Universidad de Oriente

Núcleo Nueva Esparta

Escuela de Ingeniería y Ciencias Aplicadas

Departamento de Informática

Introducción a la Inteligencia Artificial – 2304574

Analizador Sintáctico de Expresiones Algebraicas

Docente:

Lic. Samuel Rojas

Integrantes:

Br. José Silva. CI: 30230054

Br. Isaac Hernández. CI: 30563299

Br. Emanuel Aponte. CI: 29789773

Guatamare, 02/2025

Documentación del Proyecto

¿Qué es una expresión algebraica?: Son términos matemáticos que contienen números y letras. En combinación con los símbolos de las operaciones matemáticas para obtener fórmulas o ecuaciones, a partir de descripciones hechas mediante palabras. A su vez esas letras pueden ser sumadas, restadas, multiplicadas o divididas por otros números, los cuales pueden ser explícitos o representados también por letras.

Clasificación de las expresiones algebraicas: se establece según su número de términos. En general, se dividen en dos: monomios y polinomios.

- **Monomios:** Las expresiones algebraicas llamadas monomios son aquellas que están compuestas por un solo termino. Ejemplo: $2x^2$ $2x^2y^3z$.
- **Polinomios:** Los polinomios son una clasificación de expresiones algebraicas que según la cantidad de términos por la que está formada cambia su nombre: binomio, trinomio, etc.… Estas expresiones algebraicas en general se componen por dos o más términos, es decir, por más de un monomio:
  - **Binomios:** se componen por dos términos separados por un operador matemático. Ejemplo: $a^4 b^5 + 3 a^2 b^2 c^7$
  - **Trinomios:** Cuando se denomina trinomio, es una expresión algebraica compuesta por tres monomios: $ab^3 + 5a^2 b^7 m - 35 abx^5$

¿Qué hace el proyecto?

Este proyecto es un programa que utiliza inteligencia artificial para analizar expresiones algebraicas (como 3x + 2y o x^2 − 4x - 4) y clasificarlas en dos categorías:

1. **Tipo de expresión:** Por ejemplo, si es un monomio, binomio, trinomio, polinomio y si es una ecuación.
2. **Operación principal:** Si es una suma, resta, multiplicación o división.

El programa "aprende" a hacer esto a partir de un conjunto de datos que contiene ejemplos de expresiones algebraicas y sus clasificaciones

Estructura de carpeta

```
Analizador_sintacticoEA/            # Carpeta raíz del proyecto
│
├── data/                    # Carpeta para almacenar datos
│   ├── __init__.py
│   ├── datos.csv              # Archivo con expresiones y etiquetas
│   ├── generate_data.py          # Script para cargar datos desde el CSV
│   └── preprocess_data.py        # Script para preprocesar los datos
├── docs/                   # Manual de usuario
├── model/                    # Carpeta para el modelo de red neuronal
│   ├── __init__.py
│   ├── build_model.py            # Script para construir el modelo
│   └── train_model.py            # Script para entrenar el modelo
│
├── utils/                  # Carpeta para utilidades adicionales
│   ├── __init__.py
│   ├── predict.py              # Script para hacer predicciones
```

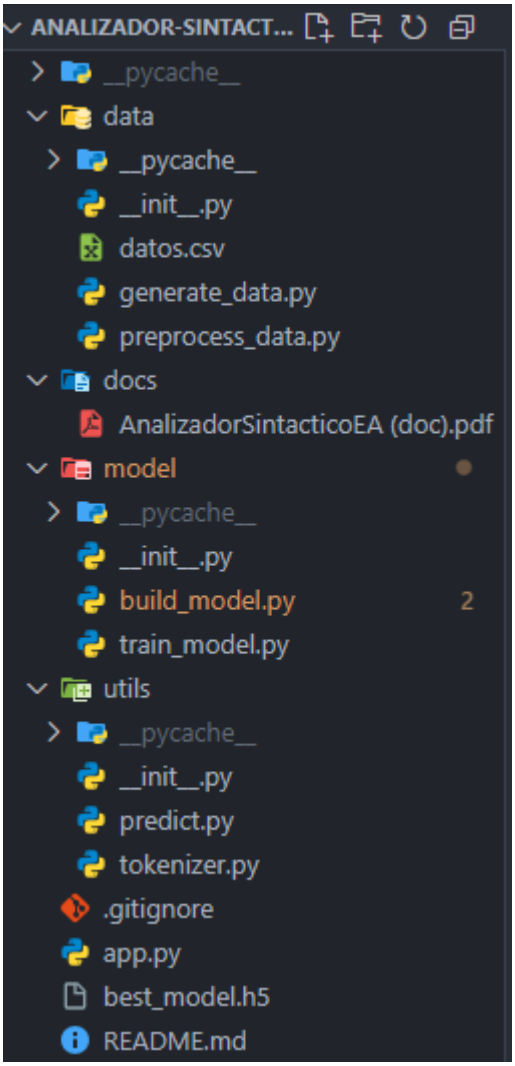
│ └── tokenizer.py # Script para tokenización y normalización

