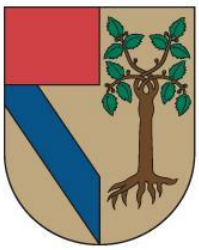


UNIVERSIDAD PANAMERICANA

Materia: Procesamiento del Lenguaje Natural

Profesor: Bernardo Irving Hernández Uribe

Alumnos: Sara Rocío Miranda Mateos

Diego Arenas Trevilla

Rodrigo Lucas Nieto

Fecha de entrega: 20/10/2023

Nombre del proyecto:

Modelos de clasificación Multiclase

DNN (Deep Neural Network) es una red neuronal con varias capas ocultas. Las capas ocultas permiten a la red aprender patrones complejos en los datos.

CNN (Convolutional Neural Network) es una red neuronal que utiliza filtros convolucionales para extraer características de los datos. Los filtros convolucionales son útiles para extraer patrones en datos de imágenes y texto.

LSTM (Long Short-Term Memory) es una red neuronal recurrente que permite aprender dependencias a largo plazo en los datos. Las redes recurrentes son útiles para procesar datos secuenciales, como el texto.

Introducción

¿Qué categorías se van a clasificar?

- Personas
- Objetos
- Lugares
- Animales
- Series

¿Qué se realizó en el código?

- Lectura y preprocesamiento de los textos de cada categoría.
- Creación de matrices de entrada y salida.
- Uso de embeddings para representar las palabras.
- Definición y entrenamiento de los modelos de clasificación.
- Evaluación de los modelos y comparación de sus resultados.

¿Por qué se eligió cierto tipo de preprocesamiento?

- Eliminación de Stopwords
    - Se reduce el ruido en los datos y se pueden obtener características más relevantes.
- Eliminación de Etiquetas HTML
    - Es importante cuando se trabaja con datos que provienen de páginas de internet, ya que muchas veces vienen con etiquetas html y pueden interferir con el análisis.
- Texto a minúsculas
    - Evitamos que el modelo considere "Palabra" y "palabra" como palabras distintas.
- Tokenización y Detokenización
    - Dividimos el texto en palabras y luego las volvemos a unir de manera coherente y legible.
- Eliminar caracteres especiales y espacios en blanco innecesarios

Evidencias

```python
# Importar todas las librerías que se utilizarán
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
import os

from numpy import array
from keras_preprocessing.sequence import pad_sequences
from keras import Sequential
from keras.layers import Activation, Dropout, Dense
from keras.layers import Flatten, Conv1D, LSTM, GlobalMaxPooling1D, Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer

from nltk.tokenize import RegexpTokenizer, TreebankWordDetokenizer

import matplotlib.pyplot as plt

# Filtrado de StopWords utilizando NLTK
from nltk.tokenize import RegexpTokenizer
from nltk.tokenize.treebank import TreebankWordDetokenizer
```

Esta celda importa varias librerías de Python que se utilizarán para entrenar modelos de aprendizaje automático para procesamiento de lenguaje natural.

```python
def build_corpus(root_folder):
    """Este método lee los de datos del corpus, que serían la serie de textos relacionados a Libros, Animales,
       Personas famosas, Lugares, Objetos y Series (la categoría agregada)

    Args:
       root folder (string): Es la carpeta que contiene a los textos a procesar

    Returns:
       corpus (diccionario): Es el conjunto de textos divididos por categorías donde las categorías son las
       llaves y los datos son los textos
    """

    corpus = {}

    # Se va por la carpeta
    for dirpath, dirnames, filenames in os.walk(root_folder):

        # Si encuentra la carpeta deseada que continue
        if dirpath == root_folder:
            continue

        # Una vez encuentra la carpeta que contiene otras carpetas, se guarda el nombre y hace éste la llave del corpus
        folder_name = os.path.basename(dirpath)
        corpus[folder_name] = []

        # Se itera por los nombres de archivos que se tienen dentro de la carpeta de cada categoría
        for filename in filenames:

            # Se asegura que estemos leyendo puros archivos de texto (que terminen con .txt)
            if filename.endswith('.txt'):
                filepath = os.path.join(dirpath, filename)
                with open(filepath, 'r') as file:

                    # Se leen los archivos de texto y se guardan en el corpus
                    content = file.read()
                    corpus[folder_name].append(content)

    # Se regresa el corpus con toda la información requerida
    return corpus
```

Lee los datos del corpus los organiza en un diccionario donde las claves son las categorías de los textos y los valores son las listas de textos correspondientes a cada categoría. Devuelve corpus (diccionario): Es el conjunto de textos divididos por categorías

```python
from nltk.tokenize import RegexpTokenizer, TreebankWordDetokenizer

def preprocess_text(corpus):
    """Este método hace el preprocesamiento del corpus

    Args:
        corpus (diccionario): Contiene la información a preprocesar

    Returns:
        X_processed (lista): Es una lista que contiene los textos preprocesados
    """

    # Se inicializa las variables necesarias
    stop_words = set(stopwords.words('spanish'))
    X_processed = []
    removedor_tags = re.compile(r'<[^>]+>')

    # Se va por cada uno de los textos y los guarda en la variable
    sentences = [text for sublist in corpus.values() for text in sublist]

    # Se itera por los textos
    for sen in sentences:

        # Se reemplazan stopwords
        for stopword in stop_words:
            sen = sen.replace(" " + stopword + " ", " ")

        # Se quitan los tags
        sen = removedor_tags.sub('', sen)

        # Se transforma todo a minúsculas
        sen = re.sub(r'\s+', ' ', sen).strip().lower()

        # Se tokenizan solo las palabras
        tokenizer = RegexpTokenizer(r'\w+')
        tokens = tokenizer.tokenize(sen)

        # Se destoqueniza y se guarda en una variable
        processed_sentence = TreebankWordDetokenizer().detokenize(tokens)

        # Se agrega a una lista que contendrá todos los textos
        X_processed.append(processed_sentence)

    # Se regresa la lista preprocesada
    return X_processed
```

El preprocesamiento hace la eliminación de stopwords, la eliminación de etiquetas HTML, la transformación a minúsculas, la tokenización de palabras y la detokenización de las palabras.

La función devuelve una lista con los textos preprocesados.

```python
from sklearn.feature_extraction.text import CountVectorizer

def create_matrix(corpus):
    """Este método crea una matriz para clasificar los textos

    Args:
        corpus (diccionario): Contiene la información inicial con categorías y textos

    Returns:
        X, Y: Son la matrices de entrada y salida
    """

    # Se llama al método que preprocesa la información y se guarda en una variable
    X_processed = preprocess_text(corpus)
    vectorizer = CountVectorizer()

    # Se vectoriza y crea la matriz y se guarda la categoría
    X = vectorizer.fit_transform(X_processed)
    Y = [label for label, texts in corpus.items() for _ in texts]

    # Se regresa la matriz y la categoría
    return X, Y, X_processed
```

Crea una matriz de características y una matriz de etiquetas para clasificar textos.

