MODELO DE CLASIFICACION DE TEXTO

Bibliotecas Importadas:

1. **tensorflow as tf**: Importa TensorFlow, una biblioteca líder en aprendizaje automático y redes neuronales.

2. **Sequential** de **keras.models**: Permite la construcción secuencial de modelos de redes neuronales.

3. Capas específicas de la red:
   - **Embedding**, **LSTM**, **Dense**, **Dropout**, **BatchNormalization** de **keras.layers**: Define las capas de la red neuronal.

4. Componentes para procesamiento de texto:
   - **Tokenizer** y **pad_sequences** de **keras.preprocessing.text**: Facilita la tokenización y secuenciación de textos.

5. **l2** de **keras.regularizers**: Implementa la regularización L2 para prevenir el sobreajuste.

6. Callbacks para entrenamiento:
   - **EarlyStopping**, **ModelCheckpoint** de **keras.callbacks**: Mejora el control del entrenamiento mediante detención temprana y almacenamiento del mejor modelo.
7. Operaciones matemáticas y manipulación de datos:
   - **numpy as np**: Herramienta fundamental para operaciones matemáticas y manipulación eficiente de datos.

8. Visualización de datos:
   - **matplotlib.pyplot as plt**: Biblioteca para la generación de gráficos y visualización de resultados.

Instalacion de librerias:

pip install tensorflow keras matplotlib numpy scikit-learn

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, Dropout, BatchNormalization
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.regularizers import l2
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from text_classification_model import TextClassificationModel
from custom_callback import CustomCallback
```

Clase TextClassificationModelImplements:

Constructor (**__init__**):

- Inicializa la clase con parámetros esenciales: clases, tokenizador y datos de instrucción.

```python
def __init__(self, clases=[], tokenizer=None,instruction = []):
    # Inicialización del modelo y configuración de capas
    self.clases = clases
    self.tokenizer = tokenizer
    self.tokenizer.fit_on_texts(instruction)
```

Método **build_model**:

- Construye un modelo secuencial de red neuronal con las siguientes capas:
  - Capa de Embedding: Mapea palabras a vectores de alta dimensión.
  - Capas LSTM: Capturan patrones a largo plazo en los datos.
  - Capas Dense: Neuronas completamente conectadas con funciones de activación.
  - Capas de Dropout: Evitan el sobreajuste mediante la eliminación de conexiones aleatorias.
  - Capa de Batch Normalization: Estabiliza y acelera el aprendizaje.

```python
def build_model(self):
    self.model = Sequential([
        Embedding(input_dim=len(self.tokenizer.word_index) + 1, output_dim=64, input_length=50),  #Tamaño del vocabulario(palabras unicas),salida, longitud máx
        LSTM(units=150, return_sequences=True, kernel_regularizer=l2(0.01)),  #capturar patrones a largo plazo en los datos, Aumentar unidades(patrones) y agre
        Dropout(0.5),  # Aumentar Dropout
        LSTM(units=150, kernel_regularizer=l2(0.01)),  # Aumentar unidades y agregar regularización
        Dense(units=256, activation='relu', kernel_regularizer=l2(0.01)),  # cada neurona o unidad está conectada a cada neurona de la capa anterior.
        BatchNormalization(), #Tecnica estabilizar y acelerar el aprendizaje.
        Dense(units=128, activation='relu', kernel_regularizer=l2(0.01)),  # Cambiar activación y agregar regularización
        Dropout(0.5),  # Ajustar Dropout, técnica de regularización que se utiliza para prevenir el sobreajuste en redes neuronales.
        Dense(units=len(self.clases), activation='softmax')
    ])
    self.model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Método **train_model**:

