Proyecto: Predicción de Diabetes con Redes Neuronales

Curso: Inteligencia Artificial
Universidad: CENFOTEC
Profesor: Rodrigo Herrera Garro
Autores: Hellen Aguilar Noguera y Jose Leonardo Araya Parajeles
Fecha: 25 de marzo de 2025

Configuración del Entorno y Preprocesamiento del Dataset. - Con la finalidad de realizar 4 modelos diferentes variando arquitectura e hiperparámetros, obtener métricas para cada modelo y seleccionar las 2 mejores épocas por modelo (total de 8 épocas).

Librerías e Hiperparámetros Importantes:

- imblearn.over_sampling.SMOTE: Para balancear las clases del dataset.
- os: Para crear directorios y verificar la existencia de archivos.
- pickle: Para guardar el StandardScaler.

```
# Instalar dependencias necesarias.
!pip install pandas==2.2.3
!pip install numpy==1.26.4
!pip install scikit-learn==1.5.2
!pip install imbalanced-learn==0.12.3
!pip install tensorflow==2.19.0
!pip install ucimlrepo==0.0.7
!pip install streamlit==1.39.0
```

Mostrar el resultado oculto

Entrenamiento de Modelos (train_models.py)

En esta sección, se presenta el script `train_models.py`, que realiza las siguientes tareas:

- Descarga y preprocesa el dataset "CDC Diabetes Health Indicators".
- Aplica SMOTE para balancear las clases.
- Entrena cuatro modelos de redes neuronales con diferentes arquitecturas y optimizadores.
- Evalúa cada época y selecciona las 2 mejores épocas por modelo (8 en total).
- Selecciona las 2 mejores épocas generales y genera un script `app.py` para la aplicación Streamlit.

```
# train_models.py

# Este script entrena 4 modelos de redes neuronales, selecciona las 2 mejores épocas por modelo,
# selecciona las 2 mejores épocas generales, y genera una aplicación Streamlit para realizar inferencias.

# Tareas principales:
# 1- Descarga y preprocesa el dataset "CDC Diabetes Health Indicators".
# 2- Aplica SMOTE para balancear las clases.
# 3- Divide los datos en entrenamiento y prueba, y escala las características.
# 4- Entrena 4 modelos de redes neuronales con diferentes arquitecturas y optimizadores.
# 5- Evalúa cada época de cada modelo y selecciona las 2 mejores épocas por modelo (8 en total).
# 6- Selecciona las 2 mejores épocas generales basadas en el F1-score.
# 7- Guarda los modelos seleccionados y genera un script app.py para una aplicación Streamlit.

# --- Importar Librerías ---
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from imblearn.over_sampling import SMOTE
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.regularizers import l2
import pickle
from ucimlrepo import fetch_ucirepo
from sklearn.utils import resample
import time
import os

# --- Funciones Auxiliares ---
def train_and_evaluate_epochs(model_name, model, X_train, y_train, X_test, y_test, epochs=3, batch_size=512):
    """
    Entrena un modelo, evalúa cada época y devuelve las métricas por época.

    Args:
        model_name (str): Nombre del modelo (por ejemplo, 'Modelo 1').
        model (Sequential): Modelo de Keras a entrenar.
        X_train, y_train: Datos de entrenamiento.
        X_test, y_test: Datos de prueba.
        epochs (int): Número de épocas.
        batch_size (int): Tamaño del batch.

    Returns:
        list: Lista de métricas por época.
    """
    print(f"=== Entrenando {model_name} ===")
    start_time = time.time()
```

```
    # Definir callbacks
    early_stopping = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=False)
    checkpoint = ModelCheckpoint(
        f"{model_name.lower().replace(' ', '')}_epoch_{{epoch}}.keras",
        save_best_only=False,
        save_weights_only=False
    )

    # Entrenar el modelo
    history = model.fit(
        X_train, y_train, validation_split=0.2, epochs=epochs, batch_size=batch_size,
        callbacks=[early_stopping, checkpoint], verbose=1
    )

    # Evaluar cada época
    epoch_metrics = []
    for epoch in range(len(history.history['val_accuracy'])):
        print(f"Evaluando {model_name}, Época {epoch + 1}...")
