Centro de Investigación en Matemáticas, A.C.

Reconocimiento de Patrones

Tarea 6. Ejercicio 2. Inciso b)

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]   Unzipping corpora/stopwords.zip.
```

Out[2]: True

Construir Bag of Word a partir de textos

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 process_scripts(directory, k, language):
    """
    Processes a directory of text files, applying the preprocessing steps,
    retaining only the k most common words across all texts, and returning
    a CountVectorizer object.

    Args:
        directory: The path to the directory containing the text files.
        k: Number of most common words to retain.

    Returns:
        A tuple (X, vectorizer) where X is the matrix representation of the
        processed texts, and vectorizer is the CountVectorizer used.
    """
    all_texts = []
    seasons = []
    episodes = []

    stemmer = SnowballStemmer(language)  # Or "spanish" if your texts are in Spanis
    stop_words = set(stopwords.words(language))

    # Collect all texts and filenames
    for filename in os.listdir(directory):
        if filename.endswith(".txt"):  # Process only .txt files
            filepath = os.path.join(directory, filename)
            with open(filepath, 'r', encoding='utf-8-sig') as f:
                text = f.read()
                processed_text = preprocess(text, stemmer, stop_words)
                all_texts.append(processed_text)    # Append original processed tex
                seasons.append(filename[1:3])     # Save the season number
                episodes.append(filename[4:6])  # Save the episode number

    # Flatten list of lists to a single list and count word frequencies
    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)]
    vectorizer = CountVectorizer(min_df=1, vocabulary=most_common_words)
    X = vectorizer.fit_transform(all_texts)

    return X, vectorizer, seasons, episodes
```

In [5]:
```
X, vectorizer, seasons, episodes = process_scripts ('./simpsons_scripts',150,"engli
```

In [6]:
```
X_dense = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())
df_episodes = pd.DataFrame({"season": seasons, "episode": episodes})
df_episodes['era'] = df_episodes.apply(lambda row: "golden age" if int(row["season"

# Merge chapter information
df_simpsons_full = pd.concat([X_dense, df_episodes], axis=1)
```

In [7]:
```
# 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.
y = label_encoder.fit_transform(df_episodes['era'])
```

In [8]:
```
# Scale the data to have zero mean and unit variance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_dense)
```

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(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, 150)]             0