├──. gitignore # Archivo de git

├── app.py # Archivo principal que ejecuta el programa

├── best_model.h5 # guarda el modelo entrenado en cada llamada

└── README.md # Documentación del proyecto (opcional)

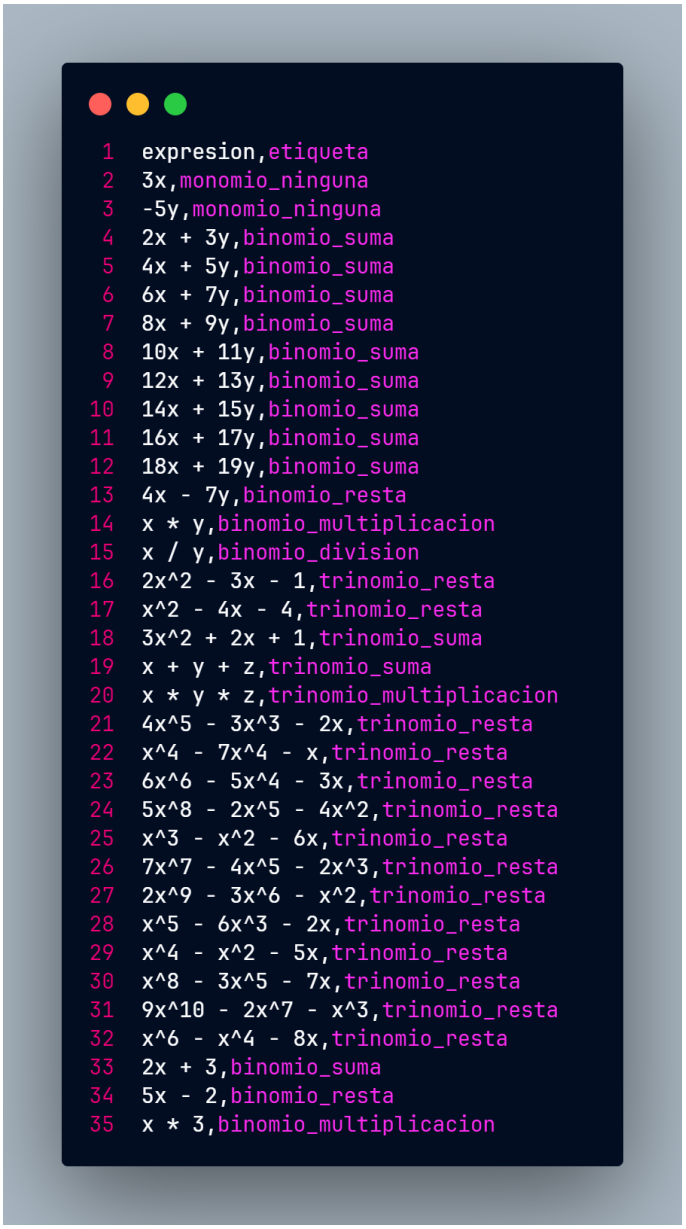


¿Cómo Funciona?

- **data/datos.csv:** Este archivo contiene los datos que el programa debe usar para aprender. Es como un libro de ejemplos que tiene 2 columnas:

    - ***expresión:*** una expresión algebraica (por ejemplo, 3x, 2x + 3y).