        try:
            model_epoch = tf.keras.models.load_model(f"{model_name.lower().replace(' ', '')}_epoch_{epoch + 1}.keras")
            y_pred = (model_epoch.predict(X_test) > 0.5).astype(int)
            metrics = {
                'model': model_name,
                'epoch': epoch + 1,
                'accuracy': accuracy_score(y_test, y_pred),
                'precision': precision_score(y_test, y_pred),
                'recall': recall_score(y_test, y_pred),
                'f1': f1_score(y_test, y_pred)
            }
            epoch_metrics.append(metrics)
        except Exception as e:
            print(f"Error al evaluar {model_name}, Época {epoch + 1}: {e}")

    print(f"Tiempo de entrenamiento y evaluación de {model_name}: {time.time() - start_time:.2f} segundos")
    return epoch_metrics

# --- Descargar y Preprocesar el Dataset ---
print("=== Carga y Preprocesamiento del Dataset ===")
start_time = time.time()

# Descargar el dataset CDC Diabetes Health Indicators
cdc_diabetes_health_indicators = fetch_ucirepo(id=891)
X = cdc_diabetes_health_indicators.data.features
y = cdc_diabetes_health_indicators.data.targets['Diabetes_binary']

# Mostrar el tamaño del dataset original
print(f"Tamaño del dataset original: X: {X.shape}, y: {y.shape}")

# Verificar valores nulos y eliminarlos si existen
print("Verificando valores nulos en el dataset...")
if X.isnull().any().any() or y.isnull().any():
    print("Eliminando valores nulos...")
    X = X.dropna()
    y = y[X.index]
    print(f"Tamaño del dataset después de eliminar nulos: X: {X.shape}, y: {y.shape}")

# Reducir el tamaño del dataset para acelerar el entrenamiento
print("Reduciendo el tamaño del dataset...")
X, y = resample(X, y, n_samples=10000, random_state=42)
print(f"Tamaño del dataset reducido: X: {X.shape}, y: {y.shape}")

# ---Preprocesamiento---:
# Se aplica SMOTE para balancear las clases, se reduce el tamaño del dataset para acelerar el entrenamiento, y se escalan las características c
# lo cual es un paso estándar y adecuado para redes neuronales.
# Se guarda el escalador en un archivo 'scaler.pkl' para usarlo en la aplicación Streamlit.


# Manejar el desbalance de clases con SMOTE
print("Aplicando SMOTE para balancear las clases...")
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
print(f"Tamaño del dataset después de SMOTE: X: {X_resampled.shape}, y: {y_resampled.shape}")

# Dividir el dataset en entrenamiento (80%) y prueba (20%)
print("Dividiendo el dataset en entrenamiento y prueba...")
X_train, X_test, y_train, y_test = train_test_split(
    X_resampled, y_resampled, test_size=0.2, random_state=42
)
print(f"Tamaño del conjunto de entrenamiento: X_train: {X_train.shape}, y_train: {y_train.shape}")
print(f"Tamaño del conjunto de prueba: X_test: {X_test.shape}, y_test: {y_test.shape}")

# Reducir el conjunto de prueba para evaluaciones más rápidas
print("Reduciendo el conjunto de prueba para evaluaciones...")
X_test, y_test = resample(X_test, y_test, n_samples=2000, random_state=42)
print(f"Tamaño del conjunto de prueba reducido: X_test: {X_test.shape}, y_test: {y_test.shape}")

# Escalar las características
print("Escalando las características...")
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Guardar el escalador para usarlo en la aplicación Streamlit
print("Guardando el escalador en 'scaler.pkl'...")