- Prepara datos para entrenamiento, realiza tokenización y secuenciación de textos.
- Divide datos en conjuntos de entrenamiento y validación.
- Convierte etiquetas a codificación one-hot.
- Entrena el modelo con callbacks de detención temprana y almacenamiento del mejor modelo.

```python
def train_model(self, data_training_class, label_training_class, epochs=20):  # Aumentar epochs
    # Preparación de datos para el entrenamiento
    all_texts = []
    all_labels = []

    for i, clase in enumerate(self.clases):

        texts = data_training_class[i]
        labels = [i] * len(texts)
        all_texts.extend(texts)
        all_labels.extend(labels)

    # Tokenización y secuenciación de textos
    if self.tokenizer is None:
        self.tokenizer = Tokenizer(num_words=5000, oov_token="<OOV>")
        self.tokenizer.fit_on_texts(all_texts)
    sequences = self.tokenizer.texts_to_sequences(all_texts)
    padded_sequences = pad_sequences(sequences, maxlen=50, truncating='post')

    # División de datos en conjuntos de entrenamiento y validación
    x_train, x_val, y_train, y_val = train_test_split(padded_sequences, np.array(all_labels), test_size=0.3, random_state=54)

    # Convertir etiquetas a one-hot encoding
    y_train_numeric = np.array(y_train, dtype=int)
    y_train_categorical = tf.keras.utils.to_categorical(y_train_numeric - np.min(y_train_numeric), num_classes=len(self.clases))
    y_val_categorical = tf.keras.utils.to_categorical(y_val - min(y_val), num_classes=len(self.clases))

    # Entrenamiento del modelo
    print("Entrenamiento del Modelo:")
    #custom_callback = CustomCallback()
    #EarlyStopping se detendrá el entrenamiento si la pérdida en el conjunto de validación no mejora después de n épocas (patience=3)
    early_stoppin llback = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

    # Guarda el mejor modelo durante el entrenamiento
    model_checkpoint_callback = ModelCheckpoint('best_model.h5', save_best_only=True)

    self.history = self.model.fit(
        x_train, y_train_categorical,
        epochs=epochs,
        validation_data=(x_val, y_val_categorical),
        verbose=2,
        callbacks=[model_checkpoint_callback,early_stopping_callback]
    )
```

Método **predict_probability**:

- Preprocesa texto para realizar predicciones de probabilidad.
- Utiliza el modelo entrenado para predecir probabilidades para una clase específica.

```python
def predict_probability(self, texto):
    # Preprocesamiento de texto para predicción
    sequence = self.tokenizer.texts_to_sequences([texto])
    padded_sequence = pad_sequences(sequence, maxlen=50, truncating='post')

    # Predicción de probabilidades
    probabilities = self.model.predict(np.array(padded_sequence))
    probability_for_class = probabilities[0]

    return probability_for_class
```

Método **plot_training_history**:

- Genera gráficos de pérdida y precisión durante el entrenamiento y la validación.

```python
def plot_training_history(self):
    # Gráficos de pérdida
    plt.plot(self.history.history['loss'], label='Entrenamiento')
    plt.plot(self.history.history['val_loss'], label='Validación')
    plt.title('Curva de Pérdida')
    plt.xlabel('Épocas')
    plt.ylabel('Pérdida')
    plt.legend()
    plt.show()

    # Gráficos de precisión
    plt.plot(self.history.history['accuracy'], label='Entrenamiento')
    plt.plot(self.history.history['val_accuracy'], label='Validación')
    plt.title('Curva de Precisión')
    plt.xlabel('Épocas')
    plt.ylabel('Precisión')
    plt.legend()
    plt.show()
```

Técnicas Utilizadas:

- Regularización L2 en algunas capas para prevenir el sobreajuste.
- Uso de Dropout para regularización durante el entrenamiento.
- Callbacks como **EarlyStopping** y **ModelCheckpoint** para mejorar el control del entrenamiento.

Clase CustomCallback

Método **on_epoch_end**:

- Este método se ejecuta al final de cada época durante el entrenamiento del modelo.
- Verifica si la precisión (accuracy) tanto en entrenamiento como en validación es mayor o igual al valor mínimo deseado (**min_accuracy**).
- Si se cumple la condición, imprime un mensaje indicando que se ha alcanzado el accuracy mínimo y detiene el entrenamiento (**self.model.stop_training = True**).

Uso del Callback:

- Este callback se integra con el entrenamiento del modelo y se activa al final de cada época.

```python
from keras.callbacks import EarlyStopping
import tensorflow as tf

Comment Code
class CustomCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None,min_accuracy=0.95):
        print("hola")
        # Verificar si se cumple la condición de precisión
        if logs['accuracy'] >= min_accuracy and logs['val_accuracy'] >= min_accuracy:
            print(f"Alcanzado el accuracy mínimo deseado ({min_accuracy}). Deteniendo el entrenamiento.")
            self.model.stop_training = True
```

Biblioteca Importada:

1. **ABC** y **abstractmethod** de **abc**: Importa la clase **ABC** (Abstract Base Class) y el decorador **abstractmethod** para crear una interfaz abstracta.