La función devuelve tres valores: la matriz de características, la matriz de etiquetas y la lista de textos preprocesados.

```python
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer

def Training_Data(X_processed, Y):
    """Este método crea las variables de X y Y train y test

    Args:
        corpus (diccionario): Es un diccionario que tiene los textos

    Returns:
        X, Y: Son la matrices de entrada y salida con el proceso ya hecho
    """

    # Se hace el split de los textos que usará para entrenar y para validar
    X_train_texts, X_test_texts, y_train, y_test = train_test_split(X_processed, Y, test_size=0.20)

    # Se hace el tokenizer
    tokenizer = Tokenizer(num_words=10000)
    tokenizer.fit_on_texts(X_train_texts)

    # Se ajusta a valores los X_train y X_test
    X_train = tokenizer.texts_to_sequences(X_train_texts)
    X_test = tokenizer.texts_to_sequences(X_test_texts)

    # regresa los valores de X y Y de train y test
    return X_train, X_test, y_train, y_test, tokenizer
```

Crea las variables de entrada y salida para entrenar y validar un modelo de aprendizaje automático para clasificación de textos.

La función devuelve las matrices de entrada y salida para entrenamiento y validación, así como el objeto tokenizer que se utilizó para convertir los textos en secuencias.

```python
def Padding(X_train, X_test):
    """Este método hace el padding

    Args:
        X_train, X_test: son las matrices de entrenamiento y testeo a las que se hará el padding

    Returns:
        X, Y: Son la matrices de entrada y salida con el proceso ya hecho
    """

    # Se establece una longitud máxima para el padding
    maxlen = 150

    # Se hace el padding en los valores de entrenamiento y de testeo
    X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
    X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

    #Se imprimen los valores
    print("Matriz de valores para las palabras:")
    X_train.shape
    print(X_train)
    print(y_train)

    # Regresa el X_train y X_test con el padding
    return X_train, X_test
```

La función aplica el padding a las matrices de entrada para que tengan la misma longitud. Se establece una longitud máxima para el padding y se utiliza la función pad_sequences.

La función devuelve las matrices de entrada con el padding aplicado.

```python
from numpy import asarray
from numpy import zeros

def Embeddings(tokenizer):
    """Este método contiene todo lo necesario para la creación y lectura de embedding así como sus
       características necesarias

    Args:
        Este método no contiene argumentos

    Returns:
        embedding_matrix (matriz): Es la matriz con los pesos del archivo Word2Vect
    """

    #Se inicializa la variable de diccionario y se lee el
    embeddings_dictionary = dict()
    Embeddings_file = open('Word2Vect_Spanish.txt', encoding="utf8")

    # Se leen los datos del archivo de embeddings
    for linea in Embeddings_file:
        caracts = linea.split()
        palabra = caracts[0]
        vector = asarray(caracts[1:], dtype='float32')
        embeddings_dictionary [palabra] = vector
    Embeddings_file.close()

    # Se inicializa el tamaño para la matriz de embeddings
    vocab_size = len(tokenizer.word_index) + 1

    # Se crea y hace la matriz de embeddings
    embedding_matrix = zeros((vocab_size, 300))
    for word, index in tokenizer.word_index.items():
        embedding_vector = embeddings_dictionary.get(word)
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector

    # regresa la matriz de embeddings
    return embedding_matrix, vocab_size
```

La función crea una matriz de embeddings a partir de un archivo de embeddings tokenizer se utiliza para convertir los textos en secuencias de números enteros.

La función devuelve la matriz de embeddings y el tamaño del vocabulario.

```python
import matplotlib.pyplot as plt

def Show_Graphics(history):
    """Este método imprime las gráficas del train y test y su accurancy

    Args:
        history: son las épocas por las que pasó el proceso para aprender

    Returns:
        no regresa nada
    """

    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])

    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train','test'], loc='upper left')
    plt.show()

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])

    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train','test'], loc='upper left')
    plt.show()
```

La función muestra dos gráficas: una que representa la precisión del modelo en el conjunto de entrenamiento y en el conjunto de validación a lo largo de las épocas, y otra que representa la pérdida del modelo en el conjunto de entrenamiento y en el conjunto de validación a lo largo de las épocas.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

def DNN(vocab_size, embedding_matrix, maxlen, y_train, y_test, X_train, X_test):
    """Este método hace la red neuronal con el método de DNN

    Args:
        vocab_size (int): es el tamaño del embedding
        embedding_matrix (matriz): es la matriz de pesos
        maxlen (int): es el máximo del input de la embedding
        y_train, y_test, x_train, x_test: son las matrices de entrenamiento y validación en x y y

    Returns:
        no regresa nada
    """
    # Definición del modelo
    model = Sequential()
    embedding_layer = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=maxlen, trainable=False)
    model.add(embedding_layer)
    model.add(Flatten())
    model.add(Dense(5, activation='sigmoid'))
    print(model.summary())

    # Codificación de etiquetas
    encoder = LabelEncoder()
    y_train_encoded = encoder.fit_transform(y_train)
    y_test_encoded = encoder.transform(y_test)
    y_train_np = to_categorical(y_train_encoded)
    y_test_np = to_categorical(y_test_encoded)

    # Compilación del modelo
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    # Se hace el Padding
    X_train_padded, X_test_padded = Padding(X_train, X_test)

    # Utilizamos el método fit para ajustar los datos de nuestro modelo a la configuración que definimos
    history = model.fit(X_train_padded, y_train_np, batch_size=10, epochs=30, verbose=1, validation_split=0.2)

    score = model.evaluate(X_test_padded, y_test_np, verbose=1)
    print("Test Loss:", score[0])
    print("Test Accuracy:", score[1])

    Show_Graphics(history)
    Confussion_Matrix(X_train, X_test, y_test_np, model)
```

La función crea y entrena un modelo de red neuronal con una capa de embedding, una capa de aplanamiento y una capa densa con activación sigmoide para clasificar textos en español.

La función toma como argumentos el tamaño del vocabulario vocab_size, la matriz de embeddings embedding_matrix, la longitud máxima de los textos maxlen, las matrices de etiquetas de entrenamiento y validación y_train y y_test, y las matrices de características de entrenamiento y validación X_train y X_test.

Se define el modelo de red neuronal utilizando la clase Sequential. Se agrega una capa de embedding con los pesos de la matriz de embeddings, una capa de aplanamiento y una capa densa de 5 por el número de categorias. Luego, se compila el modelo utilizando el optimizador Adam y la función de pérdida de entropía cruzada categórica.

Después, se aplica el padding a las matrices de entrada utilizando la función Padding. Se utiliza el método fit para entrenar el modelo con los datos de entrenamiento y se evalúa el modelo con los datos de validación utilizando el método evaluate. Al final, se muestra una gráfica de la precisión y la pérdida del modelo a lo largo de las épocas utilizando la función Show_Graphics y se muestra la matriz de confusión utilizando la función Confussion_Matrix. Usa las funciones definidas anteriormente.