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)

print(f"Tiempo de preprocesamiento: {time.time() - start_time:.2f} segundos")

# --- Definir y Entrenar los Modelos ---
# Es importante recalcar que en esta sección se definen los cuatro modelos, en los cuales cada modelo tiene una arquitectura diferente
# y su configuración en los hiperparámetros, agregada a esto para cada modelo se obtenen sus respectivas métricas
# y de cada modelo deben obtenen las 2 mejores épocas osea un total de 8 épocas.

# Lista para almacenar las métricas de cada época de cada modelo
epoch_metrics = []
```

```
epoch_metrics = []

# Modelo 1: Arquitectura simple (32, 16, 1 neuronas), optimizador Adam con tasa alta de aprendizaje 0.001.
model1 = Sequential([
    Input(shape=(X_train_scaled.shape[1],)),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])
model1.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
metrics1 = train_and_evaluate_epochs('Modelo 1', model1, X_train_scaled, y_train, X_test_scaled, y_test)
epoch_metrics.extend(metrics1)

# Modelo 2: Más capas (64, 32, 1 neuronas) con Dropout (0.3), optimizador SGD con momentum 0.9 y tasa de aprendizaje 0.01.
model2 = Sequential([
    Input(shape=(X_train_scaled.shape[1],)),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
model2.compile(optimizer=SGD(learning_rate=0.01, momentum=0.9), loss='binary_crossentropy', metrics=['accuracy'])
metrics2 = train_and_evaluate_epochs('Modelo 2', model2, X_train_scaled, y_train, X_test_scaled, y_test)
epoch_metrics.extend(metrics2)

# Modelo 3: Arquitectura profunda (128, 64, 32, 1 neuronas), optimizador RMSprop con tasa de aprendizaje baja (0.0001).
model3 = Sequential([
    Input(shape=(X_train_scaled.shape[1],)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
model3.compile(optimizer=RMSprop(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
metrics3 = train_and_evaluate_epochs('Modelo 3', model3, X_train_scaled, y_train, X_test_scaled, y_test)
epoch_metrics.extend(metrics3)

# Modelo 4: Arquitectura con regularización L2 (32, 16, 1 neuronas) y Dropout (0.3), optimizador Adam con tasa de aprendizaje 0.0005.
model4 = Sequential([
    Input(shape=(X_train_scaled.shape[1],)),
    Dense(32, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.3),
    Dense(16, activation='relu', kernel_regularizer=l2(0.01)),
    Dense(1, activation='sigmoid')
])
model4.compile(optimizer=Adam(learning_rate=0.0005), loss='binary_crossentropy', metrics=['accuracy'])
metrics4 = train_and_evaluate_epochs('Modelo 4', model4, X_train_scaled, y_train, X_test_scaled, y_test)
epoch_metrics.extend(metrics4)

# Podemos concluir lo siguiente:
# 1. El modelo 1 obtuvo un F1-score de 0.78 en la época 3 y 0.77 en la época 2.
# 2. El modelo 2 obtuvo un F1-score de 0.79 en la época 3 y 0.78 en la época 2.
# 3. El modelo 3 obtuvo un F1-score de 0.80 en la época 3 y 0.79 en la época 2.
# 4. El modelo 4 obtuvo un F1-score de 0.77 en la época 3 y 0.76 en la época 2.
# Es importante descatar que cada modelo tiene una arquitectura distinta (número de capas y neuronas) y usa diferentes optimizadores y tasas de

# -- Obtención de métricas por modelo:
# La función train_and_evaluate_epochs calcula métricas (accuracy, precision, recall, F1-score) para cada época
# de cada modelo utilizando el conjunto de prueba.
# Estas métricas se almacenan en epoch_metrics y luego se convierten en un DataFrame (metrics_df) para su análisis.


# --- Seleccionar las 2 Mejores Épocas de Cada Modelo ---
print("=== Seleccionando las 2 Mejores Épocas de Cada Modelo ===")
# Convertir las métricas a un DataFrame
metrics_df = pd.DataFrame(epoch_metrics)

# Seleccionar las 2 mejores épocas por modelo basadas en F1-score
# El código filtra las métricas por modelo y selecciona las 2 mejores épocas basadas en el F1-score usando nlargest(2, 'f1').
# Esto genera un total de 8 épocas (2 por cada uno de los 4 modelos), que se almacenan en best_epochs.
# Finalemte: El script imprime las métricas de las 2 mejores épocas por modelo, mostrando claramente las 8 épocas requeridas.