 dense (Dense)               (None, 64)                9664

 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, 150)               9750

=================================================================
Total params: 23768 (92.84 KB)
Trainable params: 23768 (92.84 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

```
Epoch 1/100
22/22 [==============================] - 3s 12ms/step - loss: 0.3822 - val_loss: 0.3
757
Epoch 2/100
22/22 [==============================] - 0s 4ms/step - loss: 0.3752 - val_loss: 0.36
62
Epoch 3/100
22/22 [==============================] - 0s 4ms/step - loss: 0.3687 - val_loss: 0.36
31
Epoch 4/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3652 - val_loss: 0.36
07
Epoch 5/100
22/22 [==============================] - 0s 4ms/step - loss: 0.3613 - val_loss: 0.35
74
Epoch 6/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3579 - val_loss: 0.35
38
Epoch 7/100
22/22 [==============================] - 0s 4ms/step - loss: 0.3556 - val_loss: 0.35
19
Epoch 8/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3540 - val_loss: 0.35
04
Epoch 9/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3530 - val_loss: 0.34
96
Epoch 10/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3523 - val_loss: 0.34
92
Epoch 11/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3517 - val_loss: 0.34
87
Epoch 12/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3512 - val_loss: 0.34
81
Epoch 13/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3508 - val_loss: 0.34
78
Epoch 14/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3504 - val_loss: 0.34
77
Epoch 15/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3500 - val_loss: 0.34
74
Epoch 16/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3496 - val_loss: 0.34
74
Epoch 17/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3493 - val_loss: 0.34
70
Epoch 18/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3490 - val_loss: 0.34
70
Epoch 19/100
22/22 [==============================] - 0s 4ms/step - loss: 0.3486 - val_loss: 0.34
```

```
69
Epoch 20/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3484 - val_loss: 0.34
65
Epoch 21/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3481 - val_loss: 0.34
63
Epoch 22/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3478 - val_loss: 0.34
65
Epoch 23/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3475 - val_loss: 0.34
65
Epoch 24/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3472 - val_loss: 0.34
62
Epoch 25/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3468 - val_loss: 0.34
68
Epoch 26/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3466 - val_loss: 0.34
65
Epoch 27/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3462 - val_loss: 0.34
67
Epoch 28/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3459 - val_loss: 0.34
68
Epoch 29/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3456 - val_loss: 0.34
71
Epoch 30/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3453 - val_loss: 0.34
68
Epoch 31/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3450 - val_loss: 0.34
72
Epoch 32/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3448 - val_loss: 0.34
71
Epoch 33/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3445 - val_loss: 0.34
71
Epoch 34/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3442 - val_loss: 0.34
77
Epoch 35/100
22/22 [==============================] - 0s 10ms/step - loss: 0.3441 - val_loss: 0.3
479
Epoch 36/100
22/22 [==============================] - 0s 11ms/step - loss: 0.3438 - val_loss: 0.3
479
Epoch 37/100
22/22 [==============================] - 0s 10ms/step - loss: 0.3437 - val_loss: 0.3
476
Epoch 38/100
```

```
22/22 [==============================] - 0s 6ms/step - loss: 0.3435 - val_loss: 0.34
81
Epoch 39/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3433 - val_loss: 0.34
78
Epoch 40/100
22/22 [==============================] - 0s 10ms/step - loss: 0.3430 - val_loss: 0.3
480
Epoch 41/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3427 - val_loss: 0.34
79
Epoch 42/100
22/22 [==============================] - 0s 9ms/step - loss: 0.3427 - val_loss: 0.34
79
Epoch 43/100
22/22 [==============================] - 0s 12ms/step - loss: 0.3424 - val_loss: 0.3
484
Epoch 44/100
22/22 [==============================] - 0s 12ms/step - loss: 0.3422 - val_loss: 0.3
484
Epoch 45/100
22/22 [==============================] - 0s 11ms/step - loss: 0.3419 - val_loss: 0.3
483
Epoch 46/100
22/22 [==============================] - 0s 9ms/step - loss: 0.3417 - val_loss: 0.34
87
Epoch 47/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3416 - val_loss: 0.34
85
Epoch 48/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3415 - val_loss: 0.34
89
Epoch 49/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3413 - val_loss: 0.34
82
Epoch 50/100
22/22 [==============================] - 0s 13ms/step - loss: 0.3412 - val_loss: 0.3
487
Epoch 51/100
22/22 [==============================] - 0s 9ms/step - loss: 0.3410 - val_loss: 0.34
86
Epoch 52/100
22/22 [==============================] - 0s 16ms/step - loss: 0.3408 - val_loss: 0.3
480
Epoch 53/100
22/22 [==============================] - 0s 17ms/step - loss: 0.3407 - val_loss: 0.3
488
Epoch 54/100
22/22 [==============================] - 0s 20ms/step - loss: 0.3407 - val_loss: 0.3
488
Epoch 55/100
22/22 [==============================] - 0s 20ms/step - loss: 0.3405 - val_loss: 0.3
488
Epoch 56/100
22/22 [==============================] - 1s 26ms/step - loss: 0.3402 - val_loss: 0.3
481
```

```
Epoch 57/100
22/22 [==============================] - 0s 21ms/step - loss: 0.3400 - val_loss: 0.3
485
Epoch 58/100
22/22 [==============================] - 0s 21ms/step - loss: 0.3399 - val_loss: 0.3
483
Epoch 59/100
22/22 [==============================] - 0s 20ms/step - loss: 0.3397 - val_loss: 0.3
485
Epoch 60/100
22/22 [==============================] - 0s 17ms/step - loss: 0.3395 - val_loss: 0.3
485
Epoch 61/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3395 - val_loss: 0.34
83
Epoch 62/100
22/22 [==============================] - 0s 12ms/step - loss: 0.3394 - val_loss: 0.3
485
Epoch 63/100
22/22 [==============================] - 0s 8ms/step - loss: 0.3392 - val_loss: 0.34
86
Epoch 64/100
22/22 [==============================] - 0s 9ms/step - loss: 0.3391 - val_loss: 0.34
86
Epoch 65/100
22/22 [==============================] - 0s 10ms/step - loss: 0.3389 - val_loss: 0.3
494
Epoch 66/100
22/22 [==============================] - 0s 10ms/step - loss: 0.3388 - val_loss: 0.3
489
Epoch 67/100
22/22 [==============================] - 0s 10ms/step - loss: 0.3388 - val_loss: 0.3
479
Epoch 68/100
22/22 [==============================] - 0s 12ms/step - loss: 0.3386 - val_loss: 0.3
484
Epoch 69/100
22/22 [==============================] - 0s 20ms/step - loss: 0.3384 - val_loss: 0.3
484
Epoch 70/100
22/22 [==============================] - 0s 21ms/step - loss: 0.3383 - val_loss: 0.3
487
Epoch 71/100
22/22 [==============================] - 0s 20ms/step - loss: 0.3381 - val_loss: 0.3
491
Epoch 72/100
22/22 [==============================] - 0s 20ms/step - loss: 0.3381 - val_loss: 0.3
493
Epoch 73/100
22/22 [==============================] - 0s 21ms/step - loss: 0.3379 - val_loss: 0.3
488
Epoch 74/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3379 - val_loss: 0.3
492
Epoch 75/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3377 - val_loss: 0.3
```

```
490
Epoch 76/100
22/22 [==============================] - 0s 20ms/step - loss: 0.3377 - val_loss: 0.3
490
Epoch 77/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3375 - val_loss: 0.3
493
Epoch 78/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3374 - val_loss: 0.3
493
Epoch 79/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3373 - val_loss: 0.3
496
Epoch 80/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3372 - val_loss: 0.3
499
Epoch 81/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3370 - val_loss: 0.3
498
Epoch 82/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3371 - val_loss: 0.3
493
Epoch 83/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3370 - val_loss: 0.3
499
Epoch 84/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3369 - val_loss: 0.3
494
Epoch 85/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3367 - val_loss: 0.3
493
Epoch 86/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3367 - val_loss: 0.3
494
Epoch 87/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3365 - val_loss: 0.3
494
Epoch 88/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3364 - val_loss: 0.3
493
Epoch 89/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3365 - val_loss: 0.3
497
Epoch 90/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3365 - val_loss: 0.3
500
Epoch 91/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3364 - val_loss: 0.3
494
Epoch 92/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3362 - val_loss: 0.3
497
Epoch 93/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3361 - val_loss: 0.3
491
Epoch 94/100
```

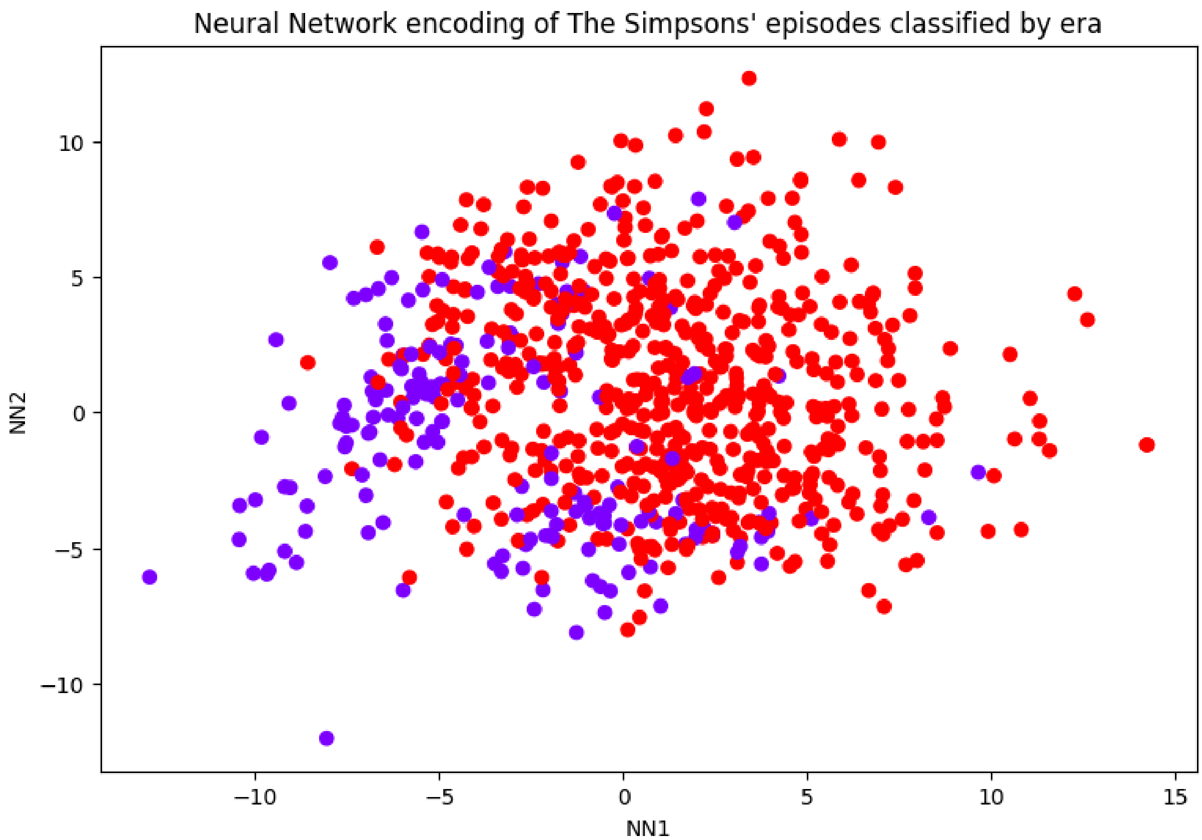
```
22/22 [==============================] - 0s 19ms/step - loss: 0.3360 - val_loss: 0.3
498
Epoch 95/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3359 - val_loss: 0.3
501
Epoch 96/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3357 - val_loss: 0.3
496
Epoch 97/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3357 - val_loss: 0.3
504
Epoch 98/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3356 - val_loss: 0.3
499
Epoch 99/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3353 - val_loss: 0.3
503
Epoch 100/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3352 - val_loss: 0.3
497
```

