EASY TRAIN AI



*Código Fuente*

**EasyTrainAI**

**Autores:**

1 Marco Fidel Mayta Quispe

2 Fred Torres Cruz

3 Wladimir Aldo Carlosviza Amanqui

4 Luz Bella Valenzuela Narvaez

**ÍNDICE**

**04**

**Código Fuente**

**05**

**Archivo: app.r**

**1. CÓDIGO FUENTE:**

EasyTrainAI ha sido desarrollada utilizando Python y el framework Streamlit con el objetivo de proporcionar una solución accesible y sencilla para entrenar modelos de redes neuronales con datos personalizados. Esta aplicación permite a los usuarios cargar datos, configurar y entrenar modelos de redes neuronales, visualizar métricas de rendimiento y realizar predicciones en tiempo real. La flexibilidad de la aplicación hace que sea útil tanto para principiantes como para profesionales que necesitan construir y evaluar modelos de inteligencia artificial de forma rápida y eficiente.

El desarrollo de EasyTrainAI se basó en tecnologías modernas ampliamente adoptadas en el campo de la ciencia de datos y el aprendizaje automático. Python fue la elección natural para el desarrollo, debido a su versatilidad y a la abundancia de bibliotecas disponibles para machine learning. Streamlit proporciona una interfaz web fácil de usar, que permite a los usuarios interactuar con la aplicación de manera sencilla y sin necesidad de escribir código. Para la carga y manipulación de los datos, se utilizan las bibliotecas pandas y numpy, las cuales son esenciales para trabajar con grandes volúmenes de datos en formato tabular.

El núcleo del entrenamiento de redes neuronales en la aplicación utiliza TensorFlow y Keras, dos de las bibliotecas más robustas y populares para el desarrollo de modelos de deep learning. Estas bibliotecas permiten a los usuarios definir la arquitectura de la red neuronal de manera flexible, eligiendo el número de capas ocultas, la cantidad de neuronas por capa y las funciones de activación. Además, los usuarios pueden entrenar los modelos y ver las métricas de rendimiento, como el coeficiente de determinación (R²) y la raíz cuadrada del error cuadrático medio (RMSE), que se calculan utilizando las funciones de scikit-learn.

Una de las características más destacadas de EasyTrainAI es su capacidad para generar gráficos que muestran el progreso del entrenamiento, como la pérdida durante las épocas, y comparar las predicciones del modelo con los valores reales. Esto se logra utilizando Matplotlib, una librería que permite crear visualizaciones de alta calidad. Además, los usuarios pueden descargar los modelos entrenados y los escaladores de datos en formato binario para su reutilización posterior, facilitando la integración de los modelos en otras aplicaciones o sistemas.

El flujo de trabajo de la aplicación comienza con la carga de los datos por parte del usuario, que puede estar en formatos CSV, TXT o Excel. Después de la carga, el usuario puede seleccionar las variables independientes y dependientes, con la opción de estandarizar los datos utilizando StandardScaler. Luego, los usuarios configuran el modelo seleccionando el número de capas ocultas, neuronas y funciones de activación. Una vez completado el entrenamiento, se muestran las métricas de rendimiento y gráficos interactivos que permiten evaluar el comportamiento del modelo.

La aplicación también ofrece la posibilidad de cargar modelos previamente entrenados, lo que permite realizar predicciones en tiempo real con nuevos datos ingresados por el usuario. Esta funcionalidad es extremadamente útil en casos de uso donde se requieren predicciones continuas o en tiempo real.

Con su interfaz fácil de usar y su capacidad para gestionar todo el ciclo de vida de un modelo de inteligencia artificial, EasyTrainAI se presenta como una solución versátil para el desarrollo rápido de modelos de redes neuronales sin necesidad de una configuración técnica compleja.

ARCHIVO: APP.PY

import streamlit as st

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

import tensorflow as tf

from tensorflow.keras.models import Sequential, load\_model

from tensorflow.keras.layers import Dense

from sklearn.metrics import r2\_score, mean\_squared\_error

import joblib

import matplotlib.pyplot as plt

# Configuración de la página

st.set\_page\_config(page\_title="IA para Predicción de Sensores", page\_icon="🧠", layout="wide")

# Título y descripción de la aplicación

st.title("🧠 IA para Predicción de Sensores")

st.markdown("""

    Esta aplicación utiliza redes neuronales para predecir valores de sensores basándose en datos históricos.