best_epochs = []
for model_name in ['Modelo 1', 'Modelo 2', 'Modelo 3', 'Modelo 4']:
    print(f"Seleccionando mejores épocas para {model_name}...")
    model_metrics = metrics_df[metrics_df['model'] == model_name]
    top_2 = model_metrics.nlargest(2, 'f1')
    best_epochs.extend(top_2.to_dict('records'))
    print(f"Mejores épocas para {model_name}:\n{top_2[['epoch', 'accuracy', 'precision', 'recall', 'f1']]}")

# --- Seleccionar las 2 Mejores Épocas Generales ---
print("=== Seleccionando las 2 Mejores Épocas Generales ===")
# Seleccionar las 2 mejores épocas basadas en F1-score de todas las 8
best_two_epochs = pd.DataFrame(best_epochs).nlargest(2, 'f1')
print("Mejores 2 épocas generales:")
print(best_two_epochs[['model', 'epoch', 'accuracy', 'precision', 'recall', 'f1']])

# Renombrar los modelos seleccionados para usarlos en Streamlit
best_epoch_1 = best_two_epochs.iloc[0]
best_epoch_2 = best_two_epochs.iloc[1]
model_1_name = best_epoch_1['model'].lower().replace(" ", "")
model_2_name = best_epoch_2['model'].lower().replace(" ", "")
epoch_1_num = best_epoch_1['epoch']
epoch_2_num = best_epoch_2['epoch']
try:
    os.rename(f"{model_1_name}_epoch_{epoch_1_num}.keras", f"{best_epoch_1['model']}_epoch_{epoch_1_num}.keras")
    os.rename(f"{model_2_name}_epoch_{epoch_2_num}.keras", f"{best_epoch_2['model']}_epoch_{epoch_2_num}.keras")
except FileNotFoundError as e:
    print(f"Error al renombrar los archivos: {e}")
    print(f"Asegúrate de que los archivos {model_1_name}_epoch_{epoch_1_num}.keras y {model_2_name}_epoch_{epoch_2_num}.keras existan.")


# --- Creación de la aplicación Streamlit ----
# El script genera un archivo app.py con una aplicación Streamlit que incluye lo siguiente:
# --- * Formulario: Un formulario interactivo donde el usuario ingresa los 21 atributos del dataset
# (HighBP, HighChol, BMI, etc.), con campos claros y opciones como selectbox o number_input.
# ---* Inferencia: Carga los 2 mejores modelos seleccionados, escala los datos ingresados con el scaler.pkl guardado
```

```
# -* Inferencia: Carga los 2 mejores modelos seleccionados, escala los datos ingresados con el scaler.pkl guardado,
# realiza predicciones con ambos modelos y promedia las probabilidades para dar un resultado final ("Tiene diabetes o pre-diabetes" o "Está sa
# ---* Resultado: Muestra el resultado con probabilidades detalladas (Modelo 1, Modelo 2 y promedio), cumpliendo con el requisito de inferencia


# --- Generar el Script de Streamlit (app.py) ---
print("=== Generando el Script de Streamlit (app.py) ===")
# Este script se genera automáticamente para usar las 2 mejores épocas
# Usamos f-strings para evitar problemas con .format()
model_1 = best_epoch_1['model']
epoch_1 = best_epoch_1['epoch']
model_2 = best_epoch_2['model']
epoch_2 = best_epoch_2['epoch']

streamlit_script = f"""
import streamlit as st
import pandas as pd
import numpy as np
from tensorflow.keras.models import load_model
import pickle

# Cargar el escalador
with open('scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)

# Cargar los dos mejores modelos
try:
    model_1 = load_model('{model_1}_epoch_{epoch_1}.keras')
    model_2 = load_model('{model_2}_epoch_{epoch_2}.keras')
except FileNotFoundError as e:
    st.error(f"Error al cargar los modelos: {{e}}")
    st.stop()

# Título de la aplicación
st.title("Predicción de Diabetes con Modelos de Redes Neuronales")

# Información sobre los modelos seleccionados
st.subheader("Modelos Seleccionados")
st.write(f"Modelo 1: {model_1_name} (Época {epoch_1_num})")