```python
def CNN(vocab_size, embedding_matrix, maxlen, y_train, y_test, X_train, X_test):
    """Este método hace la red neuronal con el método de CNN

    Args:
        vocab_size (int): es el tamaño del embedding
        embedding_matrix (matriz): es la matriz de pesos
        maxlen (int): es el máximo del input de la embedding
        y_train, y_test, x_train, x_test: son las matrices de entrenamiento y validación en x y y

    Returns:
        no regresa nada
    """
    # Declaración de modelo Secuencial
    model = Sequential()

    # Declaración de las capas del modelo convolucional
    embedding_layer = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=maxlen , trainable=False)
    model.add(embedding_layer)
    model.add(Dense(64, activation='relu'))
    model.add(Conv1D(128, 5, activation='relu'))
    model.add(GlobalMaxPooling1D())
    model.add(Dense(5, activation='sigmoid'))
    # Impresión de parámetros del modelo
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    print(model.summary())

    # Codificación de etiquetas
    encoder = LabelEncoder()
    y_train_encoded = encoder.fit_transform(y_train)
    y_test_encoded = encoder.transform(y_test)
    y_train_np = to_categorical(y_train_encoded)
    y_test_np = to_categorical(y_test_encoded)

    # Compilación del modelo
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    # Se hace el Padding
    X_train_padded, X_test_padded = Padding(X_train, X_test)

    # Utilizamos el método fit para ajustar los datos de nuestro modelo a la configuración que definimos
    history = model.fit(X_train_padded, y_train_np, batch_size=10, epochs=30, verbose=1, validation_split=0.2)

    score = model.evaluate(X_test_padded, y_test_np, verbose=1)
    print("Test Loss:", score[0])
    print("Test Accuracy:", score[1])

    Show_Graphics(history)
    Confussion_Matrix(X_train, X_test, y_test_np, model)
```

La función hace lo mismo que la función anterior (la función DNN) pero cambiando las capas del modelo.

Primero, se define una capa de embedding utilizando la clase Embedding. Luego, se agregan capas al modelo utilizando la clase Sequentia. Se agrega una capa densa con 64 neuronas y activación ReLU, una capa convolucional con 128 filtros de tamaño 5 y activación ReLU, una capa de pooling global máxima y una capa densa con 5 neuronas y activación sigmoide.

La capa de pooling global máxima se utiliza para reducir la dimensionalidad de la salida de la capa convolucional y obtener una representación fija de cada texto. La capa densa final es de 5 por el número de categorías a clasificar.

```python
from tensorflow.keras.layers import LSTM

def LSTM_model(vocab_size, embedding_matrix, maxlen, y_train, y_test, X_train, X_test):
    """Este método hace la red neuronal con el método de LSTM

    Args:
        vocab_size (int): es el tamaño del embedding
        embedding_matrix (matriz): es la matriz de pesos
        maxlen (int): es el máximo del input de la embedding
        y_train, y_test, x_train, x_test: son las matrices de entrenamiento y validación en x y y

    Returns:
        no regresa nada
    """
    model = Sequential()

    # Declaración de las capas del modelo LSTM
    embedding_layer = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=maxlen , trainable=True)
    model.add(embedding_layer)
    model.add(LSTM(64))
    model.add(Dense(5, activation='sigmoid'))

    # Impresión de parámetros del modelo
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    print(model.summary())

    # Codificación de etiquetas
    encoder = LabelEncoder()
    y_train_encoded = encoder.fit_transform(y_train)
    y_test_encoded = encoder.transform(y_test)
    y_train_np = to_categorical(y_train_encoded)
    y_test_np = to_categorical(y_test_encoded)

    # Compilación del modelo
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    # Compilación del modelo
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    # Se hace el Padding
    X_train_padded, X_test_padded = Padding(X_train, X_test)

    # Utilizamos el método fit para ajustar los datos de nuestro modelo a la configuración que definimos
    history = model.fit(X_train_padded, y_train_np, batch_size=10, epochs=30, verbose=1, validation_split=0.2)

    score = model.evaluate(X_test_padded, y_test_np, verbose=1)
    print("Test Loss:", score[0])
    print("Test Accuracy:", score[1])

    Show_Graphics(history)
    Confussion_Matrix(X_train, X_test, y_test_np, model)
```

La función hace lo mismo que la función anterior (la función DNN) pero cambiando las capas del modelo.

Primero, se define una capa de embedding utilizando la clase Embedding. La capa de embedding tiene como argumentos el tamaño del vocabulario vocab_size, la dimensión de los vectores de embeddings 300, los pesos de la matriz de embeddings embedding_matrix, la longitud máxima de los textos maxlen y se establece que los pesos de la matriz de embeddings serán entrenables.

Luego, agregamos una capa LSTM con 64 neuronas utilizando la clase LSTM. Al final se agrega una capa densa con 5 neuronas por el número de categorías que tenemos.

```python
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
import itertools

def Confussion_Matrix(X_train, X_test, y_test_np, model):
    """Este método hace la matriz de confusión

    Args:
        X_train, X_test: son los X entrenados y de testeo
        y_test_np: es el valor de testeo de y con numpy

    Returns:
        no regresa nada
    """

    X_train_padded, X_test_padded = Padding(X_train, X_test)

    y_pred = model.predict(X_test_padded)
    y_pred_classes = np.argmax(y_pred, axis=1)

    y_true_classes = np.argmax(y_test_np, axis=1)

    cm = confusion_matrix(y_true_classes, y_pred_classes)

    plt.figure(figsize=(10,7))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Matriz de Confusión')
    plt.colorbar()

    classes = sorted(set(y_true_classes))
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Verdadero')
    plt.xlabel('Predicho')
    plt.show()
```

La función crea y muestra la matriz de confusión para los modelos. La función toma como argumentos las matrices de características de entrenamiento y validación X_train y X_test, la matriz de etiquetas de validación y_test_np y el modelo de clasificación model.

La función utiliza la función Padding para aplicarlo a las matrices de entrada. Luego, se utiliza el modelo de clasificación para predecir las etiquetas de los datos de validación y se calcula la matriz de confusión utilizando la función confusion_matrix.

Al final muestra la matriz de confusión en una gráfica. La matriz de confusión muestra la cantidad de predicciones correctas e incorrectas para cada una de las categorías de clasificación.