    - ***etiqueta:*** la clasificación de la expresión (monomio_ninguna, binomio_suma)



```
expresion,etiqueta
3x,monomio_ninguna
-5y,monomio_ninguna
2x + 3y,binomio_suma
4x + 5y,binomio_suma
6x + 7y,binomio_suma
8x + 9y,binomio_suma
10x + 11y,binomio_suma
12x + 13y,binomio_suma
14x + 15y,binomio_suma
16x + 17y,binomio_suma
18x + 19y,binomio_suma
4x - 7y,binomio_resta
x * y,binomio_multiplicacion
x / y,binomio_division
2x^2 - 3x - 1,trinomio_resta
x^2 - 4x - 4,trinomio_resta
3x^2 + 2x + 1,trinomio_suma
x + y + z,trinomio_suma
x * y * z,trinomio_multiplicacion
4x^5 - 3x^3 - 2x,trinomio_resta
x^4 - 7x^4 - x,trinomio_resta
6x^6 - 5x^4 - 3x,trinomio_resta
5x^8 - 2x^5 - 4x^2,trinomio_resta
x^3 - x^2 - 6x,trinomio_resta
7x^7 - 4x^5 - 2x^3,trinomio_resta
2x^9 - 3x^6 - x^2,trinomio_resta
x^5 - 6x^3 - 2x,trinomio_resta
x^4 - x^2 - 5x,trinomio_resta
x^8 - 3x^5 - 7x,trinomio_resta
9x^10 - 2x^7 - x^3,trinomio_resta
x^6 - x^4 - 8x,trinomio_resta
2x + 3,binomio_suma
5x - 2,binomio_resta
x * 3,binomio_multiplicacion
```

- **data/generate_data.py:** contiene una función para cargar los datos desde el archivo CSV.
    - Función load_data_from_csv(file_path)

        - **Propósito:** Carga los datos de entrenamiento desde un archivo CSV.
        - **Entrada:** la ruta al archivo CSV (file_path)
        - **Salida:**
            - **Expresiones:** una lista de expresiones algebraicas
            - **Etiquetas:** una lista de etiquetas correspondientes a las expresiones

        - **¿Cómo Funciona?**

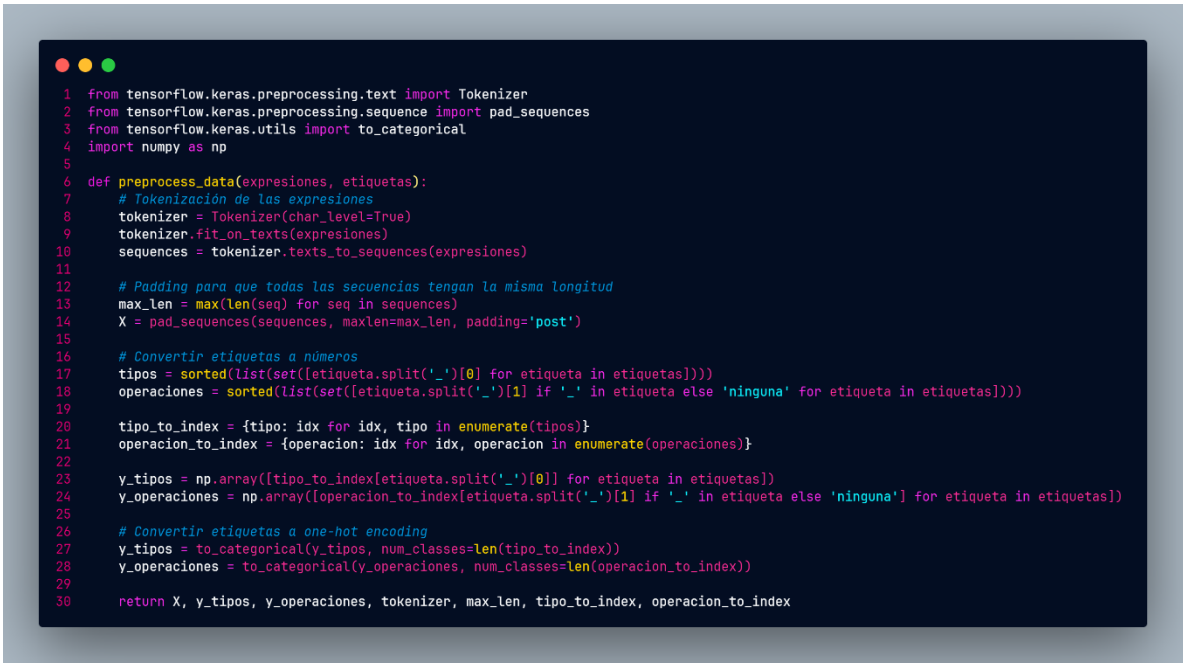
            1. Usa pandas para leer el archivo CSV.
            2. Extrae las columnas "***expresión"*** y ***"etiqueta"*** como listas.
            3. Devuelve las listas de expresiones y etiquetas

```python
import pandas as pd

def load_data_from_csv(file_path):
    """
    Carga los datos de entrenamiento desde un archivo CSV.

    Args:
        file_path (str): Ruta al archivo CSV.

    Returns:
        expresiones (list): Lista de expresiones algebraicas.
        etiquetas (list): Lista de etiquetas correspondientes.
    """
    # Leer el archivo CSV
    df = pd.read_csv(file_path)

    # Obtener las columnas como listas
    expresiones = df['expresion'].tolist()
    etiquetas = df['etiqueta'].tolist()

    return expresiones, etiquetas
```