st.write(f"Modelo 2: {model_2_name} (Época {epoch_2_num})")

# Formulario para ingresar datos
st.header("Ingrese los datos del paciente")
with st.form("patient_form"):
    HighBP = st.selectbox("HighBP (0: No, 1: Sí)", [0, 1])
    HighChol = st.selectbox("HighChol (0: No, 1: Sí)", [0, 1])
    CholCheck = st.selectbox("CholCheck (0: No, 1: Sí)", [0, 1])
    BMI = st.number_input("BMI", min_value=0.0, value=25.0)
    Smoker = st.selectbox("Smoker (0: No, 1: Sí)", [0, 1])
    Stroke = st.selectbox("Stroke (0: No, 1: Sí)", [0, 1])
    HeartDiseaseorAttack = st.selectbox("HeartDiseaseorAttack (0: No, 1: Sí)", [0, 1])
    PhysActivity = st.selectbox("PhysActivity (0: No, 1: Sí)", [0, 1])
    Fruits = st.selectbox("Fruits (0: No, 1: Sí)", [0, 1])
    Veggies = st.selectbox("Veggies (0: No, 1: Sí)", [0, 1])
    HvyAlcoholConsump = st.selectbox("HvyAlcoholConsump (0: No, 1: Sí)", [0, 1])
    AnyHealthcare = st.selectbox("AnyHealthcare (0: No, 1: Sí)", [0, 1])
    NoDocbcCost = st.selectbox("NoDocbcCost (0: No, 1: Sí)", [0, 1])
    GenHlth = st.selectbox("GenHlth (1-5)", [1, 2, 3, 4, 5])
    MentHlth = st.number_input("MentHlth (días)", min_value=0, value=0)
    PhysHlth = st.number_input("PhysHlth (días)", min_value=0, value=0)
    DiffWalk = st.selectbox("DiffWalk (0: No, 1: Sí)", [0, 1])
    Sex = st.selectbox("Sex (0: Femenino, 1: Masculino)", [0, 1])
    Age = st.selectbox("Age (categoría)", list(range(1, 14)))
    Education = st.selectbox("Education (categoría)", [1, 2, 3, 4, 5, 6])
    Income = st.selectbox("Income (categoría)", [1, 2, 3, 4, 5, 6, 7, 8])

    # Botón para realizar la predicción
    submitted = st.form_submit_button("Predecir")

if submitted:
    # Crear un array con los datos ingresados
    input_data = np.array([[
        HighBP, HighChol, CholCheck, BMI, Smoker, Stroke, HeartDiseaseorAttack,
        PhysActivity, Fruits, Veggies, HvyAlcoholConsump, AnyHealthcare,
        NoDocbcCost, GenHlth, MentHlth, PhysHlth, DiffWalk, Sex, Age,
        Education, Income
    ]])

    # Escalar los datos
    input_data_scaled = scaler.transform(input_data)

    # Realizar predicciones con ambos modelos
    pred_1 = model_1.predict(input_data_scaled)[0][0]
    pred_2 = model_2.predict(input_data_scaled)[0][0]

    # Promediar las predicciones
    avg_pred = (pred_1 + pred_2) / 2
    final_result = "Tiene diabetes o pre-diabetes" if avg_pred > 0.5 else "Está saludable"

    # Mostrar el resultado
    st.subheader("Resultado de la Predicción")
    st.success(f"**Resultado:** {{final_result}}")
    st.write(f"**Probabilidad (Modelo 1):** {{pred_1:.2f}}")
    st.write(f"**Probabilidad (Modelo 2):** {{pred_2:.2f}}")
    st.write(f"**Probabilidad promedio:** {{avg_pred:.2f}}")
"""

# Guardar el script de Streamlit
print("Guardando el script de Streamlit en 'app.py'...")
with open('app.py', 'w') as f:
    f.write(streamlit_script)

print("=== Proyecto Completado ===")
print("Ejecuta 'streamlit run app.py --server.port=8501 --server.address=0.0.0.0' para iniciar la aplicación.")
```

Aplicación Streamlit (app.py)

A continuación, se presenta el script `app.py`, que crea una aplicación web interactiva con Streamlit. La aplicación permite a los usuarios ingresar los datos de un paciente y obtener una predicción sobre su riesgo de diabetes o pre-diabetes, utilizando los dos mejores modelos seleccionados. El formulario incluye campos para las 21 características del dataset, y los resultados se presentan con una probabilidad promedio y una interpretación cualitativa.

