_loss: 0.17
03
Epoch 40/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1759 - val_loss: 0.16
98
Epoch 41/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1763 - val_loss: 0.17
04
Epoch 42/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1752 - val_loss: 0.17
05
Epoch 43/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1745 - val_loss: 0.17
07
Epoch 44/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1761 - val_loss: 0.17
09
Epoch 45/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1730 - val_loss: 0.17
08
Epoch 46/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1759 - val_loss: 0.17
03
Epoch 47/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1739 - val_loss: 0.17
10
Epoch 48/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1733 - val_loss: 0.17
09
Epoch 49/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1731 - val_loss: 0.17
11
Epoch 50/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1745 - val_loss: 0.17
11
Epoch 51/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1716 - val_loss: 0.17
10
Epoch 52/100
17/17 [==============================] - 0s 15ms/step - loss: 0.1712 - val_loss: 0.1
713
Epoch 53/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1744 - val_loss: 0.17
17
Epoch 54/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1701 - val_loss: 0.17
16
Epoch 55/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1698 - val_loss: 0.17
21
Epoch 56/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1723 - val_loss: 0.17
26
```

```
Epoch 57/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1716 - val_loss: 0.17
22
Epoch 58/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1730 - val_loss: 0.17
13
Epoch 59/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1721 - val_loss: 0.17
08
Epoch 60/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1769 - val_loss: 0.17
14
Epoch 61/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1698 - val_loss: 0.17
25
Epoch 62/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1703 - val_loss: 0.17
18
Epoch 63/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1728 - val_loss: 0.17
21
Epoch 64/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1701 - val_loss: 0.17
20
Epoch 65/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1714 - val_loss: 0.17
21
Epoch 66/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1727 - val_loss: 0.17
22
Epoch 67/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1724 - val_loss: 0.17
23
Epoch 68/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1735 - val_loss: 0.17
20
Epoch 69/100
17/17 [==============================] - 0s 10ms/step - loss: 0.1725 - val_loss: 0.1
718
Epoch 70/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1722 - val_loss: 0.1
719
Epoch 71/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1695 - val_loss: 0.1
727
Epoch 72/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1719 - val_loss: 0.1
722
Epoch 73/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1706 - val_loss: 0.17
36
Epoch 74/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1684 - val_loss: 0.17
21
Epoch 75/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1679 - val_loss: 0.17
```

```
31
Epoch 76/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1684 - val_loss: 0.17
26
Epoch 77/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1684 - val_loss: 0.17
24
Epoch 78/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1696 - val_loss: 0.17
23
Epoch 79/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1727 - val_loss: 0.17
27
Epoch 80/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1674 - val_loss: 0.17
29
Epoch 81/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1718 - val_loss: 0.17
26
Epoch 82/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1699 - val_loss: 0.17
21
Epoch 83/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1682 - val_loss: 0.17
20
Epoch 84/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1701 - val_loss: 0.17
24
Epoch 85/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1674 - val_loss: 0.17
17
Epoch 86/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1674 - val_loss: 0.17
34
Epoch 87/100
17/17 [==============================] - 0s 19ms/step - loss: 0.1689 - val_loss: 0.1
728
Epoch 88/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1679 - val_loss: 0.1
724
Epoch 89/100
17/17 [==============================] - 0s 19ms/step - loss: 0.1679 - val_loss: 0.1
727
Epoch 90/100
17/17 [==============================] - 0s 19ms/step - loss: 0.1696 - val_loss: 0.1
730
Epoch 91/100
17/17 [==============================] - 0s 19ms/step - loss: 0.1658 - val_loss: 0.1
726
Epoch 92/100
17/17 [==============================] - 0s 19ms/step - loss: 0.1695 - val_loss: 0.1
732
Epoch 93/100
17/17 [==============================] - 0s 17ms/step - loss: 0.1686 - val_loss: 0.1
725
Epoch 94/100
```