Out[10]:
```
<keras.src.callbacks.History at 0x7f5f4d1271f0>
```

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

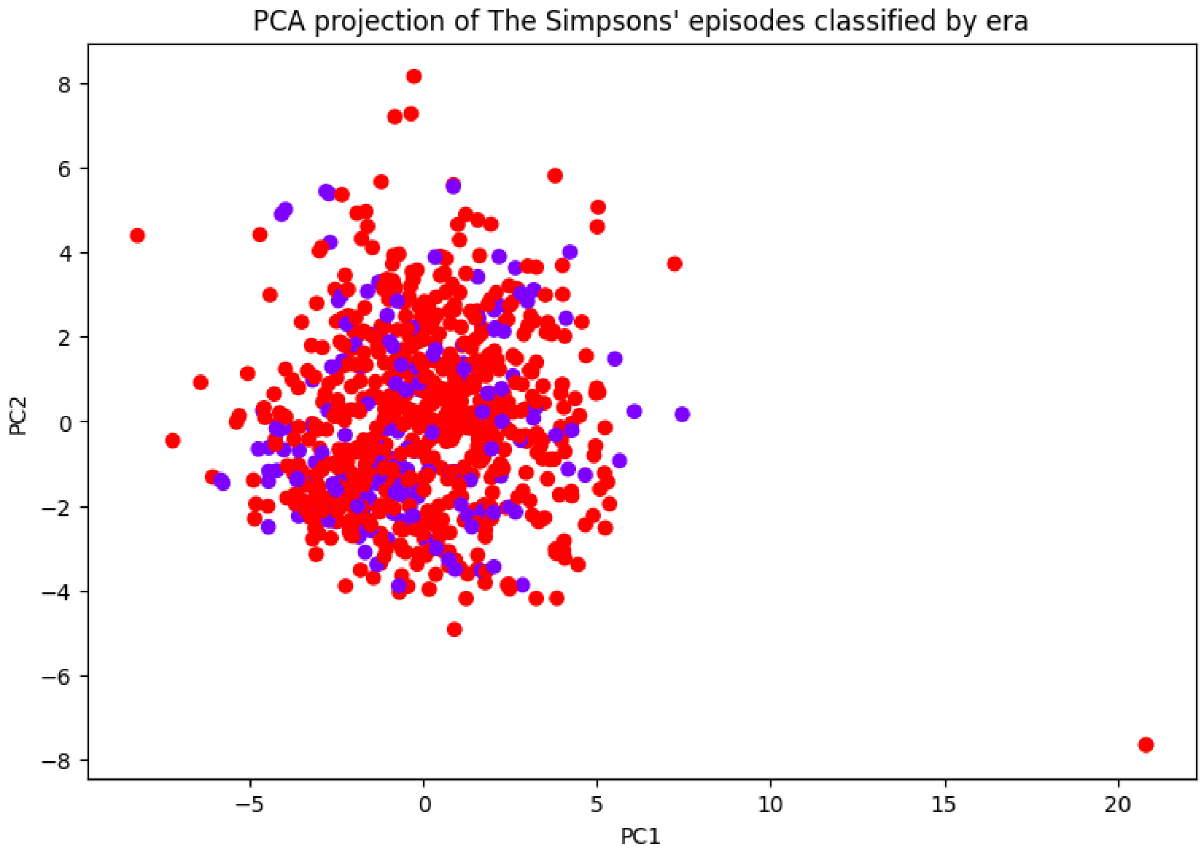
```
25/25 [==============================] - 0s 3ms/step
```

In [29]:
```
# 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 The Simpsons' episodes classified by era")
# 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()
```