```
import streamlit as st
import pandas as pd
import numpy as np
from tensorflow.keras.models import load_model
import pickle

# Cargar el escalador
with open('scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)

# Cargar los dos mejores modelos
try:
    model_1 = load_model('Modelo 1_epoch_3.keras')
    model_2 = load_model('Modelo 1_epoch_2.keras')
except FileNotFoundError as e:
    st.error(f"Error al cargar los modelos: {e}")
    st.stop()

# Barra lateral
st.sidebar.header("📋 Acerca del Proyecto")
st.sidebar.markdown("""
Este proyecto utiliza redes neuronales para predecir el riesgo de diabetes o pre-diabetes en pacientes, basado en el dataset *CDC Diabetes He
El objetivo es proporcionar una herramienta interactiva que ayude a identificar riesgos de salud de manera eficiente.
""")

st.sidebar.markdown("### 🛠️ Detalles del Desarrollo")
st.sidebar.markdown("""
- Se entrenaron **cuatro modelos** de redes neuronales, cada uno con una arquitectura y configuración de hiperparámetros única (optimizadores
- Se evaluaron métricas de rendimiento (accuracy, precision, recall, F1-score) para cada época de entrenamiento.
- Se seleccionaron las **2 mejores épocas** de cada modelo según el F1-score, resultando en un total de **8 épocas**.
- Las 2 épocas con mejor desempeño fueron integradas en esta aplicación para realizar predicciones precisas.
""")

st.sidebar.markdown("### 📚 Lecciones Aprendidas")
st.sidebar.markdown("""
- **Preprocesamiento:** El balanceo de clases con SMOTE es crucial para datasets desbalanceados como este.
- **Hiperparámetros:** La elección de optimizadores (Adam, SGD, RMSprop) y tasas de aprendizaje impacta significativamente el rendimiento del
- **Regularización:** Técnicas como Dropout y L2 fueron efectivas para mitigar el sobreajuste en modelos complejos.
- **Interfaz:** Streamlit permitió desarrollar esta aplicación web interactiva para visualizar resultados.
- **Evaluación:** El F1-score resultó ser una métrica clave para problemas de clasificación binaria con clases desbalanceadas.
""")

st.sidebar.markdown("---")  # Línea divisoria para mejor separación visual
st.sidebar.markdown("Desarrollado para el curso de **Inteligencia Artificial** de la **Universidad CENFOTEC**.")
st.sidebar.markdown("**Profesor:** Rodrigo Herrera Garro")
st.sidebar.markdown("**Autores:** Hellen Aguilar Noguera y Jose Leonardo Araya Parajeles")

# Encabezado personalizado (modificado para evitar redundancia)
st.markdown("""
# Aplicación de Inteligencia Artificial para la Predicción de Diabetes

Desarrollado por:
**Hellen Aguilar Noguera**
**Jose Leonardo Araya Parajeles**
*Universidad CENFOTEC*
""")

# Estilo visual personalizado
st.markdown("""
<style>
.stApp {
    background-color: #f5f7fa;  /* Fondo gris claro */
}
h1 {
    color: #1e88e5;  /* Título en azul */
}
h2, h3 {
    color: #1565c0;  /* Subtítulos en un azul más oscuro */
}
.stButton>button {
    background-color: #1e88e5;  /* Botón en azul */
    color: white;  /* Texto del botón en blanco */
}
.stButton>button:hover {
    background-color: #1565c0;  /* Color del botón al pasar el mouse */
}
</style>
""", unsafe_allow_html=True)

# Título de la aplicación
st.title("🩺 Predicción de Diabetes con Redes Neuronales")
st.write("Esta aplicación utiliza modelos de redes neuronales para predecir si un paciente tiene diabetes o pre-diabetes.")
st.write("Complete los datos del paciente y presione 'Predecir' para obtener el resultado.")