Clase TextClassificationModel (Interfaz Abstracta):

Método Abstracto **build_model**:

- Método que debe ser implementado por las clases concretas.
- Define la construcción del modelo de clasificación de texto, incluyendo la configuración de capas y parámetros.

Método Abstracto **train_model**:

- Método que debe ser implementado por las clases concretas.
- Define el entrenamiento del modelo utilizando datos de entrenamiento y etiquetas. Puede incluir configuraciones específicas del modelo, como número de épocas.

Método Abstracto **predict_probability**:

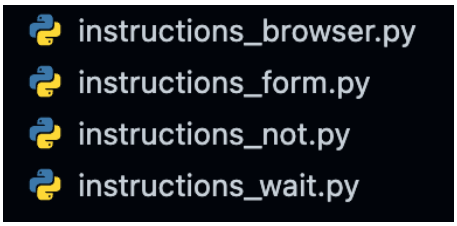
- Método que debe ser implementado por las clases concretas.
- Define la predicción de probabilidades para un texto dado. Este método puede ser utilizado para la clasificación de nuevos textos.

Método Abstracto **plot_training_history**:

- Método que debe ser implementado por las clases concretas.
- Define la generación de gráficos para visualizar la historia de entrenamiento del modelo, incluyendo pérdida y precisión a lo largo de las épocas.

```
from abc import ABC, abstractmethod

Comment Code
class TextClassificationModel(ABC):

    @abstractmethod
    def build_model(self):
        pass

    @abstractmethod
    def train_model(self, data_training, labels, epochs=20):
        pass

    @abstractmethod
    def predict_probability(self, text):
        pass

    @abstractmethod
    def plot_training_history(self):
        pass
```

Datos de entrenamiento:




```
instructions_browser = [
    "Abre el navegador Edge y ve a https://www.paginanueva.com",
    "Accede a https://www.paginanueva.com desde el navegador Chrome",
    "Abre Firefox y navega a https://www.paginanueva.com",
    "Explora https://www.paginanueva.com utilizando Microsoft Edge",
    "Ve a https://www.paginanueva.com mediante el navegador web Chrome",
    "Utiliza Firefox para acceder a https://www.paginanueva.com",
    "Navega hacia https://www.paginanueva.com con el navegador Edge",
    "Abre el navegador Chrome y dirígete a https://www.paginanueva.com",
    "Visita https://www.paginanueva.com a través de Firefox",
    "Accede a https://www.paginanueva.com usando el navegador web Edge",
    "Explora la página https://www.paginanueva.com desde Microsoft Edge",
    "Inicia Chrome y visita https://www.paginanueva.com",
```