```python
# Se leen los archivos
root_folder = 'Textos_para_Clasificar'
corpus = build_corpus(root_folder)
print(corpus)
```

✓ 0.0s

```
{'Objetos': ['El teléfono inteligente (smartphone en inglés) es un tipo de ordenador de bolsill
```

```python
# Se hace la matriz
X, Y, X_processed = create_matrix(corpus)
print(X_processed)
```

✓ 0.0s

```
['el teléfono inteligente smartphone inglés tipo ordenador bolsillo capacidades teléfono móvil
```

```python
# Se inicializan las variables de entrenamiento
X_train, X_test, y_train, y_test, tokenizer = Training_Data(X_processed, Y)
```

✓ 0.0s

```python
# Se hace la matriz de embeddings
embedding_matrix, vocab_size = Embeddings(tokenizer)
```

✓ 29.2s

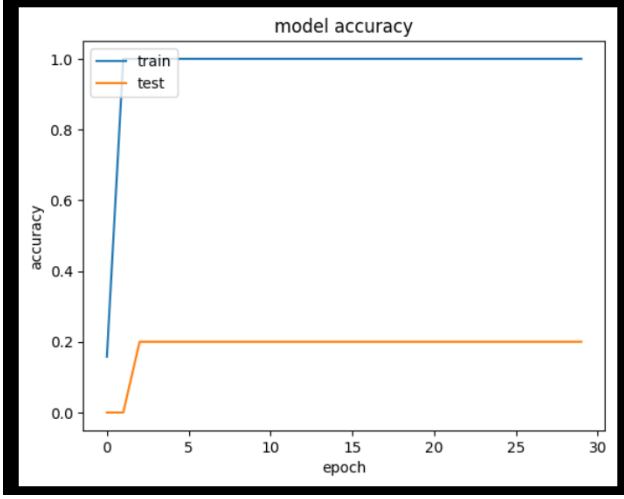
Se llaman las funciones para preparar las variables y usarlas en cada uno de los modelos.

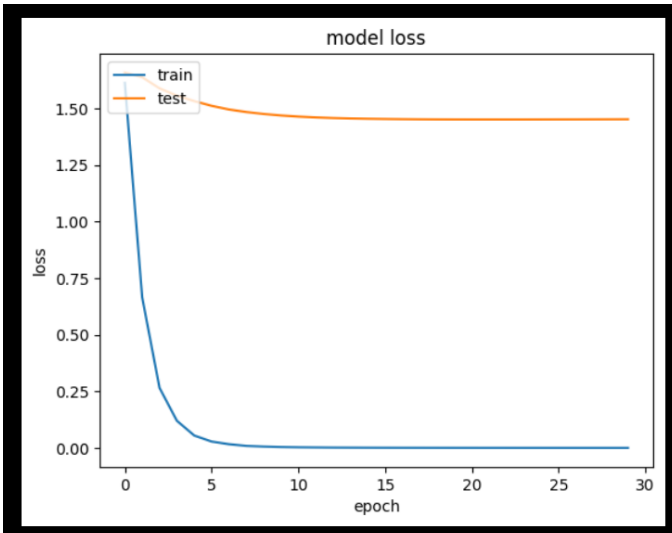
```
# Se hace la red neuronal de DNN
DNN(vocab_size, embedding_matrix, 150, y_train, y_test, X_train, X_test)
```

✓ 1.6s

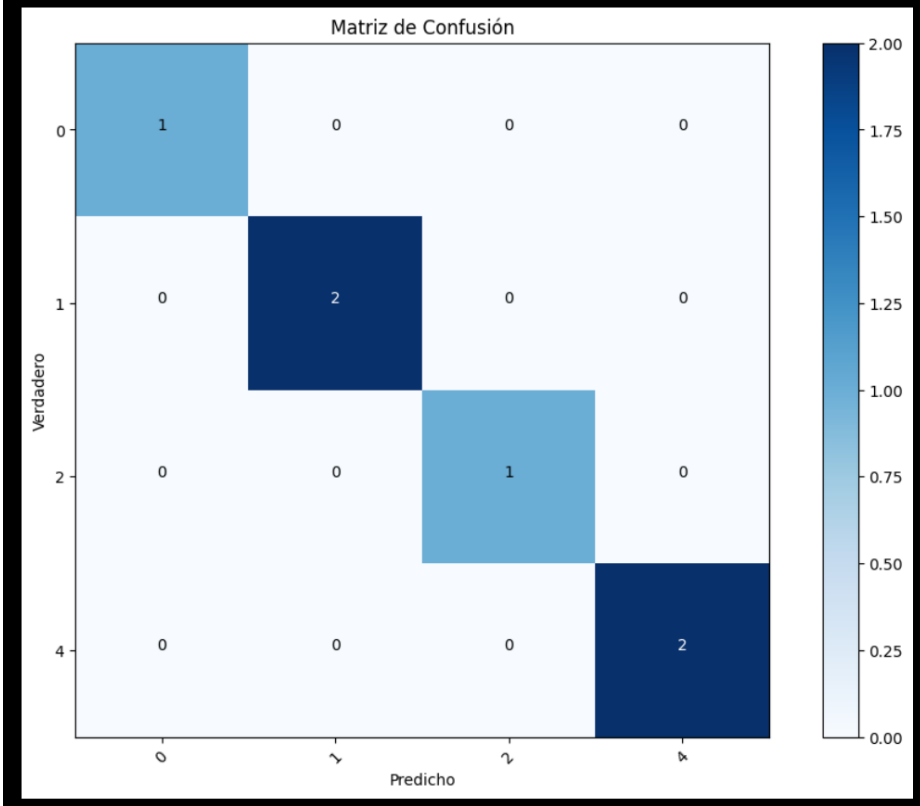
```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 150, 300)          668400