    Sube tu archivo de datos, configura el modelo, entrénalo y realiza predicciones en tiempo real.

    \*\*Características principales:\*\*

    - Carga de datos en formato CSV, TXT o XLSX

    - Personalización de variables de entrada y salida

    - Configuración flexible de la arquitectura de la red neuronal

    - Visualización de métricas de rendimiento y gráficos de entrenamiento

    - Predicción en tiempo real con nuevos datos

    - Carga de modelos previamente entrenados

""")

# Cargar el archivo de datos

uploaded\_file = st.file\_uploader("📤 Sube tu archivo de datos", type=["csv", "txt", "xlsx"])

if uploaded\_file is not None:

    # Crear columnas para organizar el layout

    col1, col2 = st.columns([2, 1])

    with col1:

        # Selección del delimitador si es un archivo CSV o TXT

        delimiter = st.selectbox("Selecciona el delimitador", [",", ";", "\t", " "], index=0)

        # Cargar el archivo según su formato

        if uploaded\_file.name.endswith(".csv") or uploaded\_file.name.endswith(".txt"):

            data\_mult = pd.read\_csv(uploaded\_file, delimiter=delimiter, header=None)

        elif uploaded\_file.name.endswith(".xlsx"):

            data\_mult = pd.read\_excel(uploaded\_file, header=None)

        st.write("Vista previa de los datos:")

        st.dataframe(data\_mult.head(), use\_container\_width=True)

    with col2:

        # Añadir nombres de columnas si el usuario lo solicita

        if st.checkbox("Añadir nombres de columnas"):

            columns = st.text\_area("Especifica los nombres de las columnas separados por comas:",

                                   value="ID,Temperature,Humidity,UV,Voltage,Current,Illuminance,ClientIP,SensorID,DateTime")

            data\_mult.columns = [col.strip() for col in columns.split(",")]

            st.write("Nombres de columnas actualizados:")

            st.dataframe(data\_mult.head(), use\_container\_width=True)

    # Seleccionar las variables independientes y dependientes

    st.subheader("🎯 Selección de Variables")

    input\_vars = st.multiselect("Variables independientes (X)", data\_mult.columns.tolist(), default=data\_mult.columns.tolist()[:-1])

    target\_var = st.selectbox("Variable dependiente (y)", data\_mult.columns.tolist(), index=len(data\_mult.columns.tolist()) - 1)

    X = data\_mult[input\_vars]

    y = data\_mult[target\_var]

    # Opción de estandarizar los datos

    if st.checkbox("Estandarizar las variables independientes"):

        scaler = StandardScaler()

        X = scaler.fit\_transform(X)

        st.write("Datos estandarizados:")

        st.dataframe(pd.DataFrame(X, columns=input\_vars).head(), use\_container\_width=True)

    # Dividir los datos en entrenamiento y prueba

    test\_size = st.slider("Tamaño del conjunto de prueba (%)", 10, 50, 20)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=test\_size/100, random\_state=42)

    # Configurar la red neuronal

    st.subheader("🛠️ Configuración de la Red Neuronal")

    num\_layers = st.slider("Número de capas ocultas", 1, 10, 3)

    layers = []

    for i in range(num\_layers):

        col1, col2 = st.columns(2)

        with col1:

            neurons = st.number\_input(f"Neurones en la capa {i+1}", min\_value=1, value=64)

        with col2:

            activation = st.selectbox(f"Activación capa {i+1}", ["relu", "sigmoid", "tanh"], index=0)

        layers.append((neurons, activation))

    epochs = st.number\_input("Número de épocas", min\_value=10, value=100)

    if st.button("🚀 Entrenar el modelo"):

        with st.spinner("Entrenando el modelo..."):

            # Crear el modelo

            model = Sequential()

            model.add(Dense(layers[0][0], input\_dim=X\_train.shape[1], activation=layers[0][1]))

            for neurons, activation in layers[1:]:

                model.add(Dense(neurons, activation=activation))

            model.add(Dense(1))  # Capa de salida para regresión

            # Compilar el modelo

            model.compile(optimizer='adam', loss='mean\_squared\_error')