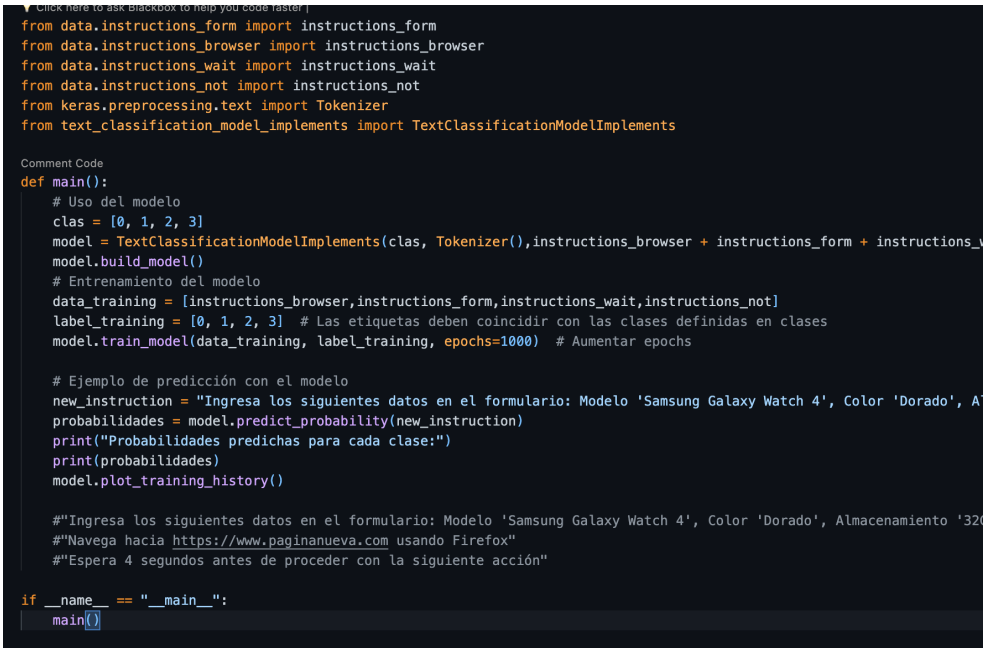
```
instructions_form = [
    "Completa el formulario con los siguientes datos: Nombre 'Roberto Hernández', Edad '33', Ciudad 'Caracas', Email 'rober
    "Ingresa la información en el formulario: Empresa 'SoftSolutions', Cargo 'Desarrollador Frontend', Departamento 'Desarr
    "Rellena el formulario con: Producto 'Smartphone Xiaomi Mi 12', Cantidad '1', Precio '699.99 USD', FechaCompra '2023-03
    "Completa los campos del formulario con: Destino 'Ciudad del Cabo', FechaSalida '2023-08-08', FechaRegreso '2023-08-18'
    "Llena el formulario con: Título 'Proyecto de Marketing Digital', Descripción 'Campaña en redes sociales para lanzamien
    "Ingresa los datos en el formulario: Platillo 'Ensalada César', Tipo 'Entrada', Precio '8.99 USD', Restaurante 'FreshSa
    "Rellena los campos requeridos del formulario: NombreUsuario 'Usuario505', Contraseña 'ContraseñaFuerte5567', PreguntaS
    "Completa el formulario de registro con: Nombre 'Sofía Díaz', Usuario 'sofia505', Contraseña 'ContraseñaSegura5667', Em
    "Ingresa los siguientes datos en el formulario: Modelo 'LG OLED TV', Tamaño '55 pulgadas', Precio '1299.99 USD', Tienda
    "Rellena los campos del formulario con: Dirección 'Calle Principal 678', CódigoPostal '70001', País 'Venezuela', Ciudad
    "Completa el formulario con los siguientes datos: Nombre 'Diego Mendoza', Edad '28', Ciudad 'Buenos Aires', Email 'dieg
```

```
instructions_wait = [
    "Espera 5 segundos",
    "Aguarda 3 segundos antes de continuar",
    "Realiza una pausa de 4 segundos",
    "Detén la ejecución durante 2 segundos",
    "Espera 6 segundos antes de proceder",
    "Haz una pausa de 7 segundos",
    "Aguarda 3 segundos antes de realizar la siguiente acción",
    "Realiza una espera de 5 segundos",
    "Detén la actividad durante 4 segundos",
```

```
instructions_not = ['manzana', 'gato', 'computadora', 'viaje', 'feliz', 'océano', 'libro', 'montaña', 'sol', 'familia',
          'tiempo', 'canción', 'correr', 'arte', 'ciudad', 'verano', 'perro', 'sueño', 'chocolate', 'naturaleza',
          'amistad', 'aventura', 'cine', 'color', 'lluvia', 'risa', 'estrella', 'aire', 'flor', 'energía',
          'explorar', 'silencio', 'música', 'playa', 'amor', 'creatividad', 'esperanza', 'fuego', 'viajero',
          'misterio', 'reflejo', 'sabor', 'pintura', 'serenidad', 'relámpago', 'invierno', 'sorpresa', 'paz']
```