```
17/17 [==============================] - 0s 6ms/step - loss: 0.1685 - val_loss: 0.17
23
Epoch 95/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1686 - val_loss: 0.17
32
Epoch 96/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1675 - val_loss: 0.17
27
Epoch 97/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1700 - val_loss: 0.17
29
Epoch 98/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1678 - val_loss: 0.17
25
Epoch 99/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1661 - val_loss: 0.17
38
Epoch 100/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1688 - val_loss: 0.17
35
```

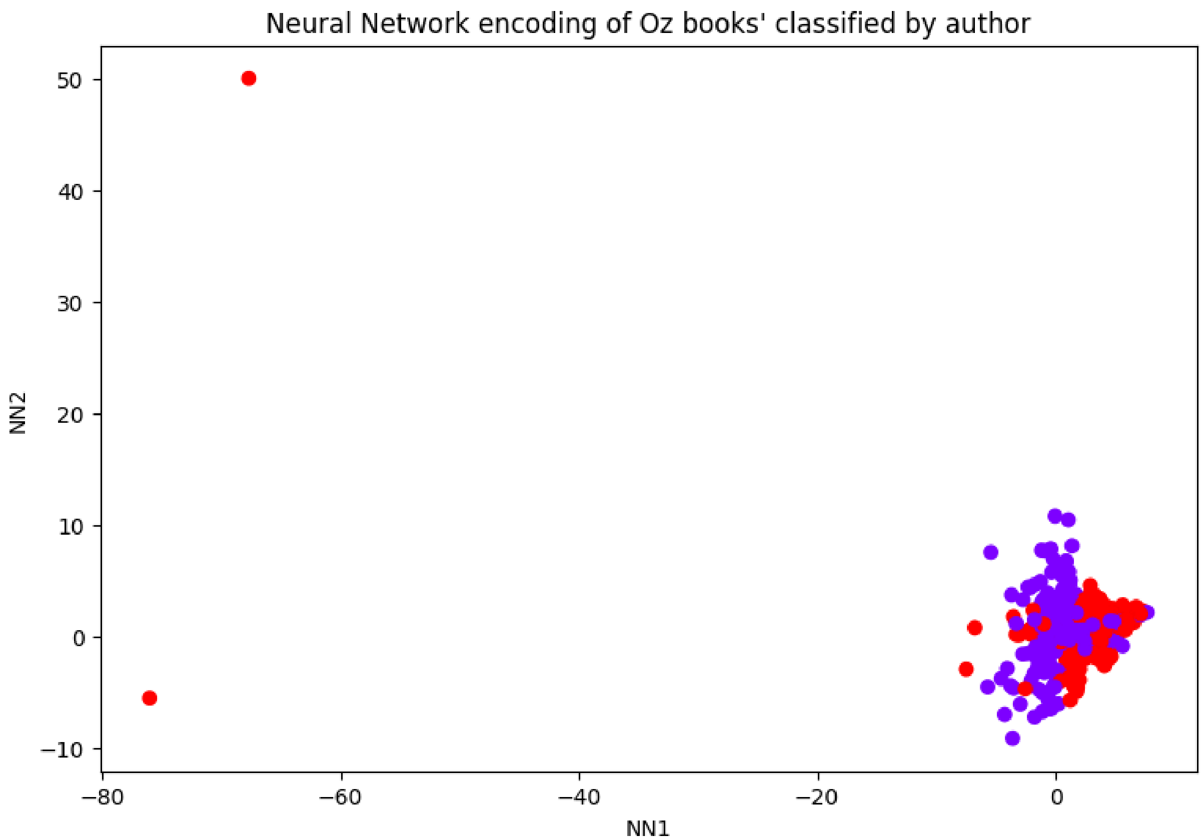
Out[10]:
```
<keras.src.callbacks.History at 0x7fa4ace6e080>
```

Ahora, pasamos los datos por la parte codificadora de la red neuronal:

In [11]:
```python
# Get 2D encoded data
X_encoded = encoder.predict(X_scaled)
```

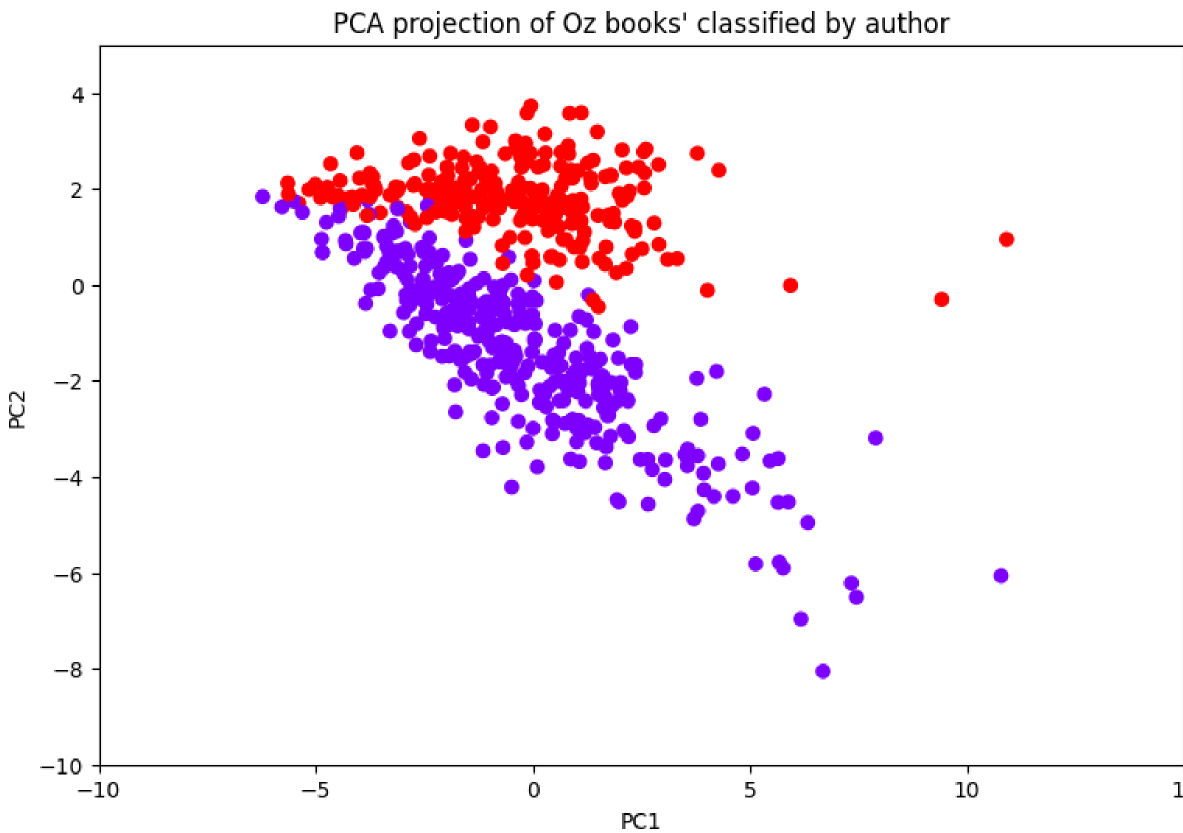
```
19/19 [==============================] - 0s 2ms/step
```

In [12]:
```python
# Create a scatter plot of the data in the Encoder dimensions.
# The color of each point is determined by the corresponding class label.
plt.figure(figsize=(9,6))
plt.scatter(X_encoded[:,0],X_encoded[:,1], c = y, cmap = 'rainbow')
#plt.xlim(-15, 10)
#plt.ylim(-10,5)
# Add a title to the plot.
plt.title('Neural Network encoding of Oz books\' classified by author')
# Label the x-axis as "NN1" (first Neural Network component)
plt.xlabel("NN{}".format(1))
# Label the y-axis as "NN2" (second Neural Network component)
plt.ylabel("NN{}".format(2))
# Display the plot
plt.show()
```



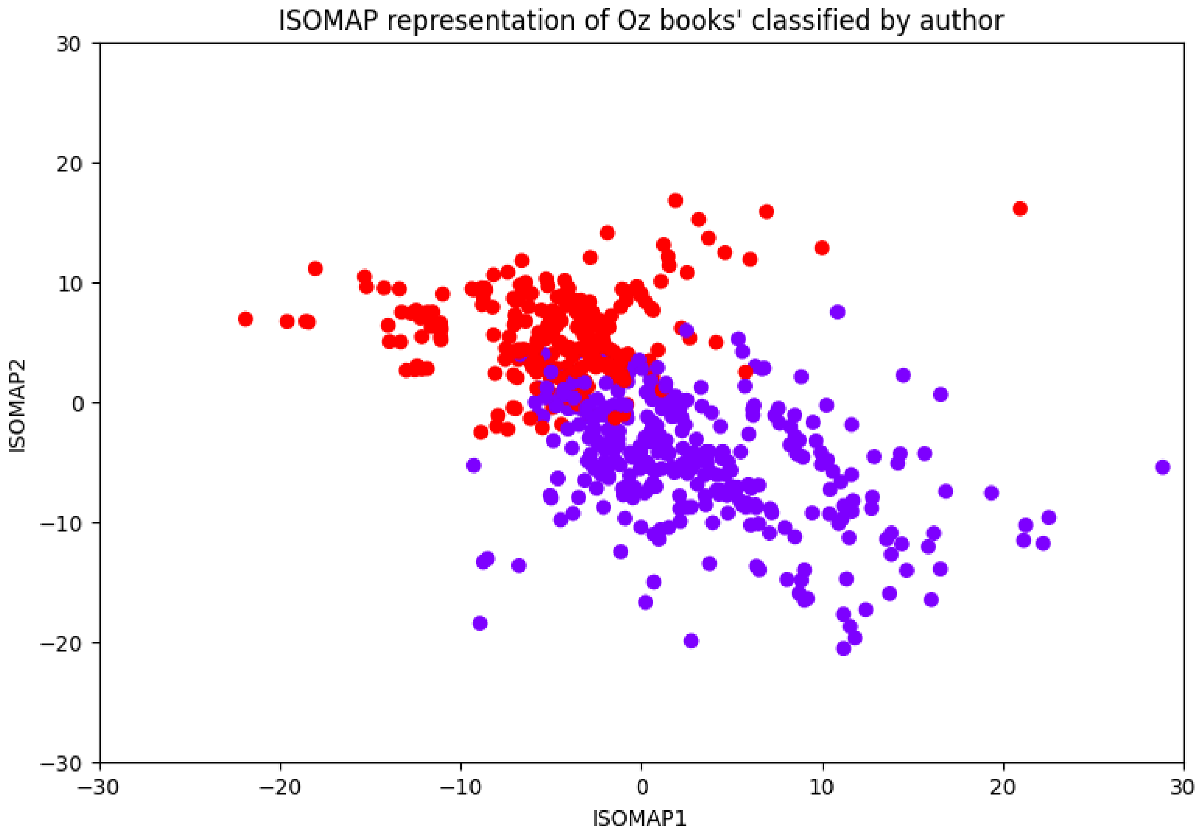
In [13]:
```
# Perform Principal Component Analysis (PCA) to reduce data to 2 dimensions
pca = PCA(n_components=2)
pca_data = pca.fit_transform(X_scaled)
```

In [14]:
```
# Create a scatter plot of the data in the first two principal components.
# The color of each point is determined by the corresponding class label.
plt.figure(figsize=(9,6))