- **data/preprocess_data.py:** este archivo se encarga de preprocesar los datos antes de entrenar el modelo. Preprocesar hace referencia a que procesa sus datos de entrada para producir una salida que se utiliza como entrada en otro programa, es decir, convierte los datos brutos en un formato que el computador pueda procesar.

    - **Tokenización:** convierte las expresiones en números (por ejmplo, 3x se convierte en [3, x])
    - **Padding:** asegura que todas las expresiones tengan la misma longitud añadiendo ceros al final si es necesario

- **Conversión de etiquetas:** convierte las etiquetas (como ***monomio_ninguna***) en números y luego en un formato especial llamado "***one-hot-encoding***", que es más fácil de procesar para el programa



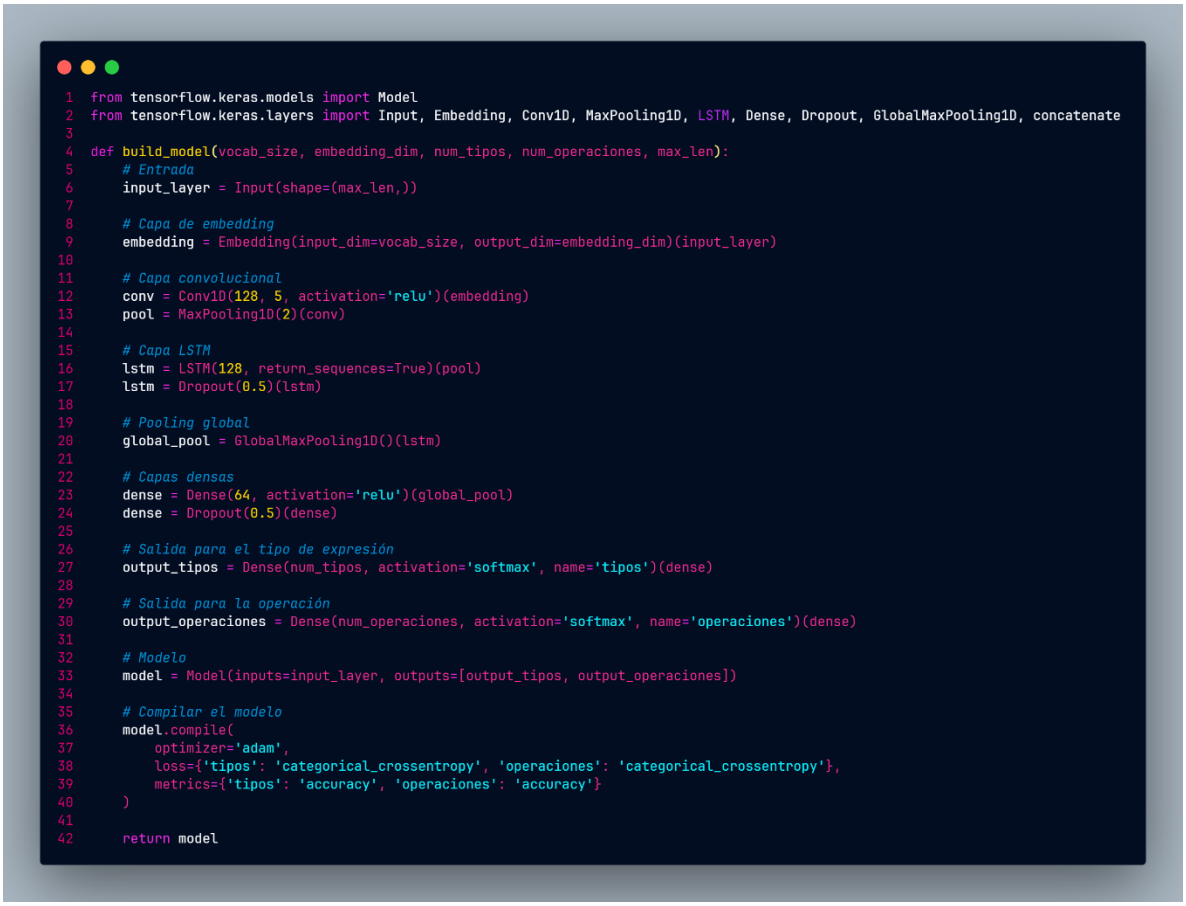
```python
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
import numpy as np

def preprocess_data(expresiones, etiquetas):
    # Tokenización de las expresiones
    tokenizer = Tokenizer(char_level=True)
    tokenizer.fit_on_texts(expresiones)
    sequences = tokenizer.texts_to_sequences(expresiones)

    # Padding para que todas las secuencias tengan la misma longitud
    max_len = max(len(seq) for seq in sequences)
    X = pad_sequences(sequences, maxlen=max_len, padding='post')

    # Convertir etiquetas a números
    tipos = sorted(list(set([etiqueta.split('_')[0] for etiqueta in etiquetas])))
    operaciones = sorted(list(set([etiqueta.split('_')[1] if '_' in etiqueta else 'ninguna' for etiqueta in etiquetas])))

    tipo_to_index = {tipo: idx for idx, tipo in enumerate(tipos)}
    operacion_to_index = {operacion: idx for idx, operacion in enumerate(operaciones)}

    y_tipos = np.array([tipo_to_index[etiqueta.split('_')[0]] for etiqueta in etiquetas])
    y_operaciones = np.array([operacion_to_index[etiqueta.split('_')[1] if '_' in etiqueta else 'ninguna'] for etiqueta in etiquetas])

    # Convertir etiquetas a one-hot encoding
    y_tipos = to_categorical(y_tipos, num_classes=len(tipo_to_index))
    y_operaciones = to_categorical(y_operaciones, num_classes=len(operacion_to_index))

    return X, y_tipos, y_operaciones, tokenizer, max_len, tipo_to_index, operacion_to_index
```