 flatten_4 (Flatten)         (None, 45000)             0

 dense_6 (Dense)             (None, 5)                 225005

=================================================================
Total params: 893405 (3.41 MB)
Trainable params: 225005 (878.93 KB)
Non-trainable params: 668400 (2.55 MB)
_________________________________________________________________
None
Matriz de valores para las palabras:
[[ 662  663   89 ...    0    0    0]
 [ 207   96  362 ...  786  787  788]
 [  32  852   10 ...  230   11  906]
 ...
 [ 585 2094 2095 ... 2169 2170 2171]
 [2172  338   34 ...    0    0    0]
 [   4   24 2197 ...    0    0    0]]
['Personas', 'Animales', 'Lugares', 'Series', 'Series', 'Lugares', 'Animales', 'Animales', 'Objetos', 'Personas', 'Lugares', 'Lugares',
Epoch 1/30
2/2 [==============================] - 0s 97ms/step - loss: 1.6121 - accuracy: 0.1579 - val_loss: 1.6580 - val_accuracy: 0.0000e+00
Epoch 2/30
2/2 [==============================] - 0s 16ms/step - loss: 0.6657 - accuracy: 1.0000 - val_loss: 1.6374 - val_accuracy: 0.0000e+00
Epoch 3/30
2/2 [==============================] - 0s 17ms/step - loss: 0.2657 - accuracy: 1.0000 - val_loss: 1.5882 - val_accuracy: 0.2000
Epoch 4/30
2/2 [==============================] - 0s 15ms/step - loss: 0.1205 - accuracy: 1.0000 - val_loss: 1.5561 - val_accuracy: 0.2000
Epoch 5/30
2/2 [==============================] - 0s 15ms/step - loss: 0.0554 - accuracy: 1.0000 - val_loss: 1.5319 - val_accuracy: 0.2000
Epoch 6/30
2/2 [==============================] - 0s 15ms/step - loss: 0.0288 - accuracy: 1.0000 - val_loss: 1.5111 - val_accuracy: 0.2000
Epoch 7/30
2/2 [==============================] - 0s 16ms/step - loss: 0.0170 - accuracy: 1.0000 - val_loss: 1.4949 - val_accuracy: 0.2000
Epoch 8/30
...
2/2 [==============================] - 0s 16ms/step - loss: 7.0118e-04 - accuracy: 1.0000 - val_loss: 1.4517 - val_accuracy: 0.2000
1/1 [==============================] - 0s 12ms/step - loss: 0.6625 - accuracy: 1.0000
Test Loss: 0.6625456213951111
Test Accuracy: 1.0
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...






```
Matriz de valores para las palabras:
[[ 662  663   89 ...    0    0    0]
 [ 207   96  362 ...  786  787  788]
 [  32  852   10 ...  230   11  906]
 ...
 [ 585 2094 2095 ... 2169 2170 2171]
 [2172  338   34 ...    0    0    0]
 [   4   24 2197 ...    0    0    0]]
['Personas', 'Animales', 'Lugares', 'Series', 'Series', 'Lugares', 'Animales', 'Animales', 'Objetos', 'Personas', 'Lugares', 'L
1/1 [==============================] - 0s 37ms/step
```