plt.scatter(pca_data[:,0],pca_data[:,1], c = y, cmap = 'rainbow')
plt.xlim(-10, 15)
plt.ylim(-10,5)
# Add a title to the plot.
plt.title('PCA projection of Oz books\' classified by author')
# Label the x-axis as "PC1" (first principal component)
plt.xlabel("PC{}".format(1))
# Label the y-axis as "PC2" (second principal component)
plt.ylabel("PC{}".format(2))
# Display the plot
plt.show()
```



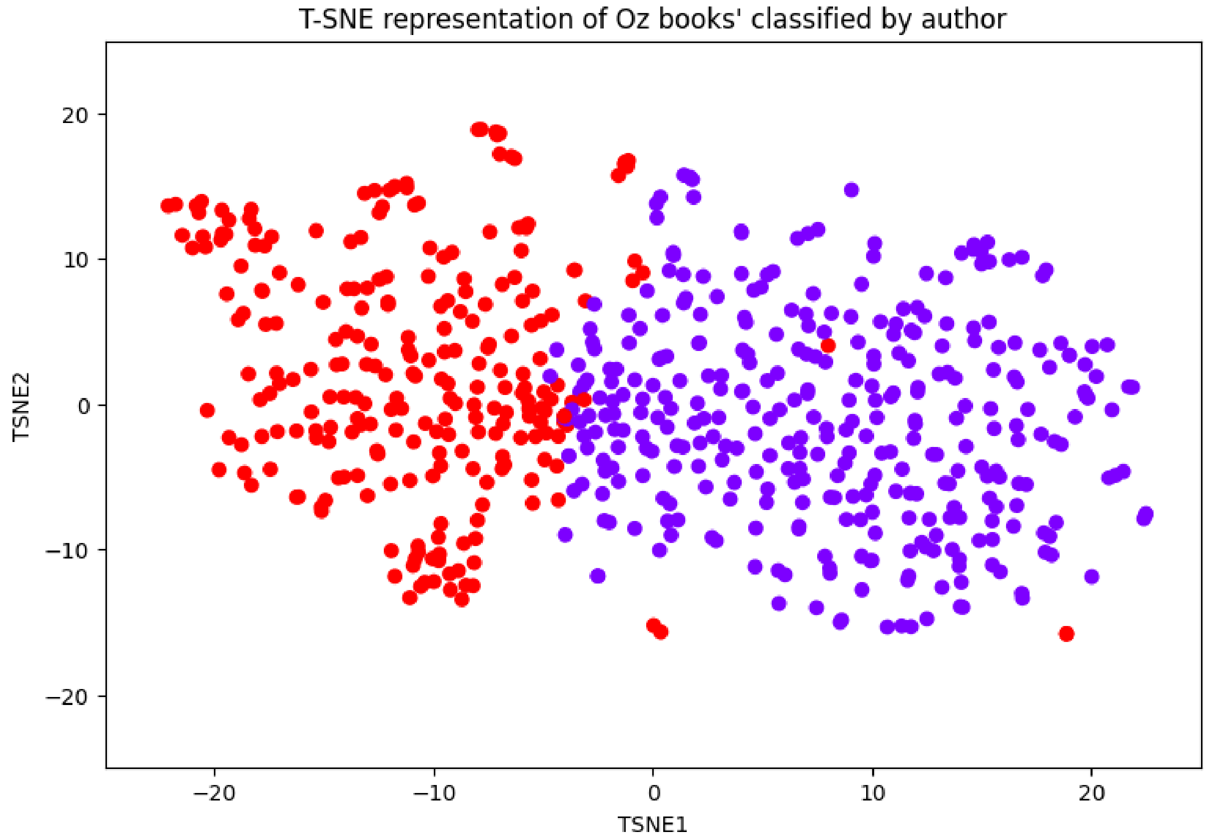
In [15]:
```
iso = manifold.Isomap(n_components=2)
X_iso = iso.fit_transform(X_scaled)
```

In [16]:
```
# Create a scatter plot of the data representation via ISOMAP.
# The color of each point is determined by the corresponding class label.
plt.figure(figsize=(9,6))
plt.scatter(X_iso[:,0],X_iso[:,1], c = y, cmap = 'rainbow')
plt.xlim(-30, 30)
plt.ylim(-30,30)
# Add a title to the plot.
plt.title('ISOMAP representation of Oz books\' classified by author')
# Label the x-axis as "NN1" (first Neural Network component)
plt.xlabel("ISOMAP{}".format(1))
# Label the y-axis as "NN2" (second Neural Network component)
plt.ylabel("ISOMAP{}".format(2))
# Display the plot
plt.show()
```



In [17]:
```
tsne = manifold.TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)
```

In [18]:
```
# Create a scatter plot of the data representation via ISOMAP.
# The color of each point is determined by the corresponding class label.
plt.figure(figsize=(9,6))
plt.scatter(X_tsne[:,0],X_tsne[:,1], c = y, cmap = 'rainbow')
plt.xlim(-25, 25)
plt.ylim(-25, 25)
# Add a title to the plot.
plt.title('T-SNE representation of Oz books\' classified by author')
# Label the x-axis as "NN1" (first Neural Network component)
plt.xlabel("TSNE{}".format(1))
# Label the y-axis as "NN2" (second Neural Network component)
plt.ylabel("TSNE{}".format(2))
# Display the plot
plt.show()
```



En este caso, casi todos los métodos resultan en datos bastante separables. Si acaso, el mejor es T-SNE.

In [ ]:
```
# Build the autoencoder
input_dim = X_scaled.shape[1]  # should be 100