- **model/build_model.py:** este construye el "*cerebro*" del programa, es decir, aquí se construye la red neuronal.

  - **Capa de entrada:** recibe las expresiones convertidas en números
  - **Embedding:** convierte esos números en vectores (lista de números) que representan mejor la información
  - **Capas convolucionales y LSTM:** estas capas ayudan a extraer características importantes de las expresiones
  - **Capas densas:** procesan la información para tomar decisiones

❖ **Salidas:** El programa predice dos cosas:

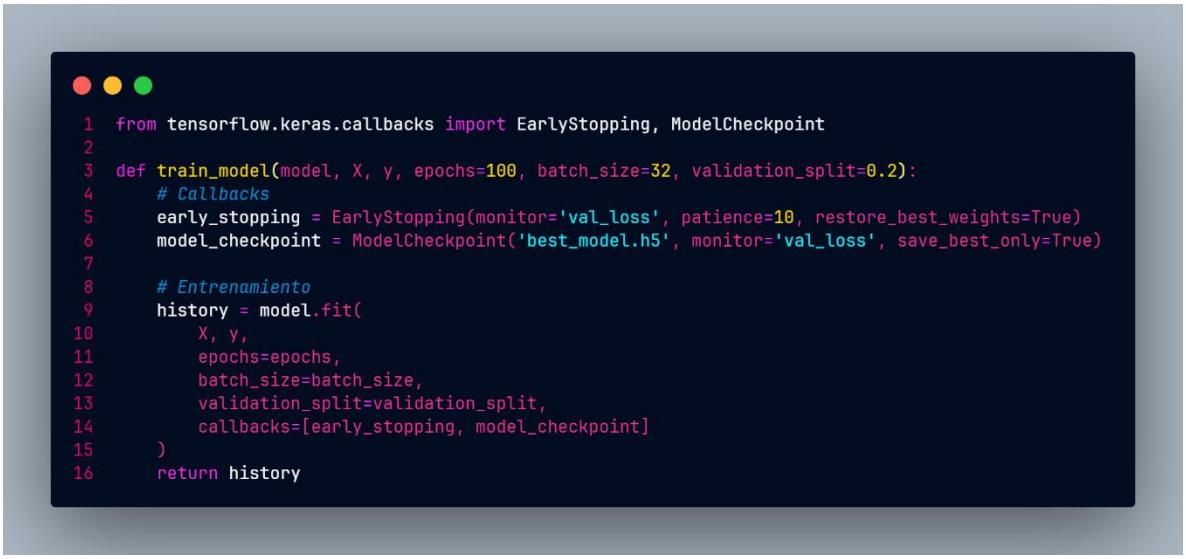
- El tipo de expresión (por ejemplo, monomio, binomio).
- La operación principal (por ejemplo, suma, resta)



```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Conv1D, MaxPooling1D, LSTM, Dense, Dropout, GlobalMaxPooling1D, concatenate

def build_model(vocab_size, embedding_dim, num_tipos, num_operaciones, max_len):
    # Entrada
    input_layer = Input(shape=(max_len,))

    # Capa de embedding
    embedding = Embedding(input_dim=vocab_size, output_dim=embedding_dim)(input_layer)

    # Capa convolucional
    conv = Conv1D(128, 5, activation='relu')(embedding)
    pool = MaxPooling1D(2)(conv)

    # Capa LSTM
    lstm = LSTM(128, return_sequences=True)(pool)
    lstm = Dropout(0.5)(lstm)

    # Pooling global
    global_pool = GlobalMaxPooling1D()(lstm)

    # Capas densas
    dense = Dense(64, activation='relu')(global_pool)
    dense = Dropout(0.5)(dense)

    # Salida para el tipo de expresión
    output_tipos = Dense(num_tipos, activation='softmax', name='tipos')(dense)

    # Salida para la operación
    output_operaciones = Dense(num_operaciones, activation='softmax', name='operaciones')(dense)

    # Modelo
    model = Model(inputs=input_layer, outputs=[output_tipos, output_operaciones])

    # Compilar el modelo
    model.compile(
        optimizer='adam',
        loss={'tipos': 'categorical_crossentropy', 'operaciones': 'categorical_crossentropy'},
        metrics={'tipos': 'accuracy', 'operaciones': 'accuracy'}
    )

    return model
```