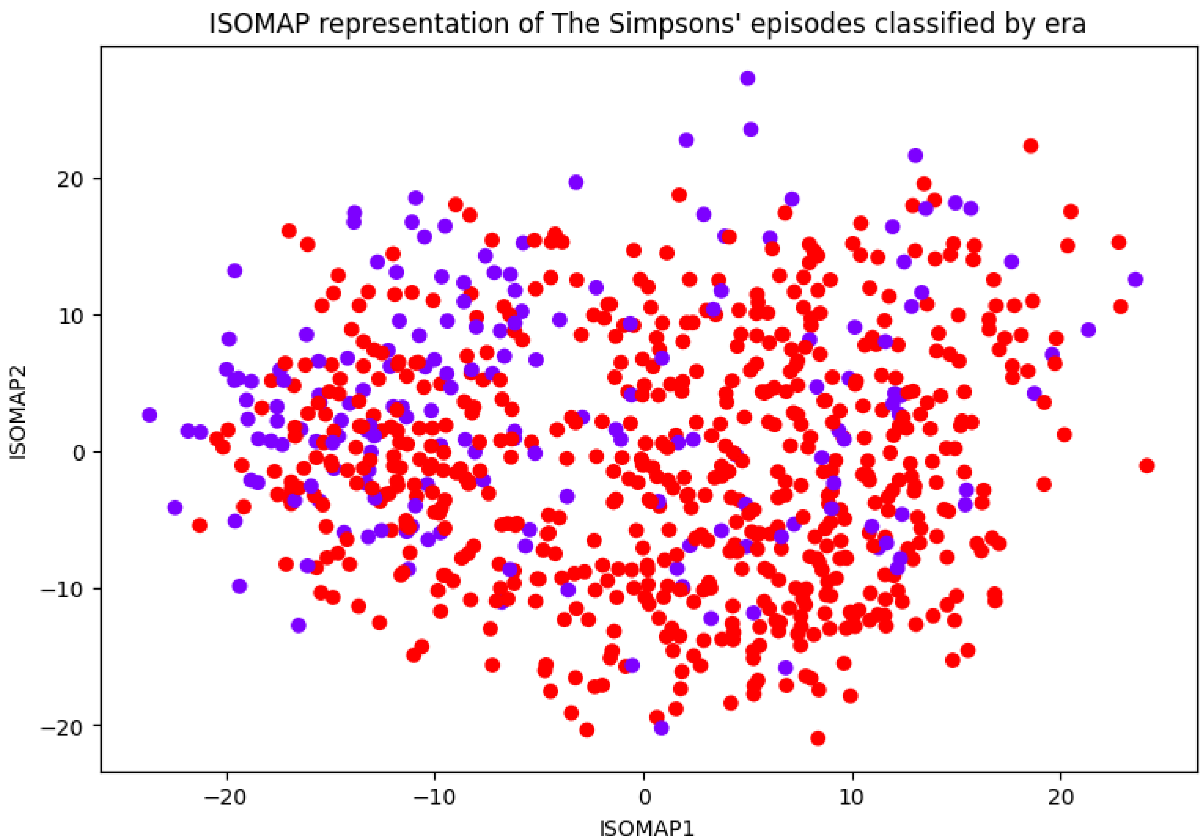
In [13]:
```
# Perform Principal Component Analysis (PCA) to reduce data to 2 dimensions
pca = PCA(n_components=2)
pca_data = pca.fit_transform(X_scaled)
```

In [30]:
```
# 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 The Simpsons' episodes classified by era")
# 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()
```



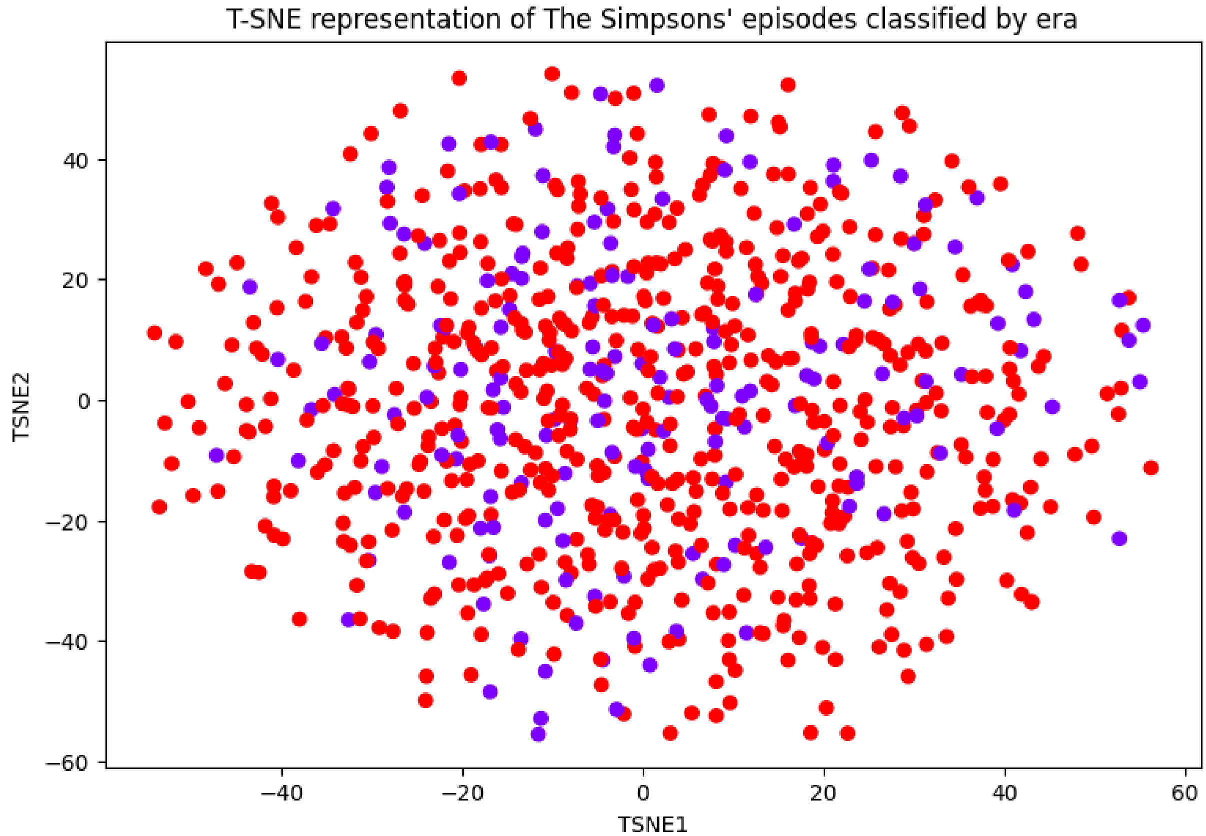
In [15]:
```
iso = manifold.Isomap(n_components=2)
X_iso = iso.fit_transform(X_scaled)
```

In [31]:
```
# 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 The Simpsons' episodes classified by era")
# 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 [32]:
```
# 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 The Simpsons' episodes classified by era")
# 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 dos dimensiones, parece que el método que mejor separa es el autoencoder. Sin embargo, ningún en método se distingue que los datos sean linealmente separables. A continuación, seguimos con tres dimensiones:

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")