# Input layer
input_layer = Input(shape=(input_dim,))

# Encoder
encode1 = Dense(64, activation='relu')(input_layer)
encode2 = Dense(32, activation='relu')(encode1)
reduced = Dense(3, activation='linear')(encode2)  # final 3D representation

# Decoder
decode1 = Dense(32, activation='relu')(reduced)
decode2 = Dense(64, activation='relu')(decode1)
decoded = Dense(input_dim, activation='linear')(decode2)

# Full autoencoder model
autoencoder = Model(inputs=input_layer, outputs=decoded, name="Autoencoder-3D")

# Encoder model (for 3D representation)
encoder = Model(inputs=input_layer, outputs=reduced)

# Compile and train
autoencoder.compile(optimizer=Adam(learning_rate=1e-3,beta_1=0.5), loss=Huber(delta

autoencoder.summary()
```

```
Model: "Autoencoder-3D"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 100)]             0

 dense_6 (Dense)             (None, 64)                6464

 dense_7 (Dense)             (None, 32)                2080

 dense_8 (Dense)             (None, 3)                 99

 dense_9 (Dense)             (None, 32)                128

 dense_10 (Dense)            (None, 64)                2112

 dense_11 (Dense)            (None, 100)               6500

=================================================================
Total params: 17383 (67.90 KB)
Trainable params: 17383 (67.90 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Entrenamos la red neuronal con los datos escalados:

In [20]:
```python
autoencoder.fit(
    X_scaled,
    X_scaled,
    epochs=100,
    batch_size=32,
    shuffle=True,
    validation_split=0.1,
    verbose=1
)
```

```
Epoch 1/100
17/17 [==============================] - 1s 24ms/step - loss: 0.2499 - val_loss: 0.1
844
Epoch 2/100
17/17 [==============================] - 0s 18ms/step - loss: 0.2341 - val_loss: 0.1
737
Epoch 3/100
17/17 [==============================] - 0s 19ms/step - loss: 0.2180 - val_loss: 0.1
716
Epoch 4/100
17/17 [==============================] - 0s 15ms/step - loss: 0.2051 - val_loss: 0.1
698
Epoch 5/100
17/17 [==============================] - 0s 11ms/step - loss: 0.1982 - val_loss: 0.1
680
Epoch 6/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1931 - val_loss: 0.16
60
Epoch 7/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1909 - val_loss: 0.16
44
Epoch 8/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1888 - val_loss: 0.16
51
Epoch 9/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1920 - val_loss: 0.16
42
Epoch 10/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1857 - val_loss: 0.16
28
Epoch 11/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1819 - val_loss: 0.16
20
Epoch 12/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1810 - val_loss: 0.16
35
Epoch 13/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1810 - val_loss: 0.16
24
Epoch 14/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1792 - val_loss: 0.16
24
Epoch 15/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1796 - val_loss: 0.16
19
Epoch 16/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1796 - val_loss: 0.16
16
Epoch 17/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1750 - val_loss: 0.16
14
Epoch 18/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1760 - val_loss: 0.16
16
Epoch 19/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1746 - val_loss: 0.16
```

```
09
Epoch 20/100
17/17 [==============================] - 0s 17ms/step - loss: 0.1724 - val_loss: 0.1
610
Epoch 21/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1743 - val_loss: 0.1
611
Epoch 22/100
17/17 [==============================] - 0s 15ms/step - loss: 0.1724 - val_loss: 0.1
605
Epoch 23/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1738 - val_loss: 0.16
08
Epoch 24/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1755 - val_loss: 0.16
02
Epoch 25/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1717 - val_loss: 0.16
08
Epoch 26/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1724 - val_loss: 0.16
05
Epoch 27/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1711 - val_loss: 0.16
06
Epoch 28/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1713 - val_loss: 0.16
02
Epoch 29/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1701 - val_loss: 0.16
05
Epoch 30/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1718 - val_loss: 0.16
12
Epoch 31/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1690 - val_loss: 0.16
13
Epoch 32/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1677 - val_loss: 0.16
07
Epoch 33/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1691 - val_loss: 0.16
07
Epoch 34/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1670 - val_loss: 0.16
10
Epoch 35/100
17/17 [==============================] - 0s 12ms/step - loss: 0.1688 - val_loss: 0.1
615
Epoch 36/100
17/17 [==============================] - 0s 25ms/step - loss: 0.1706 - val_loss: 0.1
613
Epoch 37/100
17/17 [==============================] - 0s 25ms/step - loss: 0.1703 - val_loss: 0.1
612
Epoch 38/100
```

```
17/17 [==============================] - 0s 25ms/step - loss: 0.1701 - val_loss: 0.1
613
Epoch 39/100
17/17 [==============================] - 0s 24ms/step - loss: 0.1679 - val_loss: 0.1
615
Epoch 40/100
17/17 [==============================] - 0s 27ms/step - loss: 0.1675 - val_loss: 0.1
612
Epoch 41/100
17/17 [==============================] - 0s 28ms/step - loss: 0.1656 - val_loss: 0.1
615
Epoch 42/100
17/17 [==============================] - 0s 23ms/step - loss: 0.1673 - val_loss: 0.1
616
Epoch 43/100
17/17 [==============================] - 0s 23ms/step - loss: 0.1675 - val_loss: 0.1
614
Epoch 44/100
17/17 [==============================] - 0s 22ms/step - loss: 0.1682 - val_loss: 0.1
625
Epoch 45/100
17/17 [==============================] - 0s 24ms/step - loss: 0.1663 - val_loss: 0.1
615
Epoch 46/100
17/17 [==============================] - 0s 23ms/step - loss: 0.1652 - val_loss: 0.1
618
Epoch 47/100
17/17 [==============================] - 0s 27ms/step - loss: 0.1658 - val_loss: 0.1
615
Epoch 48/100
17/17 [==============================] - 1s 31ms/step - loss: 0.1676 - val_loss: 0.1
620
Epoch 49/100
17/17 [==============================] - 0s 26ms/step - loss: 0.1682 - val_loss: 0.1
622
Epoch 50/100
17/17 [==============================] - 0s 27ms/step - loss: 0.1676 - val_loss: 0.1
625
Epoch 51/100
17/17 [==============================] - 0s 26ms/step - loss: 0.1642 - val_loss: 0.1
621
Epoch 52/100
17/17 [==============================] - 0s 28ms/step - loss: 0.1631 - val_loss: 0.1
622
Epoch 53/100
17/17 [==============================] - 0s 25ms/step - loss: 0.1647 - val_loss: 0.1
628
Epoch 54/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1656 - val_loss: 0.1
624
Epoch 55/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1633 - val_loss: 0.1
626
Epoch 56/100
17/17 [==============================] - 0s 21ms/step - loss: 0.1628 - val_loss: 0.1
625
```

```
Epoch 57/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1647 - val_loss: 0.1
625
Epoch 58/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1662 - val_loss: 0.1
630
Epoch 59/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1654 - val_loss: 0.1
617
Epoch 60/100
17/17 [==============================] - 0s 22ms/step - loss: 0.1619 - val_loss: 0.1
628
Epoch 61/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1638 - val_loss: 0.1
626
Epoch 62/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1632 - val_loss: 0.1
633
Epoch 63/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1654 - val_loss: 0.1
631
Epoch 64/100
17/17 [==============================] - 0s 17ms/step - loss: 0.1642 - val_loss: 0.1
629
Epoch 65/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1639 - val_loss: 0.16
33
Epoch 66/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1613 - val_loss: 0.16
22
Epoch 67/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1630 - val_loss: 0.16
31
Epoch 68/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1624 - val_loss: 0.16
28
Epoch 69/100
17/17 [==============================] - 0s 10ms/step - loss: 0.1630 - val_loss: 0.1
634
Epoch 70/100
17/17 [==============================] - 0s 9ms/step - loss: 0.1621 - val_loss: 0.16
33
Epoch 71/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1642 - val_loss: 0.16
26
Epoch 72/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1610 - val_loss: 0.16
32
Epoch 73/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1631 - val_loss: 0.16
32
Epoch 74/100
17/17 [==============================] - 0s 9ms/step - loss: 0.1612 - val_loss: 0.16
33
Epoch 75/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1640 - val_loss: 0.16
```

```
35
Epoch 76/100
17/17 [==============================] - 0s 8ms/step - loss: 0.1596 - val_loss: 0.16
34
Epoch 77/100
17/17 [==============================] - 0s 19ms/step - loss: 0.1652 - val_loss: 0.1
638
Epoch 78/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1588 - val_loss: 0.1
634
Epoch 79/100
17/17 [==============================] - 0s 19ms/step - loss: 0.1597 - val_loss: 0.1
633
Epoch 80/100
17/17 [==============================] - 0s 19ms/step - loss: 0.1609 - val_loss: 0.1
635
Epoch 81/100
17/17 [==============================] - 0s 20ms/step - loss: 0.1648 - val_loss: 0.1
637
Epoch 82/100
17/17 [==============================] - 0s 19ms/step - loss: 0.1579 - val_loss: 0.1
639
Epoch 83/100
17/17 [==============================] - 0s 16ms/step - loss: 0.1593 - val_loss: 0.1
641
Epoch 84/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1581 - val_loss: 0.16
27
Epoch 85/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1611 - val_loss: 0.16
35
Epoch 86/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1584 - val_loss: 0.16
35
Epoch 87/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1611 - val_loss: 0.16
40
Epoch 88/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1625 - val_loss: 0.16
39
Epoch 89/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1591 - val_loss: 0.16
36
Epoch 90/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1591 - val_loss: 0.16
44
Epoch 91/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1613 - val_loss: 0.16
43
Epoch 92/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1655 - val_loss: 0.16
39
Epoch 93/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1587 - val_loss: 0.16
40
Epoch 94/100
```

```
17/17 [==============================] - 0s 6ms/step - loss: 0.1612 - val_loss: 0.16
42
Epoch 95/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1610 - val_loss: 0.16
43
Epoch 96/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1601 - val_loss: 0.16
38
Epoch 97/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1591 - val_loss: 0.16
36
Epoch 98/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1573 - val_loss: 0.16
44
Epoch 99/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1583 - val_loss: 0.16
41
Epoch 100/100
17/17 [==============================] - 0s 18ms/step - loss: 0.1594 - val_loss: 0.1
642
```

Out[20]: `<keras.src.callbacks.History at 0x7fa4ac944af0>`

Ahora, pasamos los datos por la parte codificadora de la red neuronal:

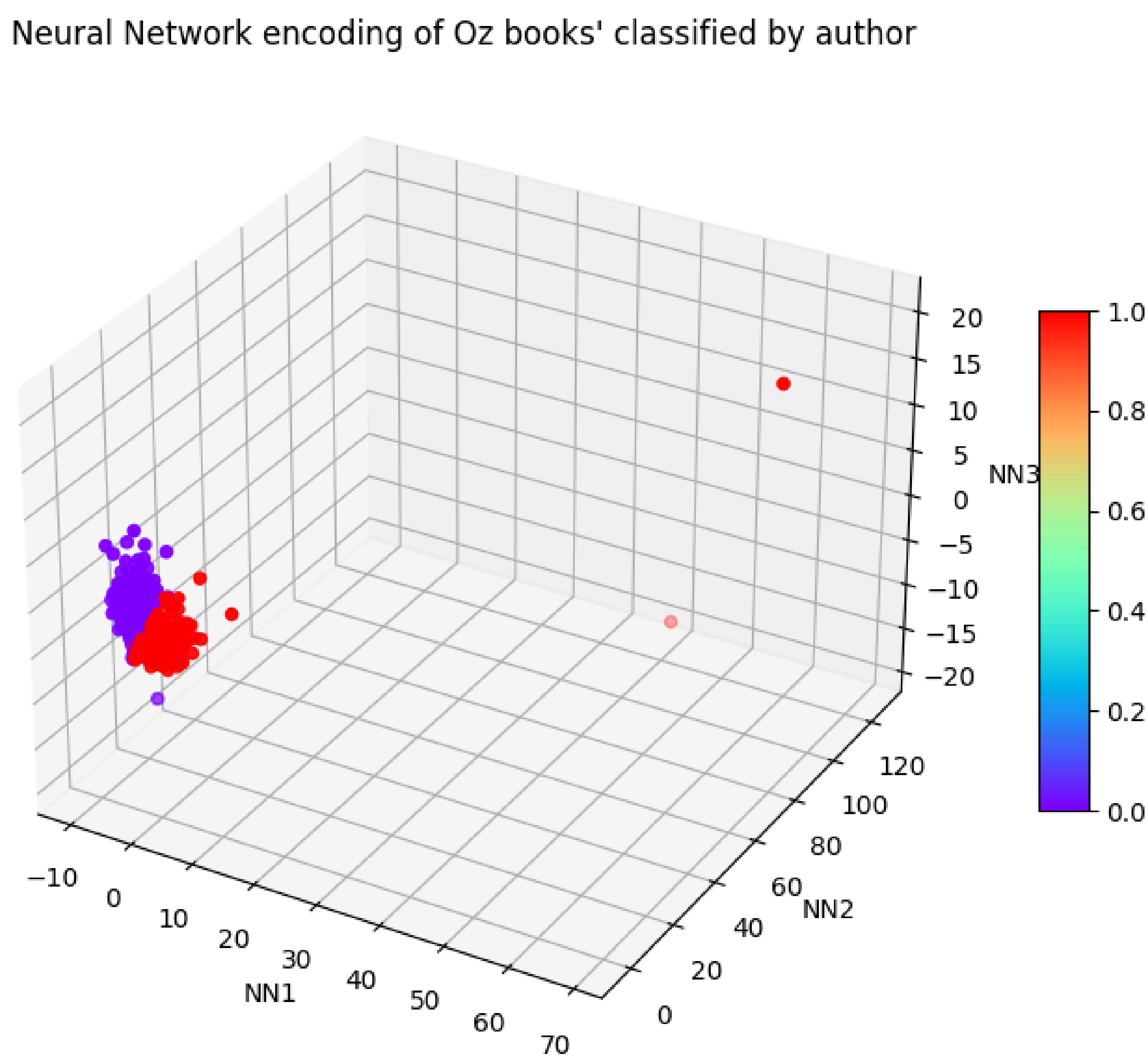
In [ ]:

```python
# Get 3D encoded data
X_encoded = encoder.predict(X_scaled)
```

```
19/19 [==============================] - 0s 2ms/step
```

In [22]:

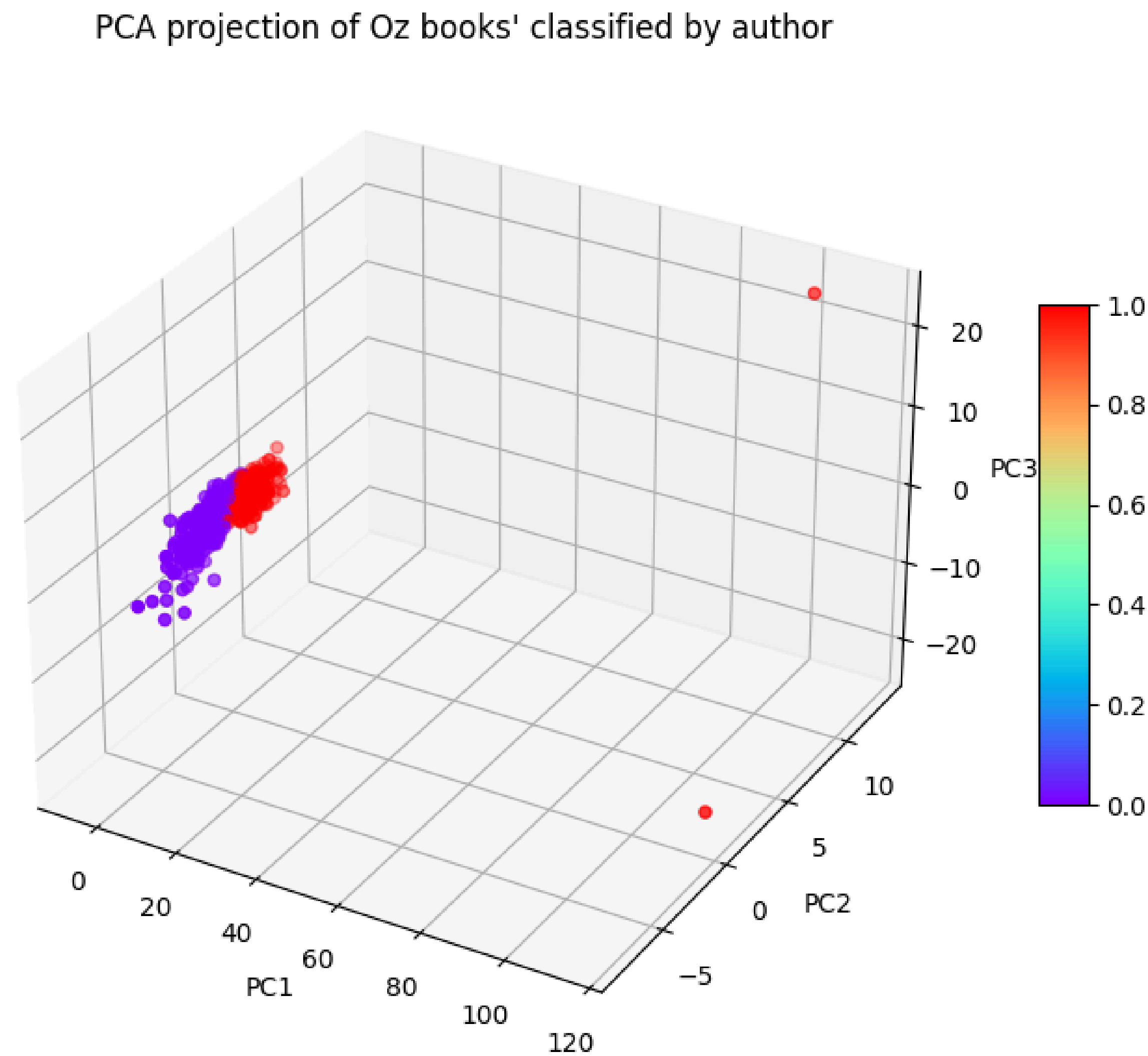
```python
# Create a 3D scatter plot of the data in the encoder dimensions
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Plot the 3D encoded points, colored by class labels `y`
sc = ax.scatter(X_encoded[:, 0], X_encoded[:, 1], X_encoded[:, 2], c=y, cmap='rainb

# Add axis labels
ax.set_xlabel('NN1')
ax.set_ylabel('NN2')
ax.set_zlabel('NN3')

# Add a title
ax.set_title("Neural Network encoding of Oz books' classified by author")

# Add color bar if desired
plt.colorbar(sc, ax=ax, shrink=0.5, aspect=10)

# Display the plot
plt.show()
```



In [23]:
```
# Perform Principal Component Analysis (PCA) to reduce data to 2 dimensions
pca = PCA(n_components=3)
pca_data = pca.fit_transform(X_scaled)
```

In [24]:
```
# Create a 3D scatter plot of the data in the encoder dimensions
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Plot the 3D encoded points, colored by class labels `y`
sc = ax.scatter(pca_data[:, 0], pca_data[:, 1], pca_data[:, 2], c=y, cmap='rainbow'

# Add axis labels
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')

# Add a title
ax.set_title('PCA projection of Oz books\' classified by author')

# Add color bar if desired
plt.colorbar(sc, ax=ax, shrink=0.5, aspect=10)
```

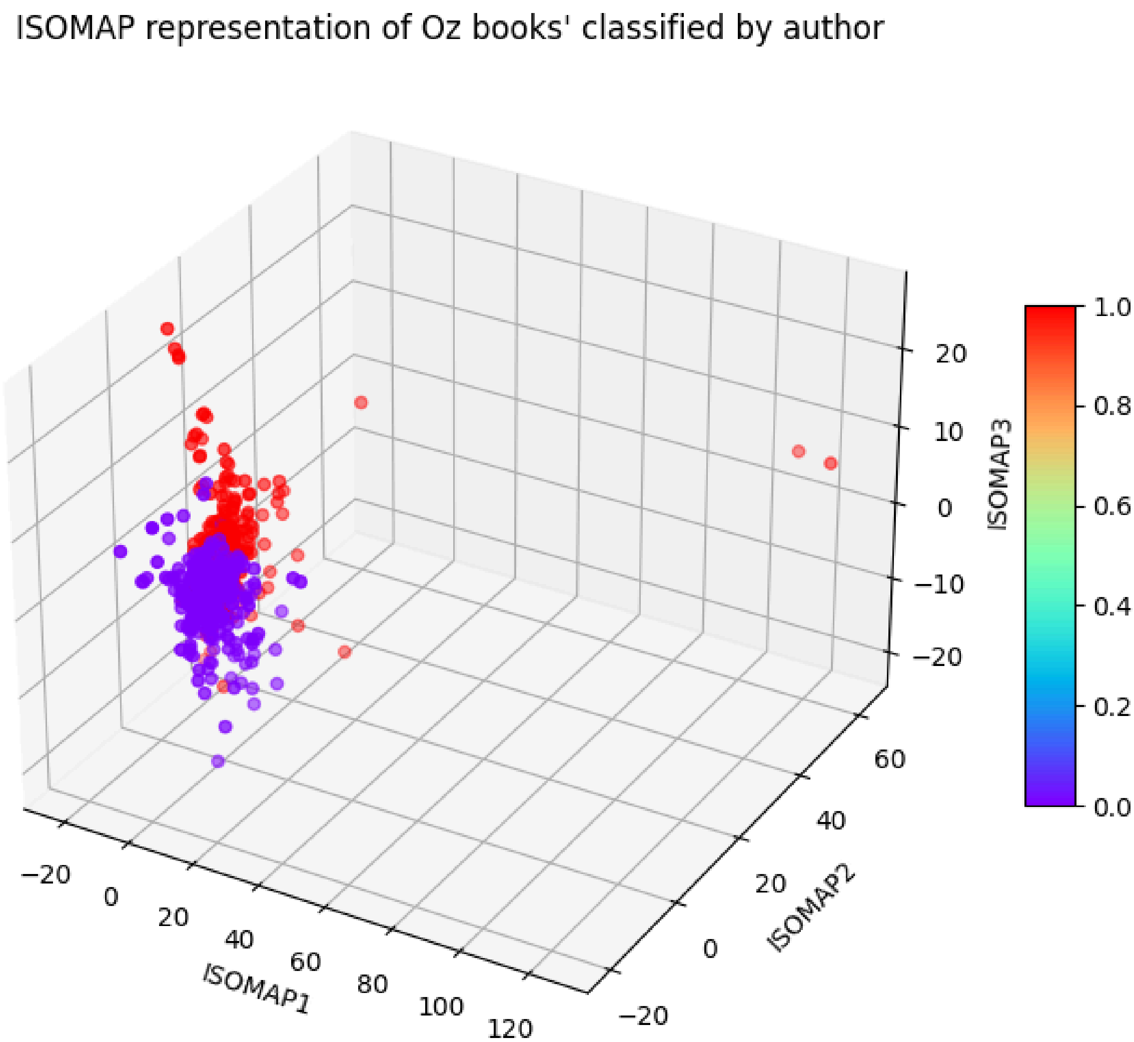
```python
# Display the plot
plt.show()
```