            # Entrenar el modelo

            history = model.fit(X\_train, y\_train, epochs=epochs, batch\_size=32, validation\_data=(X\_test, y\_test), verbose=0)

            # Hacer predicciones

            y\_pred = model.predict(X\_test)

            # Calcular R^2 y RMSE

            r2 = r2\_score(y\_test, y\_pred)

            rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

            col1, col2 = st.columns(2)

            with col1:

                st.metric("R² Score", f"{r2:.4f}")

            with col2:

                st.metric("RMSE", f"{rmse:.4f}")

            # Gráficos de la pérdida durante el entrenamiento y la validación

            fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

            ax1.plot(history.history['loss'], label='Entrenamiento')

            ax1.plot(history.history['val\_loss'], label='Validación')

            ax1.set\_title('Pérdida durante el entrenamiento')

            ax1.set\_xlabel('Época')

            ax1.set\_ylabel('Pérdida')

            ax1.legend()

            ax2.scatter(y\_test, y\_pred, alpha=0.5)

            ax2.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'r--', lw=2)

            ax2.set\_title('Predicciones vs Valores Reales')

            ax2.set\_xlabel('Valores Reales')

            ax2.set\_ylabel('Predicciones')

            st.pyplot(fig)

            st.success("Entrenamiento completado.")

        # Guardar el modelo entrenado

        model.save('trained\_model.h5')

        if 'scaler' in locals():

            joblib.dump(scaler, 'scaler.pkl')

        st.download\_button('📥 Descargar modelo', data=open('trained\_model.h5', 'rb'), file\_name='trained\_model.h5')

# Sección para cargar un modelo previamente entrenado

st.subheader("📁 Cargar Modelo Entrenado")

uploaded\_model = st.file\_uploader("Sube un modelo entrenado (.h5)", type=["h5"])

uploaded\_scaler = st.file\_uploader("Sube el scaler (opcional, .pkl)", type=["pkl"])

if uploaded\_model is not None:

    # Guardar el modelo cargado

    with open("loaded\_model.h5", "wb") as f:

        f.write(uploaded\_model.getbuffer())

    # Cargar el modelo

    model = load\_model("loaded\_model.h5")

    st.success("Modelo cargado correctamente.")

    # Cargar el scaler si se proporcionó

    if uploaded\_scaler is not None:

        with open("loaded\_scaler.pkl", "wb") as f:

            f.write(uploaded\_scaler.getbuffer())

        scaler = joblib.load("loaded\_scaler.pkl")

        st.success("Scaler cargado correctamente.")

    # Obtener la estructura del modelo

    input\_shape = model.input\_shape[1]

    # Probar el modelo con nuevos datos

    st.subheader("🔮 Predicción en Tiempo Real")

    st.write("Ingresa los valores para las variables de entrada:")

    # Crear un diccionario para almacenar los valores de entrada

    user\_input = {}

    # Crear dos columnas para organizar los inputs

    col1, col2 = st.columns(2)

    # Distribuir los inputs en las columnas

    for i in range(input\_shape):

        if i % 2 == 0:

            with col1:

                user\_input[f"Variable {i+1}"] = st.number\_input(f"Valor para Variable {i+1}", value=0.0, format="%.4f")

        else:

            with col2:

                user\_input[f"Variable {i+1}"] = st.number\_input(f"Valor para Variable {i+1}", value=0.0, format="%.4f")

    if st.button("Realizar Predicción"):

        # Convertir los inputs del usuario a un array numpy

        input\_data = np.array(list(user\_input.values())).reshape(1, -1)

        # Aplicar el scaler si está disponible

        if 'scaler' in locals():

            input\_data = scaler.transform(input\_data)

        # Realizar la predicción

        prediction = model.predict(input\_data)

        # Mostrar la predicción

        st.metric("Predicción del modelo", f"{prediction[0][0]:.4f}")

        # Visualizar la predicción en un gráfico

        fig, ax = plt.subplots()

        ax.bar(["Predicción"], prediction[0])

        ax.set\_ylabel("Valor Predicho")

        ax.set\_title("Resultado de la Predicción")

        st.pyplot(fig)

else:

    st.info("Por favor, carga un modelo entrenado para realizar predicciones.")