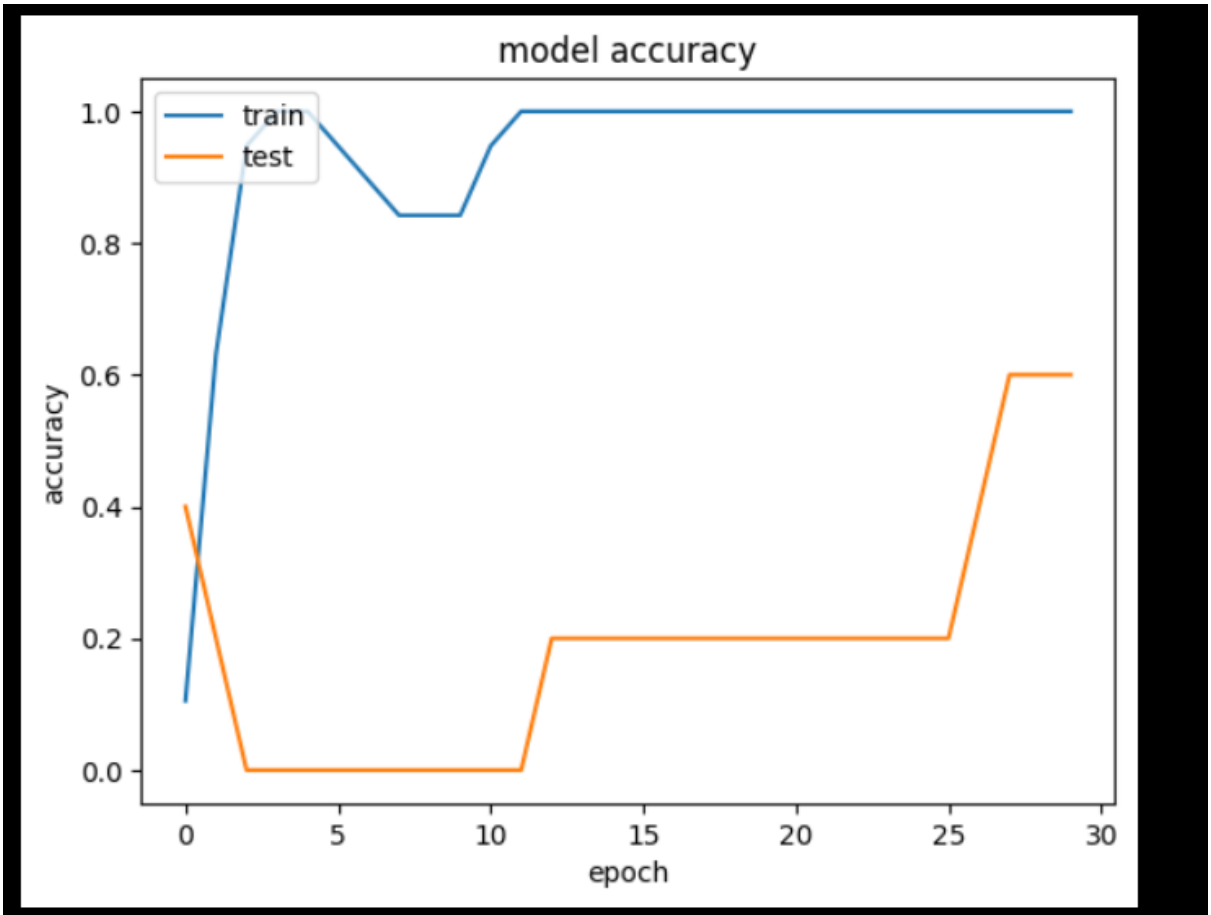


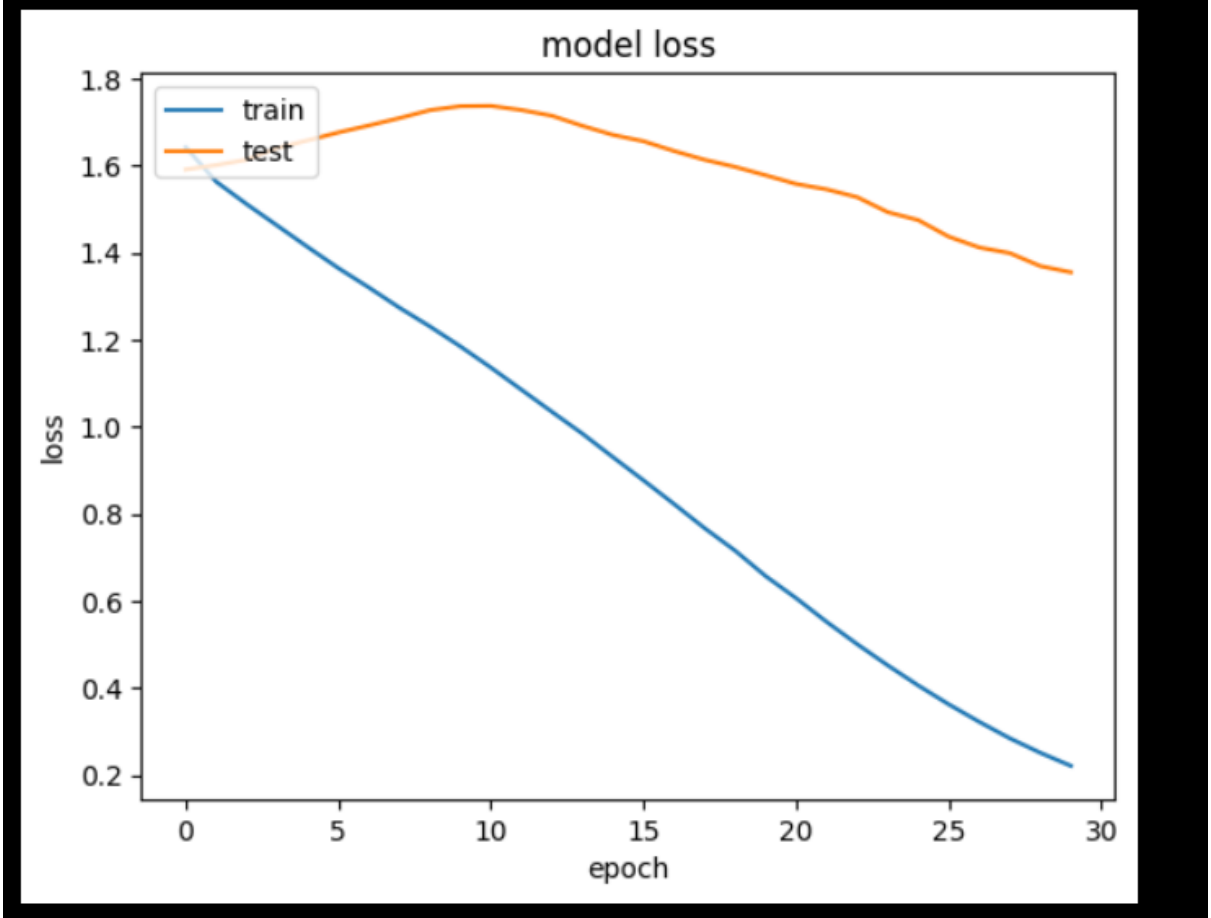
Se llama la función DNN

```
  # Se hace la red neuronal de CNN
  CNN(vocab_size, embedding_matrix, 150, y_train, y_test, X_train, X_test)
```

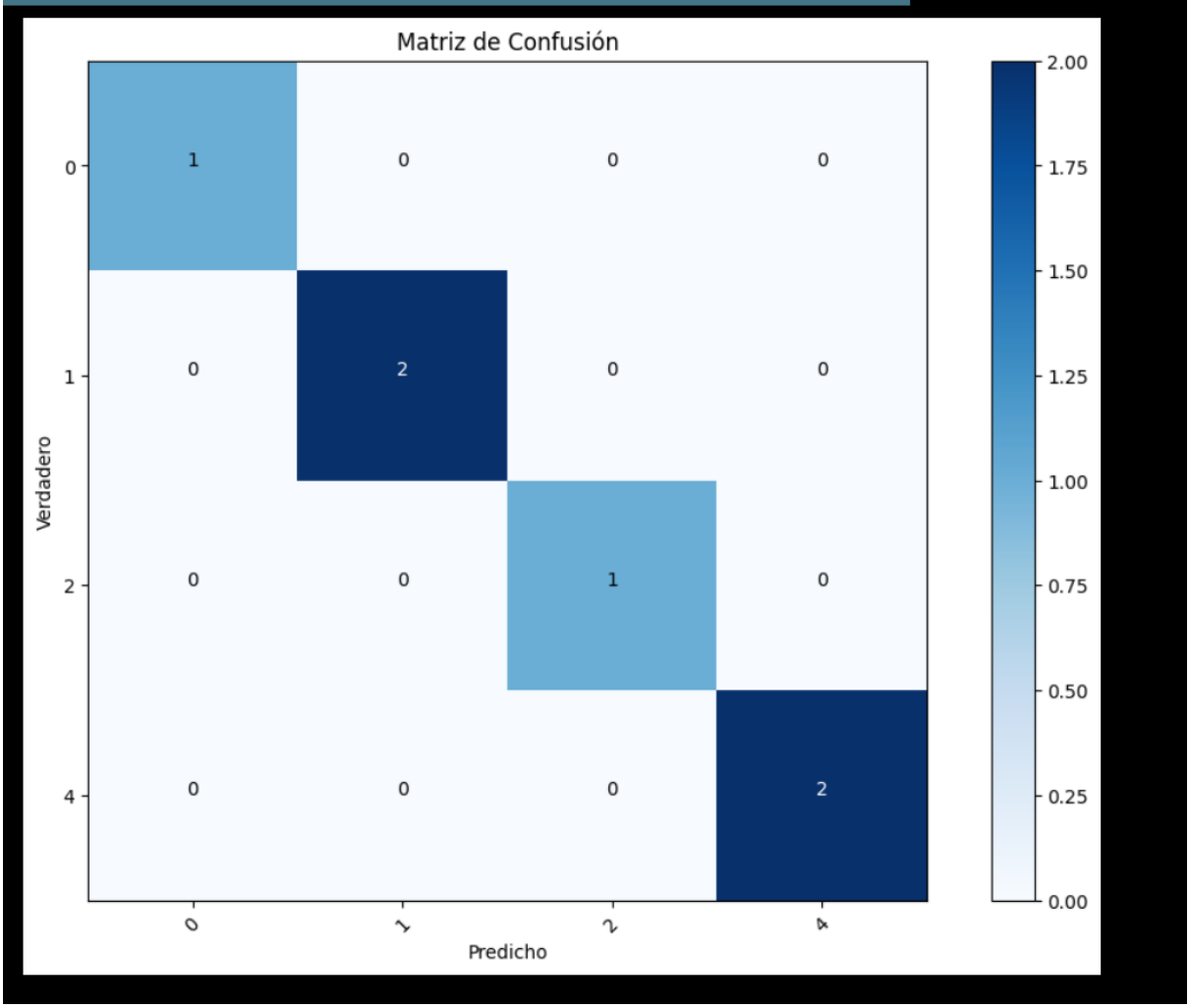
✓ 1.6s

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, 150, 300)          668400

 dense_7 (Dense)             (None, 150, 64)           19264

 conv1d_1 (Conv1D)           (None, 146, 128)          41088

 global_max_pooling1d_1 (Gl  (None, 128)               0
 obalMaxPooling1D)