- **model/train_model.py:** Este archivo se encarga de entrenar el modelo con los datos.

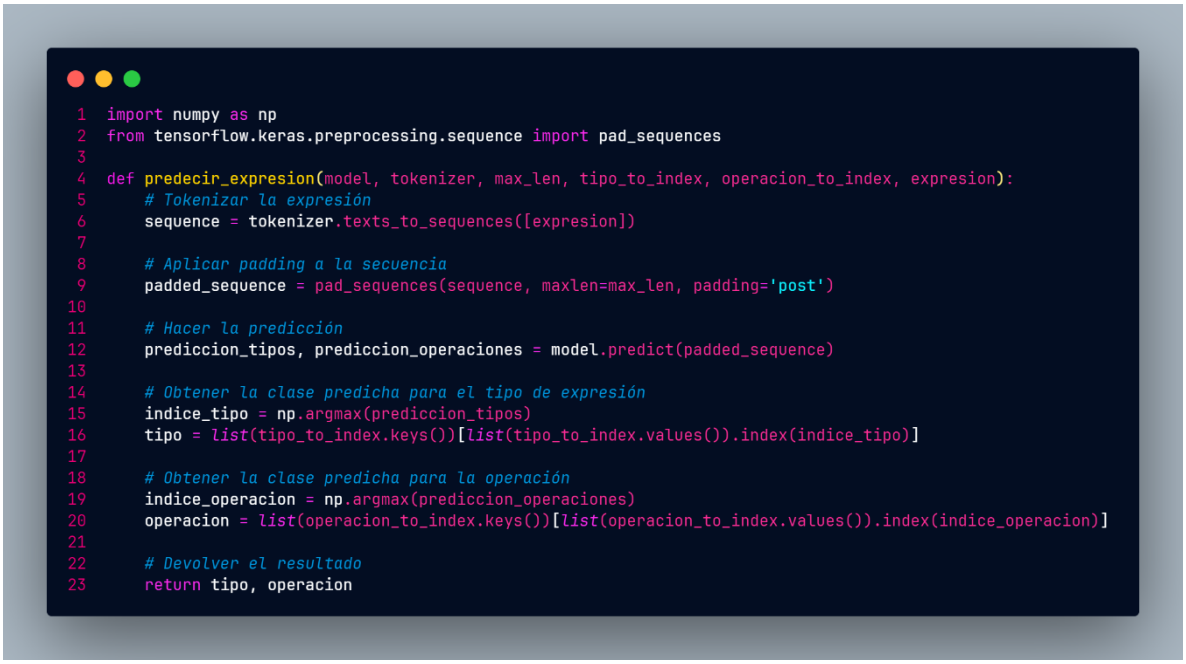
  - **Entrenamiento:** usa los datos preprocesados para enseñarle al modelo cómo clasificar las expresiones
  - **Callback:** durante el entrenamiento, el programa verifica si está mejorando y guarda el mejor modelo



```python
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

def train_model(model, X, y, epochs=100, batch_size=32, validation_split=0.2):
    # Callbacks
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
    model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)

    # Entrenamiento
    history = model.fit(
        X, y,
        epochs=epochs,
        batch_size=batch_size,
        validation_split=validation_split,
        callbacks=[early_stopping, model_checkpoint]
    )
    return history
```