In [25]:
```python
iso = manifold.Isomap(n_components=3)
X_iso = iso.fit_transform(X_scaled)
```

In [26]:
```python
# Create a 3D scatter plot of the data in the encoder dimensions
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Plot the 3D encoded points, colored by class labels `y`
sc = ax.scatter(X_iso[:, 0], X_iso[:, 1], X_iso[:, 2], c=y, cmap='rainbow')

# Add axis labels
ax.set_xlabel('ISOMAP1')
ax.set_ylabel('ISOMAP2')
ax.set_zlabel('ISOMAP3')

# Add a title
ax.set_title('ISOMAP representation of Oz books\' classified by author')

# Add color bar if desired
plt.colorbar(sc, ax=ax, shrink=0.5, aspect=10)
```

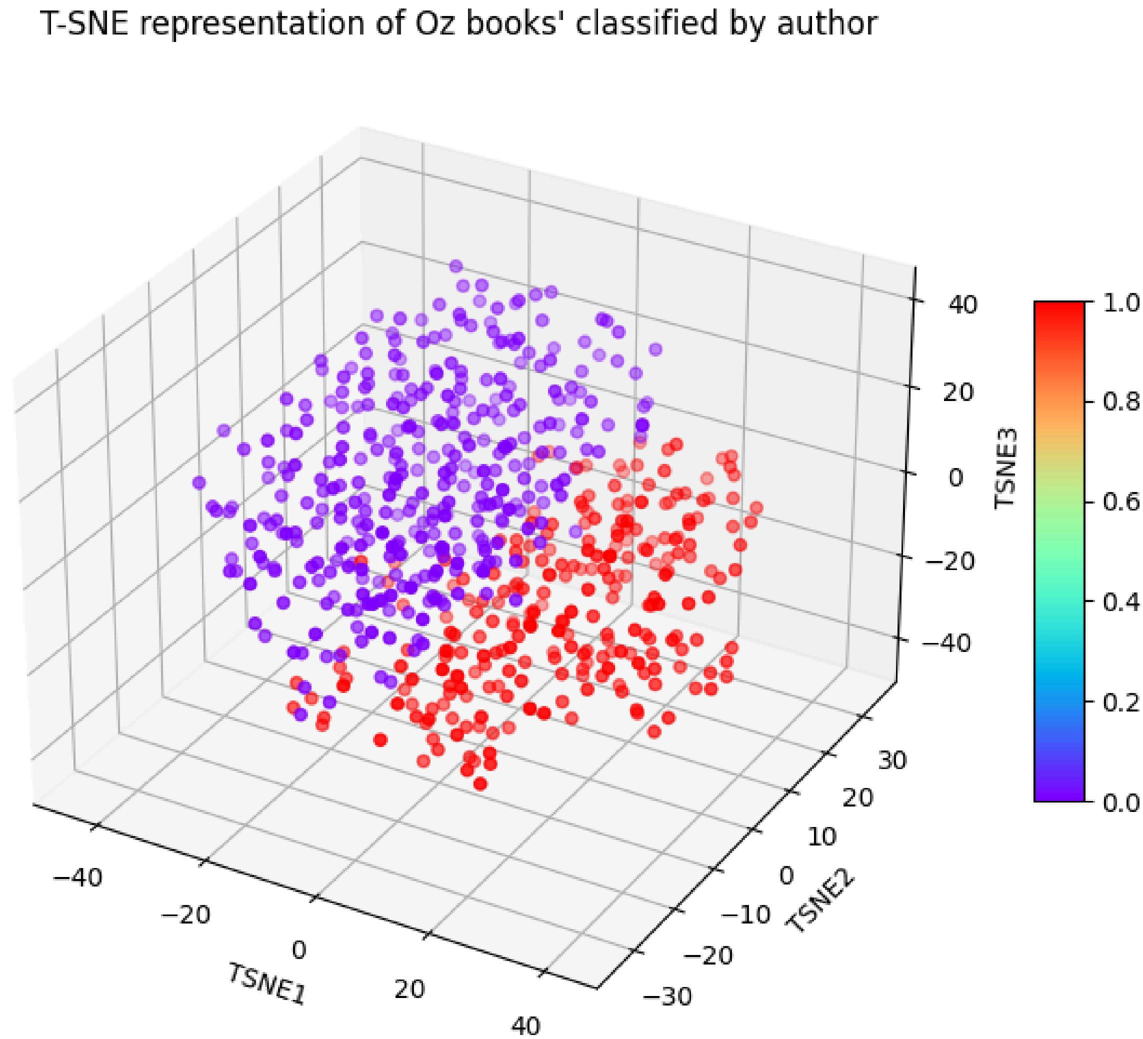
```python
# Display the plot
plt.show()
```



In [27]:
```python
tsne = manifold.TSNE(n_components=3, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)
```

In [28]:
```python
# Create a 3D scatter plot of the data in the encoder dimensions
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Plot the 3D encoded points, colored by class labels `y`
sc = ax.scatter(X_tsne[:, 0], X_tsne[:, 1], X_tsne[:, 2], c=y, cmap='rainbow')

# Add axis labels
ax.set_xlabel('TSNE1')
ax.set_ylabel('TSNE2')
ax.set_zlabel('TSNE3')

# Add a title
ax.set_title('T-SNE representation of Oz books\' classified by author')

# Add color bar if desired
```

```
plt.colorbar(sc, ax=ax, shrink=0.5, aspect=10)

# Display the plot
plt.show()
```



Aquí se repite lo anterior, los datos son casi separables en todos los casos, el mejor siendo T-SNE.