# 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"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 150)]             0         
                                                                 
 dense_6 (Dense)             (None, 64)                9664      
                                                                 
 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, 150)               9750      
                                                                 
=================================================================
Total params: 23833 (93.10 KB)
Trainable params: 23833 (93.10 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

```
Epoch 1/100
22/22 [==============================] - 2s 20ms/step - loss: 0.3825 - val_loss: 0.3
771
Epoch 2/100
22/22 [==============================] - 0s 16ms/step - loss: 0.3776 - val_loss: 0.3
722
Epoch 3/100
22/22 [==============================] - 0s 16ms/step - loss: 0.3716 - val_loss: 0.3
658
Epoch 4/100
22/22 [==============================] - 0s 14ms/step - loss: 0.3656 - val_loss: 0.3
607
Epoch 5/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3601 - val_loss: 0.35
57
Epoch 6/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3554 - val_loss: 0.35
16
Epoch 7/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3519 - val_loss: 0.34
90
Epoch 8/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3495 - val_loss: 0.34
74
Epoch 9/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3478 - val_loss: 0.34
58
Epoch 10/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3464 - val_loss: 0.34
48
Epoch 11/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3454 - val_loss: 0.34
45
Epoch 12/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3445 - val_loss: 0.34
32
Epoch 13/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3438 - val_loss: 0.34
29
Epoch 14/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3432 - val_loss: 0.34
23
Epoch 15/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3426 - val_loss: 0.34
24
Epoch 16/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3422 - val_loss: 0.34
21
Epoch 17/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3417 - val_loss: 0.34
18
Epoch 18/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3413 - val_loss: 0.34
15
Epoch 19/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3409 - val_loss: 0.34
```

```
14
Epoch 20/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3405 - val_loss: 0.34
15
Epoch 21/100
22/22 [==============================] - 0s 11ms/step - loss: 0.3402 - val_loss: 0.3
414
Epoch 22/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3398 - val_loss: 0.34
14
Epoch 23/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3395 - val_loss: 0.34
12
Epoch 24/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3392 - val_loss: 0.34
07
Epoch 25/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3389 - val_loss: 0.34
07
Epoch 26/100
22/22 [==============================] - 0s 4ms/step - loss: 0.3386 - val_loss: 0.34
09
Epoch 27/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3383 - val_loss: 0.34
05
Epoch 28/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3380 - val_loss: 0.34
04
Epoch 29/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3378 - val_loss: 0.34
07
Epoch 30/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3375 - val_loss: 0.34
07
Epoch 31/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3372 - val_loss: 0.34
06
Epoch 32/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3370 - val_loss: 0.34
08
Epoch 33/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3367 - val_loss: 0.34
08
Epoch 34/100
22/22 [==============================] - 0s 8ms/step - loss: 0.3364 - val_loss: 0.34
05
Epoch 35/100
22/22 [==============================] - 0s 11ms/step - loss: 0.3362 - val_loss: 0.3
409
Epoch 36/100
22/22 [==============================] - 0s 20ms/step - loss: 0.3359 - val_loss: 0.3
405
Epoch 37/100
22/22 [==============================] - 0s 21ms/step - loss: 0.3356 - val_loss: 0.3
407
Epoch 38/100
```

```
22/22 [==============================] - 0s 18ms/step - loss: 0.3354 - val_loss: 0.3
410
Epoch 39/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3352 - val_loss: 0.3
415
Epoch 40/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3349 - val_loss: 0.3
410
Epoch 41/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3347 - val_loss: 0.3
411
Epoch 42/100
22/22 [==============================] - 0s 14ms/step - loss: 0.3344 - val_loss: 0.3
412
Epoch 43/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3342 - val_loss: 0.34
10
Epoch 44/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3339 - val_loss: 0.34
11
Epoch 45/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3337 - val_loss: 0.34
10
Epoch 46/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3335 - val_loss: 0.34
21
Epoch 47/100
22/22 [==============================] - 0s 8ms/step - loss: 0.3334 - val_loss: 0.34
17
Epoch 48/100
22/22 [==============================] - 0s 9ms/step - loss: 0.3330 - val_loss: 0.34
12
Epoch 49/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3327 - val_loss: 0.34
20
Epoch 50/100
22/22 [==============================] - 0s 9ms/step - loss: 0.3324 - val_loss: 0.34
20
Epoch 51/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3323 - val_loss: 0.34
21
Epoch 52/100
22/22 [==============================] - 0s 9ms/step - loss: 0.3321 - val_loss: 0.34
23
Epoch 53/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3319 - val_loss: 0.3
416
Epoch 54/100
22/22 [==============================] - 0s 17ms/step - loss: 0.3316 - val_loss: 0.3
421
Epoch 55/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3313 - val_loss: 0.3
416
Epoch 56/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3310 - val_loss: 0.3
418
```

```
Epoch 57/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3307 - val_loss: 0.3
416
Epoch 58/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3305 - val_loss: 0.3
424
Epoch 59/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3304 - val_loss: 0.3
422
Epoch 60/100
22/22 [==============================] - 0s 22ms/step - loss: 0.3302 - val_loss: 0.3
428
Epoch 61/100
22/22 [==============================] - 0s 20ms/step - loss: 0.3300 - val_loss: 0.3
429
Epoch 62/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3297 - val_loss: 0.3
436
Epoch 63/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3298 - val_loss: 0.3
435
Epoch 64/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3295 - val_loss: 0.3
428
Epoch 65/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3292 - val_loss: 0.3
432
Epoch 66/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3290 - val_loss: 0.3
428
Epoch 67/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3286 - val_loss: 0.3
428
Epoch 68/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3284 - val_loss: 0.3
433
Epoch 69/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3283 - val_loss: 0.3
439
Epoch 70/100
22/22 [==============================] - 0s 10ms/step - loss: 0.3280 - val_loss: 0.3
431
Epoch 71/100
22/22 [==============================] - 0s 10ms/step - loss: 0.3279 - val_loss: 0.3
440
Epoch 72/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3277 - val_loss: 0.34
39
Epoch 73/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3275 - val_loss: 0.34
60
Epoch 74/100
22/22 [==============================] - 0s 7ms/step - loss: 0.3278 - val_loss: 0.34
42
Epoch 75/100
22/22 [==============================] - 0s 8ms/step - loss: 0.3273 - val_loss: 0.34
```

```
36
Epoch 76/100
22/22 [==============================] - 0s 8ms/step - loss: 0.3268 - val_loss: 0.34
36
Epoch 77/100
22/22 [==============================] - 0s 8ms/step - loss: 0.3265 - val_loss: 0.34
34
Epoch 78/100
22/22 [==============================] - 0s 9ms/step - loss: 0.3264 - val_loss: 0.34
34
Epoch 79/100
22/22 [==============================] - 0s 14ms/step - loss: 0.3262 - val_loss: 0.3
443
Epoch 80/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3261 - val_loss: 0.3
440
Epoch 81/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3259 - val_loss: 0.3
441
Epoch 82/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3258 - val_loss: 0.3
443
Epoch 83/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3257 - val_loss: 0.3
438
Epoch 84/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3256 - val_loss: 0.3
439
Epoch 85/100
22/22 [==============================] - 0s 17ms/step - loss: 0.3254 - val_loss: 0.3
443
Epoch 86/100
22/22 [==============================] - 0s 17ms/step - loss: 0.3251 - val_loss: 0.3
448
Epoch 87/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3249 - val_loss: 0.3
445
Epoch 88/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3248 - val_loss: 0.3
442
Epoch 89/100
22/22 [==============================] - 0s 18ms/step - loss: 0.3245 - val_loss: 0.3
441
Epoch 90/100
22/22 [==============================] - 0s 17ms/step - loss: 0.3243 - val_loss: 0.3
438
Epoch 91/100
22/22 [==============================] - 0s 19ms/step - loss: 0.3241 - val_loss: 0.3
445
Epoch 92/100
22/22 [==============================] - 0s 17ms/step - loss: 0.3238 - val_loss: 0.3
439
Epoch 93/100
22/22 [==============================] - 0s 17ms/step - loss: 0.3237 - val_loss: 0.3
447
Epoch 94/100
```

```
22/22 [==============================] - 0s 13ms/step - loss: 0.3236 - val_loss: 0.3
442
Epoch 95/100
22/22 [==============================] - 0s 5ms/step - loss: 0.3234 - val_loss: 0.34
41
Epoch 96/100
22/22 [==============================] - 0s 6ms/step - loss: 0.3233 - val_loss: 0.34
44
Epoch 97/100
22/22 [==============================] - 0s 8ms/step - loss: 0.3231 - val_loss: 0.34
51
Epoch 98/100
22/22 [==============================] - 0s 8ms/step - loss: 0.3231 - val_loss: 0.34
52
Epoch 99/100
22/22 [==============================] - 0s 8ms/step - loss: 0.3229 - val_loss: 0.34
46
Epoch 100/100
22/22 [==============================] - 0s 8ms/step - loss: 0.3229 - val_loss: 0.34
69
```