- **utils/predict.py:** Este archivo permite usar el modelo entrenado para hacer predicciones

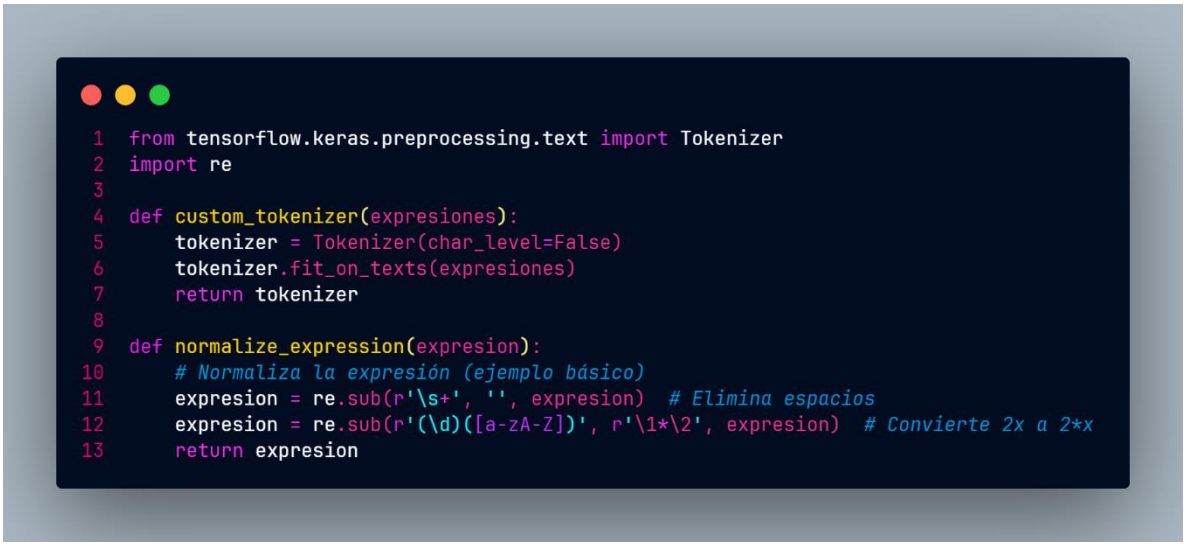
  - **Tokenización:** Convierte una nueva expresión en números
  - **Predicción:** usa el modelo para predecir el tipo y la operación de la expresión
  - **Resultado:** muestra la clasificación (por ejemplo, “la expresión 3x + 2y” es un binomio con operación de suma)



```python
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences

def predecir_expresion(model, tokenizer, max_len, tipo_to_index, operacion_to_index, expresion):
    # Tokenizar la expresión
    sequence = tokenizer.texts_to_sequences([expresion])

    # Aplicar padding a la secuencia
    padded_sequence = pad_sequences(sequence, maxlen=max_len, padding='post')

    # Hacer la predicción
    prediccion_tipos, prediccion_operaciones = model.predict(padded_sequence)

    # Obtener la clase predicha para el tipo de expresión
    indice_tipo = np.argmax(prediccion_tipos)
    tipo = list(tipo_to_index.keys())[list(tipo_to_index.values()).index(indice_tipo)]

    # Obtener la clase predicha para la operación
    indice_operacion = np.argmax(prediccion_operaciones)
    operacion = list(operacion_to_index.keys())[list(operacion_to_index.values()).index(indice_operacion)]

    # Devolver el resultado
    return tipo, operacion
```

- **utils/tokenizer.py:** contiene funciones adicionales para procesar las expresiones.