 dense_8 (Dense)             (None, 5)                 645

=================================================================
Total params: 729397 (2.78 MB)
Trainable params: 60997 (238.27 KB)
Non-trainable params: 668400 (2.55 MB)
_________________________________________________________________
None
Matriz de valores para las palabras:
[[ 662  663   89 ...    0    0    0]
 [ 207   96  362 ...  786  787  788]
 [  32  852   10 ...  230   11  906]
...
 [2172  338   34 ...    0    0    0]
 [   4   24 2197 ...    0    0    0]]
['Personas', 'Animales', 'Lugares', 'Series', 'Series', 'Lugares', 'Animales', 'Animales', 'Objetos', 'Personas', 'Lugares', 'Lugares',
Epoch 1/30
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
2/2 [==============================] - 1s 96ms/step - loss: 1.6417 - accuracy: 0.1053 - val_loss: 1.5904 - val_accuracy: 0.4000
Epoch 2/30
2/2 [==============================] - 0s 18ms/step - loss: 1.5624 - accuracy: 0.6316 - val_loss: 1.6006 - val_accuracy: 0.2000
Epoch 3/30
2/2 [==============================] - 0s 18ms/step - loss: 1.5107 - accuracy: 0.9474 - val_loss: 1.6129 - val_accuracy: 0.0000e+00
Epoch 4/30
2/2 [==============================] - 0s 17ms/step - loss: 1.4616 - accuracy: 1.0000 - val_loss: 1.6398 - val_accuracy: 0.0000e+00
Epoch 5/30
2/2 [==============================] - 0s 17ms/step - loss: 1.4123 - accuracy: 1.0000 - val_loss: 1.6574 - val_accuracy: 0.0000e+00
Epoch 6/30
2/2 [==============================] - 0s 19ms/step - loss: 1.3641 - accuracy: 0.9474 - val_loss: 1.6754 - val_accuracy: 0.0000e+00
Epoch 7/30
2/2 [==============================] - 0s 17ms/step - loss: 1.3196 - accuracy: 0.8947 - val_loss: 1.6916 - val_accuracy: 0.0000e+00
Epoch 8/30
2/2 [==============================] - 0s 18ms/step - loss: 1.2732 - accuracy: 0.8421 - val_loss: 1.7084 - val_accuracy: 0.0000e+00
Epoch 9/30
2/2 [==============================] - 0s 17ms/step - loss: 1.2301 - accuracy: 0.8421 - val_loss: 1.7267 - val_accuracy: 0.0000e+00
Epoch 10/30
2/2 [==============================] - 0s 19ms/step - loss: 1.1842 - accuracy: 0.8421 - val_loss: 1.7358 - val_accuracy: 0.0000e+00
Epoch 11/30
2/2 [==============================] - 0s 18ms/step - loss: 1.1356 - accuracy: 0.9474 - val_loss: 1.7365 - val_accuracy: 0.0000e+00
Epoch 12/30
2/2 [==============================] - 0s 17ms/step - loss: 1.0848 - accuracy: 1.0000 - val_loss: 1.7270 - val_accuracy: 0.0000e+00
Epoch 13/30
2/2 [==============================] - 0s 19ms/step - loss: 1.0340 - accuracy: 1.0000 - val_loss: 1.7138 - val_accuracy: 0.2000
...
2/2 [==============================] - 0s 18ms/step - loss: 0.2211 - accuracy: 1.0000 - val_loss: 1.3545 - val_accuracy: 0.6000
1/1 [==============================] - 0s 12ms/step - loss: 0.9151 - accuracy: 1.0000
Test Loss: 0.9151215553283691
Test Accuracy: 1.0
```


model accuracy
train
test
accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30


model loss
train
test
loss
epoch
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30