Out[20]: <keras.src.callbacks.History at 0x7f5f0d9dc0a0>

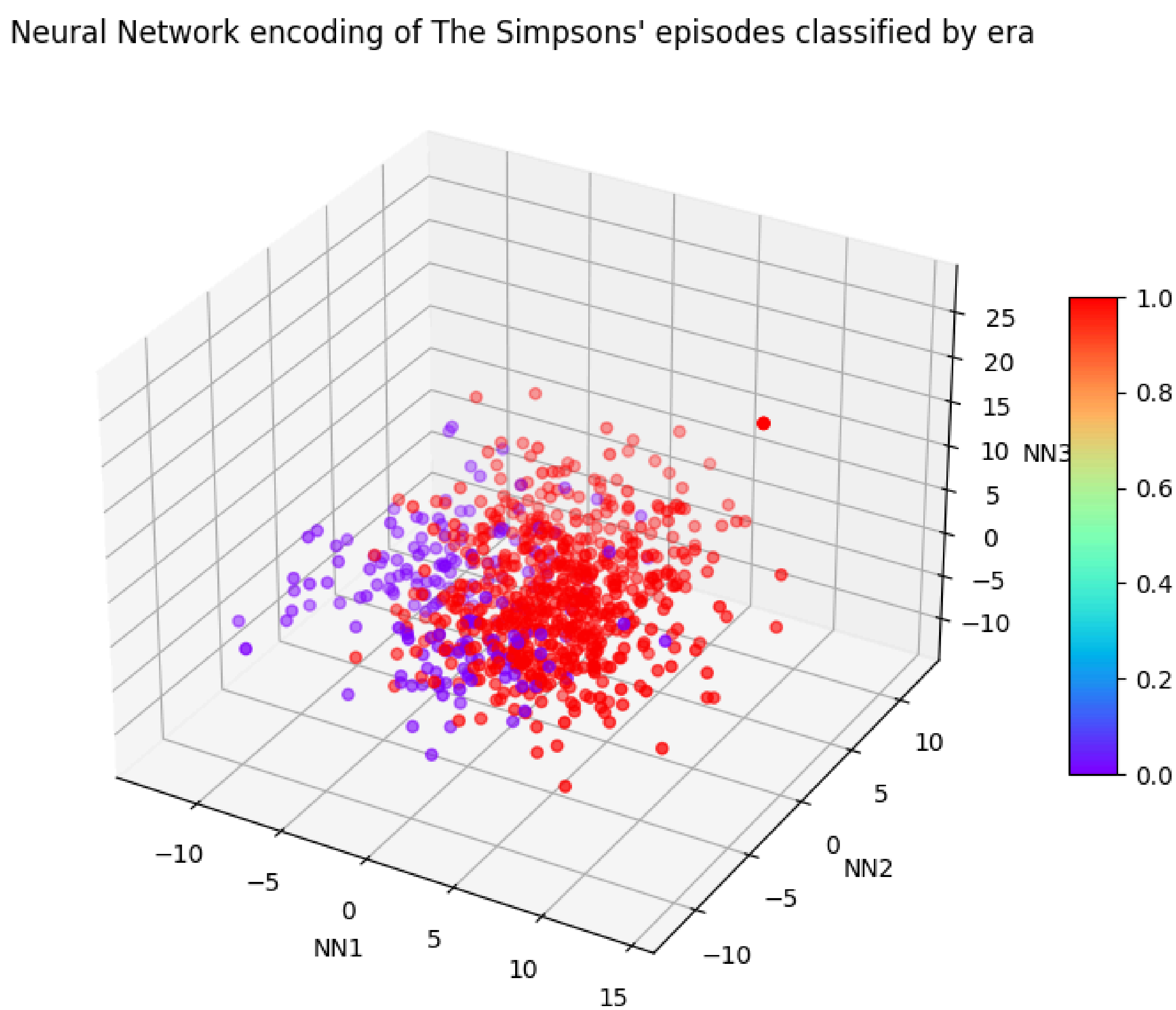
In [ ]:

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

```
25/25 [==============================] - 0s 2ms/step
```

In [22]:

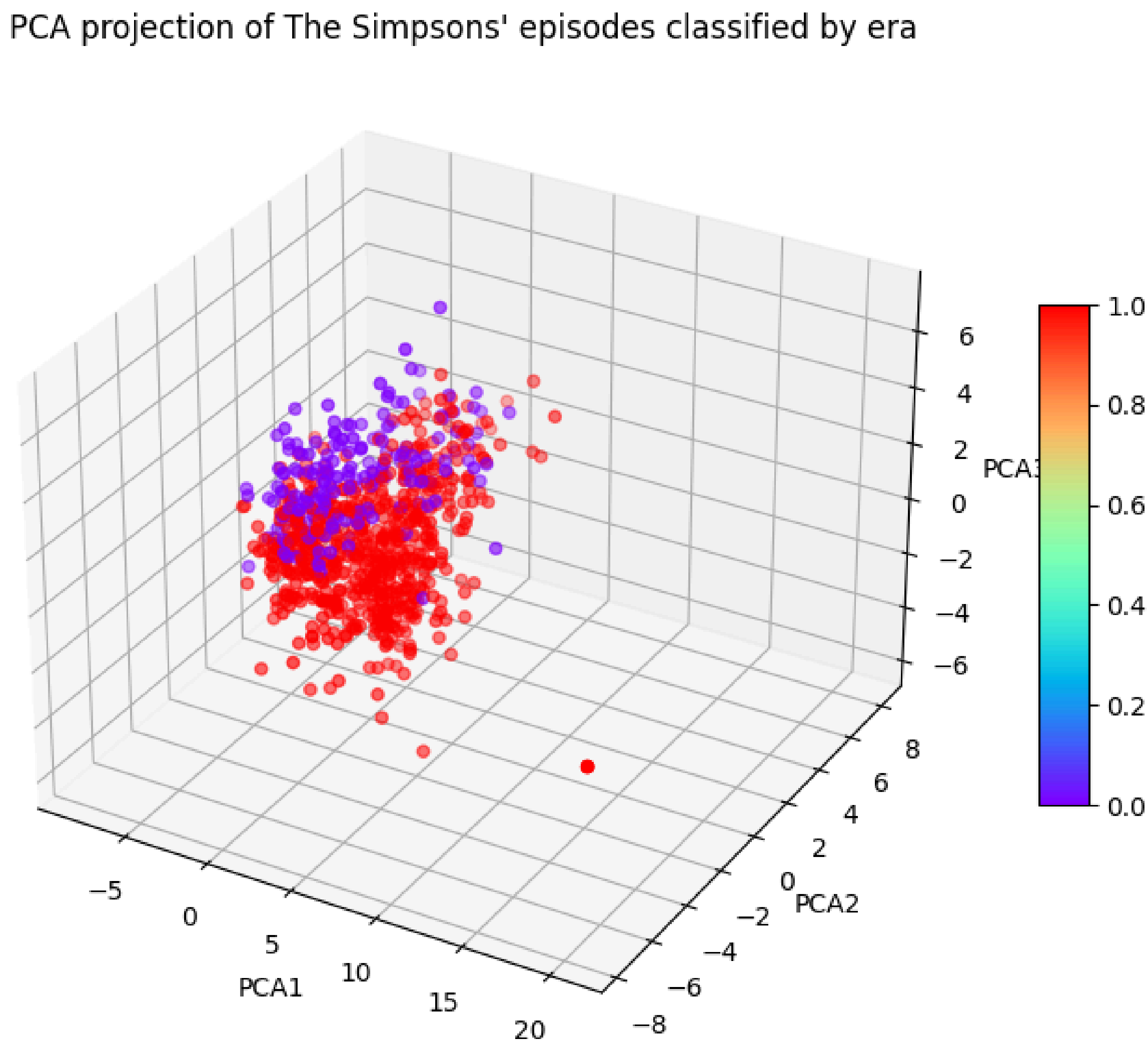
```
# Create a 3D scatter plot of the data in the encoder dimensions
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# 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 The Simpsons' episodes classified by era")

# 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 [33]:
```
# 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('PCA1')
ax.set_ylabel('PCA2')
ax.set_zlabel('PCA3')