Archivo main

- **Definición de Clases:**
  - **clases**: Lista que define las clases para la clasificación.
- **Inicialización del Modelo:**
  - **model**: Instancia de **TextClassificationModelImplements** con clases, un tokenizador y datos de instrucciones concatenados.
- **Construcción y Entrenamiento del Modelo:**
  - **model.build_model()**: Construye el modelo de clasificación de texto.
  - **model.train_model(data_training, label_training, epochs=1000)**: Entrena el modelo con datos y etiquetas especificadas.
- **Predicción y Visualización:**
  - **new_instruction**: Ejemplo de instrucción para predecir.
  - **model.predict_probability(new_instruction)**: Predice probabilidades para la nueva instrucción.
  - **model.plot_training_history()**: Visualiza la historia de entrenamiento.
- **Ejecución del Programa:**
  - **if __name__ == "__main__":**: Ejecuta la función **main()** si el script es ejecutado directamente.



```
Click here to ask Blackbox to help you code faster
from data.instructions_form import instructions_form
from data.instructions_browser import instructions_browser
from data.instructions_wait import instructions_wait
from data.instructions_not import instructions_not
from keras.preprocessing.text import Tokenizer
from text_classification_model_implements import TextClassificationModelImplements

Comment Code
def main():
    # Uso del modelo
    clas = [0, 1, 2, 3]
    model = TextClassificationModelImplements(clas, Tokenizer(),instructions_browser + instructions_form + instructions_
    model.build_model()
    # Entrenamiento del modelo
    data_training = [instructions_browser,instructions_form,instructions_wait,instructions_not]
    label_training = [0, 1, 2, 3]  # Las etiquetas deben coincidir con las clases definidas en clases
    model.train_model(data_training, label_training, epochs=1000)  # Aumentar epochs

    # Ejemplo de predicción con el modelo
    new_instruction = "Ingresa los siguientes datos en el formulario: Modelo 'Samsung Galaxy Watch 4', Color 'Dorado', A
    probabilidades = model.predict_probability(new_instruction)
    print("Probabilidades predichas para cada clase:")
    print(probabilidades)
    model.plot_training_history()

    #"Ingresa los siguientes datos en el formulario: Modelo 'Samsung Galaxy Watch 4', Color 'Dorado', Almacenamiento '32
    #"Navega hacia https://www.paginanueva.com usando Firefox"
    #"Espera 4 segundos antes de proceder con la siguiente acción"

if __name__ == "__main__":
    main()
```

Ejecucion

El enunciado que se probara es `"Ingresa los siguientes datos en el formulario: Modelo 'Samsung Galaxy Watch 4', Color 'Dorado', Almacenamiento '32GB', Precio '249.99 USD', Tienda 'GizmoHub'"`

Deberia de clasificarse en el segunda clase con mas alto porcentaje de precision, ya que pertenece a instructions_form

```
data_training = [instructions_browser,instructions_form,instructions_wait,instructions_not]
```

Pare ejecutar se escribe python y el nombre o path del archivo

```
python /Users/hugocapuchino/Desktop/Tensorflow/main.py
```

```
<python /Users/hugocapuchino/Desktop/Tensorflow/main.py
Entrenamiento del Modelo:
Epoch 1/1000
/Users/hugocapuchino/Desktop/Tensorflow/venv/lib/python3.11/site-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
5/5 - 2s - loss: 8.0550 - accuracy: 0.5725 - val_loss: 7.7483 - val_accuracy: 0.4000 - 2s/epoch - 335ms/step
Epoch 2/1000
5/5 - 0s - loss: 6.8570 - accuracy: 0.8116 - val_loss: 7.0419 - val_accuracy: 0.7833 - 254ms/epoch - 51ms/step
Epoch 3/1000
5/5 - 0s - loss: 5.8695 - accuracy: 0.9203 - val_loss: 6.4305 - val_accuracy: 0.8667 - 255ms/epoch - 51ms/step
Epoch 4/1000
5/5 - 0s - loss: 5.1665 - accuracy: 0.9855 - val_loss: 5.8730 - val_accuracy: 1.0000 - 282ms/epoch - 56ms/step
Epoch 5/1000
5/5 - 0s - loss: 4.5944 - accuracy: 1.0000 - val_loss: 5.3202 - val_accuracy: 1.0000 - 266ms/epoch - 53ms/step
```

```
Probabilidades predichas para cada clase:
[0.01186326 0.95363873 0.01644472 0.01805333]
```

Se clasifico en la segunda clase con mayor porcentaje de precision

Informacion durante el entrenamiento.