  - **Tokenización personalizada:** convierte las expresiones en números de una manera específica.
  - **Normalización:** Limpia las expresiones (por ejemplo, elimina espacios y convierte **2x** en **2*x**).



```python
from tensorflow.keras.preprocessing.text import Tokenizer
import re

def custom_tokenizer(expresiones):
    tokenizer = Tokenizer(char_level=False)
    tokenizer.fit_on_texts(expresiones)
    return tokenizer

def normalize_expression(expresion):
    # Normaliza la expresión (ejemplo básico)
    expresion = re.sub(r'\s+', '', expresion)  # Elimina espacios
    expresion = re.sub(r'(\d)([a-zA-Z])', r'\1*\2', expresion)  # Convierte 2x a 2*x
    return expresion
```

➢ **main.py:** archivo principal que ejecuta todo el programa

- ❖ **Carga los datos:** carga las expresiones y etiquetas
- ❖ **Preprocesa los datos:** prepara los datos en el formato correcto
- ❖ **Construye el modelo:** construye la red neuronal
- ❖ **Entrena el modelo:** usa "***train_model.py***" para enseñarle el modelo cómo clasificar
- ❖ **Hace predicciones:** usa "***predict.py***" para clasificar una nueva expresión ingresada por el usuario

```python
from data.generate_data import load_data_from_csv
from data.preprocess_data import preprocess_data
from model.build_model import build_model
from model.train_model import train_model
from utils.predict import predecir_expresion

def main():
    # Cargar datos desde el archivo CSV
    file_path = "data/datos.csv"  # Ruta al archivo CSV
    expresiones, etiquetas = load_data_from_csv(file_path)

    # Preprocesar datos
    X, y_tipos, y_operaciones, tokenizer, max_len, tipo_to_index, operacion_to_index = preprocess_data(expresiones, etiquetas)

    # Construir el modelo
    vocab_size = len(tokenizer.word_index) + 1
    embedding_dim = 64
    num_tipos = len(tipo_to_index)
    num_operaciones = len(operacion_to_index)
    model = build_model(vocab_size, embedding_dim, num_tipos, num_operaciones, max_len)

    # Entrenar el modelo
    train_model(model, X, {'tipos': y_tipos, 'operaciones': y_operaciones})

    # Hacer una predicción
    print("Modelo entrenado. Haciendo una predicción...")
    expresion_test = str(input("Ingrese una expresión algebraica: "))
    tipo, operacion = predecir_expresion(model, tokenizer, max_len, tipo_to_index, operacion_to_index, expresion_test)
    print(f"La expresión '{expresion_test}' es clasificada como: {tipo}, Operacion: {operacion}")

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

- **¿Qué tipo de entrenamiento usa el programa?**

  El programa utiliza el aprendizaje supervisado que es un subconjunto del ***machine learning*** que consiste en la deducción de información a partir de datos de entrenamiento. Estos datos se clasifican en dos secciones: **datos de entrenamiento y datos de prueba.** Los datos de entrenamiento se utilizan para entrenar a un modelo, y los datos de prueba son los que se usan para determinar la eficacia del modelo creado.

  El objetivo del aprendizaje supervisado es crear un programa que sea capaz de resolver cualquier variable de entrada luego de ser sometido a un proceso de entrenamiento, en pocas palabras, es aquel método que se refiere a la generación de modelos para predecir resultados basándose en ejemplos históricos de dichas variables.

Referencias

- Ceupe (s.f). Aprendizaje supervisado: Qué es, tipos y ejemplo. https://www.ceupe.com/blog/aprendizaje-supervisado.html

- Hiatoriadelaempresa.com (s.f). ¿Qué es el preprocesamiento de datos y quién lo utiliza? https://historiadelaempresa.com/que-es-el-preprocesamiento-de-datos

- Morales, M. (18 de agosto de 2023). Clasificación de las expresiones algebraicas: ¿Qué es y cómo funcionan? Conaliteg SEP. https://libros-conaliteg-sep.com.mx/clasificacion-de-expresiones-algebraicas-que-es-y-como-funciona/

- Medina, H (s.f). Expresiones algebraicas. Enciclopedia Iberoamericana. https://enciclopediaiberoamericana.com/expresiones-algebraicas/#:~:text=Una%20expresión%20algebraica%20es%20una%20expresión%20compuesta%20por,y%20división%2C%20además%20de%20la%20potenciación%20y%20radicación.

- Zapata, F (28 de abril de 2020). ¿Qué son las expresiones algebraicas y cuáles son más frecuentes? Lifeder. https://www.lifeder.com/expresiones-algebraicas/