# Add a title
ax.set_title("PCA projection of The Simpsons' episodes classified by era")

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

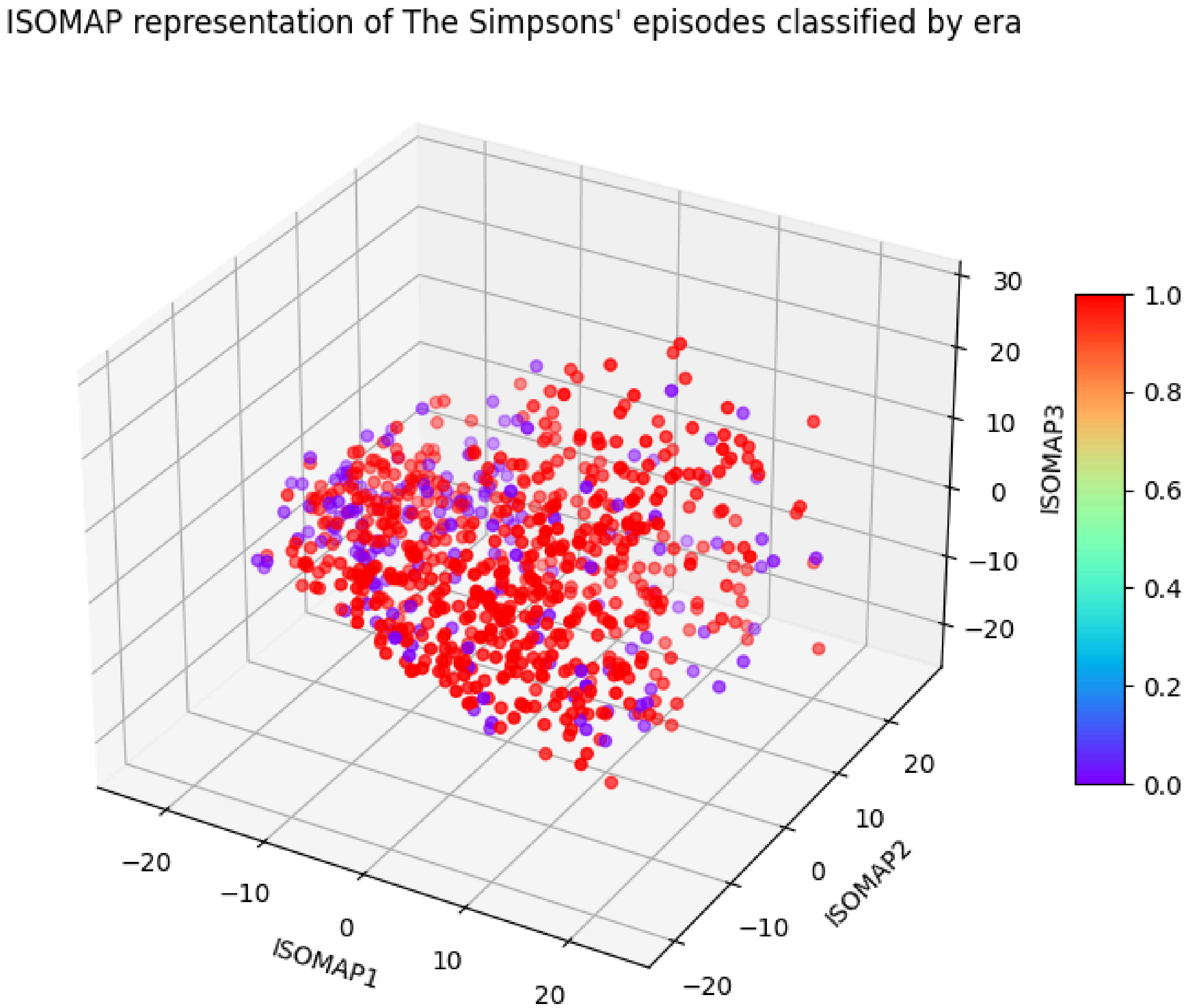
```
# Display the plot
plt.show()
```



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

In [34]:
```
# 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 The Simpsons' episodes classified by era")

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

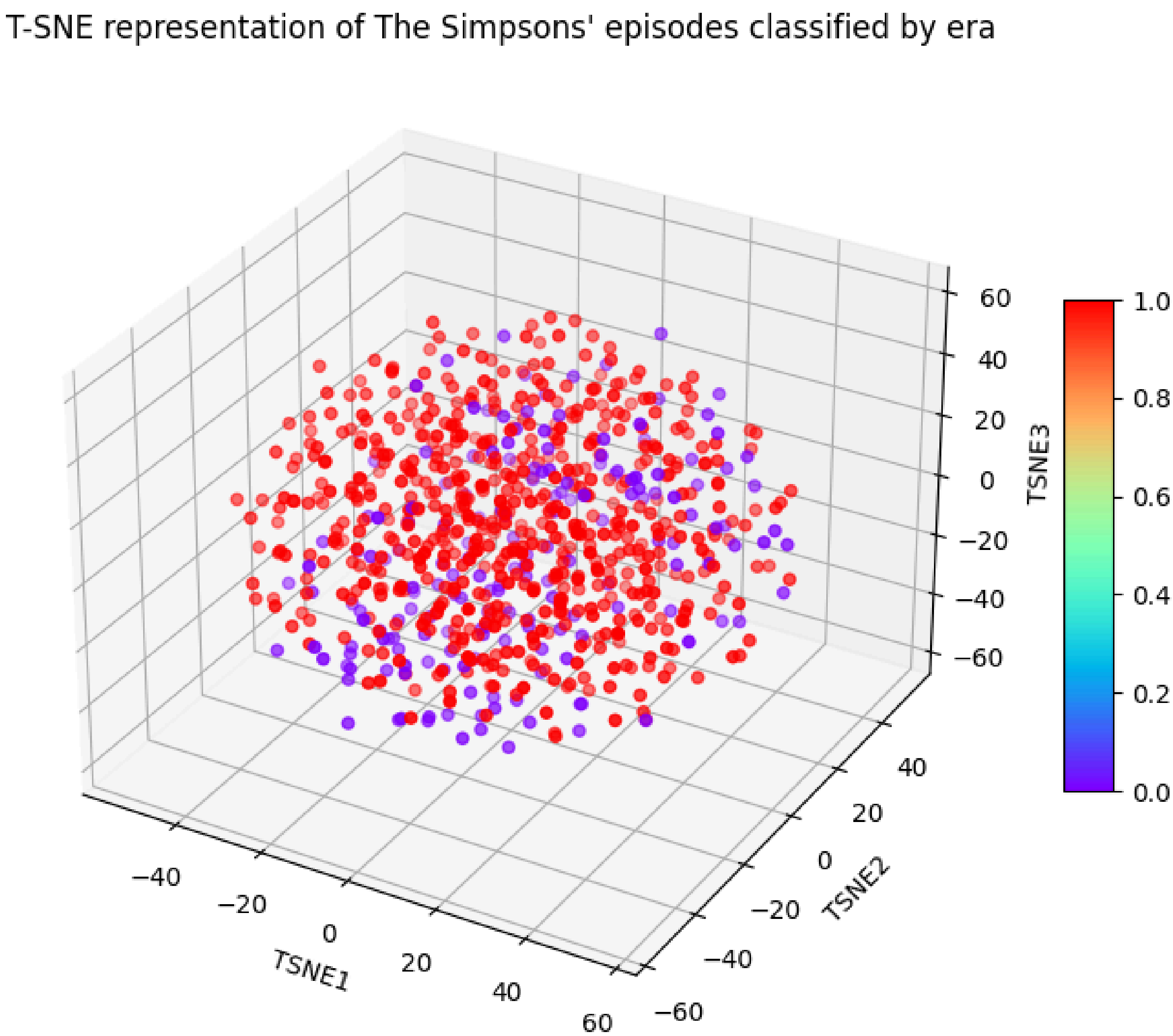
```
# Display the plot
plt.show()
```



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

In [35]:
```
# 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 The Simpsons' episodes classified by era")

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

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



En este caso, la mejor separación la logra el autoencoder, y en este caso hasta parece factible que se pueda identificar la era de los capítulos de The Simpsons