```
Matriz de valores para las palabras:
[[ 662  663   89 ...    0    0    0]
 [ 207   96  362 ...  786  787  788]
 [  32  852   10 ...  230   11  906]
 ...
 [ 585 2094 2095 ... 2169 2170 2171]
 [2172  338   34 ...    0    0    0]
 [   4   24 2197 ...    0    0    0]]
['Personas', 'Animales', 'Lugares', 'Series', 'Series', 'Lugares', 'Animales', 'Animales', 'Objetos', '
1/1 [==============================] - 0s 41ms/step
```



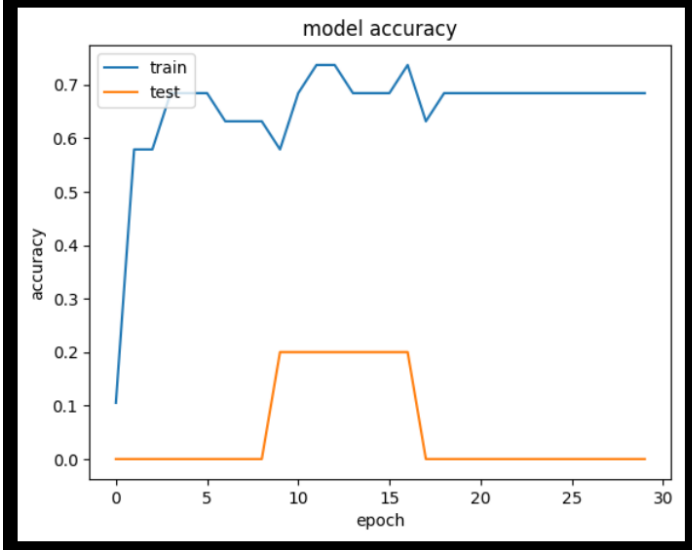
Se llama a la función CNN

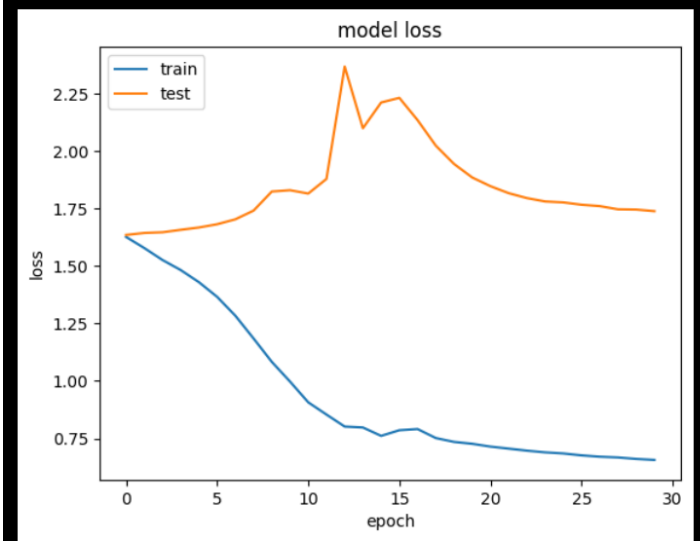
```
LSTM_model(vocab_size, embedding_matrix, 150, y_train, y_test, X_train, X_test)
```

✓ 4.0s

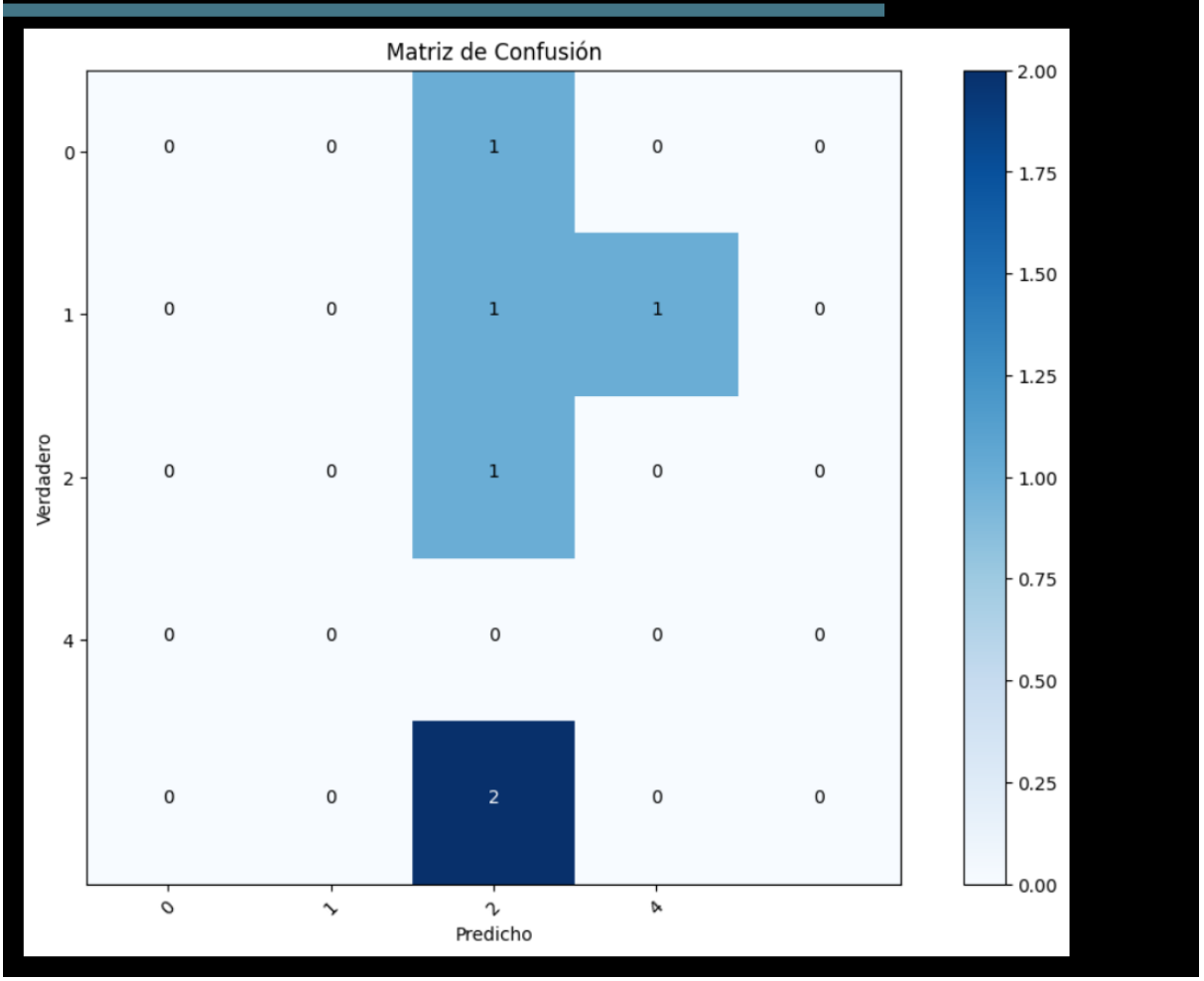
```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_7 (Embedding)     (None, 150, 300)          668400

 lstm (LSTM)                 (None, 64)                93440

 dense_9 (Dense)             (None, 5)                 325

=================================================================
Total params: 762165 (2.91 MB)
Trainable params: 762165 (2.91 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Matriz de valores para las palabras:
[[ 662  663   89 ...    0    0    0]
 [ 207   96  362 ...  786  787  788]
 [  32  852   10 ...  230   11  906]
 ...
 [ 585 2094 2095 ... 2169 2170 2171]
 [2172  338   34 ...    0    0    0]
 [   4   24 2197 ...    0    0    0]]
['Personas', 'Animales', 'Lugares', 'Series', 'Series', 'Lugares', 'Animales', 'Animales', 'Objetos', 'Personas', 'Lugares
...
2/2 [==============================] - 0s 41ms/step - loss: 0.6566 - accuracy: 0.6842 - val_loss: 1.7383 - val_accuracy: 0
1/1 [==============================] - 0s 15ms/step - loss: 1.5983 - accuracy: 0.1667
Test Loss: 1.5982526540756226
Test Accuracy: 0.1666666716337204
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...





```
Matriz de valores para las palabras:
[[ 662  663   89 ...    0    0    0]
 [ 207   96  362 ...  786  787  788]
 [  32  852   10 ...  230   11  906]
 ...
 [ 585 2094 2095 ... 2169 2170 2171]
 [2172  338   34 ...    0    0    0]
 [   4   24 2197 ...    0    0    0]]
['Personas', 'Animales', 'Lugares', 'Series', 'Series', 'Lugares', 'Animales', 'Animales', 'Objetos', 'Perso
1/1 [==============================] - 0s 174ms/step
```



Se llama a la función LSTM_model

Conclusión

¿Qué observas en la comparación de los modelos?

- El modelo DNN alcanza una precisión del 100% en el conjunto de pruebas, lo que indica un sobreentrenamiento.
- El modelo CNN alcanza una precisión del 100% en el conjunto de pruebas, lo que indica un sobreentrenamiento.
- El modelo LSTM tiene una precisión de solo 16.67% en el conjunto de pruebas, lo que dice que tiene un rendimiento mucho peor.

¿Cómo afectan los hiperparámetros de cada tipo de red neuronal?

- Los hiperparámetros, como el número de capas, unidades, funciones de activación y la longitud de secuencia máxima, pueden tener un gran impacto en el rendimiento de los modelos. Una mala elección de hiperparamentar puede hacer que el modelo falle mucho en el test.

¿Qué se complicó en el clasificador?

Sobreentrenamiento o baja calificación en el test del modelo.

¿Cómo se podría mejorar el modelo?

- Se podría agregar un dropout a DNN y CNN para evitar el sobreentrenamiento.
- Buscar la configuración óptima de los hiperparámetros
- Buscar de manera más detallada características en los textos
- Más datos