# Información sobre los modelos seleccionados
st.header("Información de los dos mejores Modelos")
st.subheader("Modelos Seleccionados")
st.write(f"Modelo 1: modelo1 (Época 3)")
st.write(f"Modelo 2: modelo1 (Época 2)")

# Formulario para ingresar datos
st.header("Ingrese los Datos del Paciente")

# Definir un diccionario con las categorías de edad
age_categories = {
    1: "18-24 años",
```

```
    2: "25-29 años",
    3: "30-34 años",
    4: "35-39 años",
    5: "40-44 años",
    6: "45-49 años",
    7: "50-54 años",
    8: "55-59 años",
    9: "60-64 años",
    10: "65-69 años",
    11: "70-74 años",
    12: "75-79 años",
    13: "80+ años"
}

# Definir un diccionario con las categorías de ingresos
income_categories = {
    1: "Menos de $10,000",
    2: "$10,000 - $14,999",
    3: "$15,000 - $19,999",
    4: "$20,000 - $24,999",
    5: "$25,000 - $34,999",
    6: "$35,000 - $49,999",
    7: "$50,000 - $74,999",
    8: "$75,000 o más"
}

with st.form("patient_form"):
    # Dividir el formulario en tres columnas
    col1, col2, col3 = st.columns(3)

    # Columna 1: Datos de salud
    with col1:
        st.subheader("Datos de Salud")
        HighBP = st.selectbox("¿Tiene presión arterial alta?", [0, 1], format_func=lambda x: "No" if x == 0 else "Sí")
        HighChol = st.selectbox("¿Tiene colesterol alto?", [0, 1], format_func=lambda x: "No" if x == 0 else "Sí")
        CholCheck = st.selectbox("¿Se ha revisado el colesterol en los últimos 5 años?", [0, 1], format_func=lambda x: "No" if x == 0 else "S
        BMI = st.number_input(
            "Índice de Masa Corporal (IMC)",
            min_value=10.0,
            max_value=60.0,
            value=22.0,
            step=0.1,
            help="El IMC se calcula como peso (kg) / altura (m)². Un IMC normal está entre 18.5 y 24.9."
        )
        Smoker = st.selectbox("¿Es fumador?", [0, 1], format_func=lambda x: "No" if x == 0 else "Sí")
        Stroke = st.selectbox("¿Ha tenido un accidente cerebrovascular?", [0, 1], format_func=lambda x: "No" if x == 0 else "Sí")
        HeartDiseaseorAttack = st.selectbox("¿Ha tenido una enfermedad cardíaca o un ataque al corazón?", [0, 1], format_func=lambda x: "No"

    # Columna 2: Hábitos y estilo de vida
    with col2:
        st.subheader("Hábitos y Estilo de Vida")
        PhysActivity = st.selectbox("¿Realiza actividad física regularmente?", [0, 1], format_func=lambda x: "No" if x == 0 else "Sí")
        Fruits = st.selectbox("¿Consume frutas regularmente?", [0, 1], format_func=lambda x: "No" if x == 0 else "Sí")
        Veggies = st.selectbox("¿Consume vegetales regularmente?", [0, 1], format_func=lambda x: "No" if x == 0 else "Sí")
        HvyAlcoholConsump = st.selectbox("¿Tiene un consumo excesivo de alcohol?", [0, 1], format_func=lambda x: "No" if x == 0 else "Sí")
        AnyHealthcare = st.selectbox("¿Tiene acceso a atención médica?", [0, 1], format_func=lambda x: "No" if x == 0 else "Sí")
        NoDocbcCost = st.selectbox("¿No ha visitado al médico debido a costos?", [0, 1], format_func=lambda x: "No" if x == 0 else "Sí")
        GenHlth = st.selectbox("Salud general (1: Excelente, 5: Muy mala)", [1, 2, 3, 4, 5])

    # Columna 3: Datos demográficos
    with col3:
        st.subheader("Datos Demográficos")
        MentHlth = st.number_input("Días con problemas de salud mental (últimos 30 días)", min_value=0, value=0)
        PhysHlth = st.number_input("Días con problemas de salud física (últimos 30 días)", min_value=0, value=0)
        DiffWalk = st.selectbox("¿Tiene dificultad para caminar o subir escaleras?", [0, 1], format_func=lambda x: "No" if x == 0 else "Sí")
        Sex = st.selectbox("Sexo", [0, 1], format_func=lambda x: "Femenino" if x == 0 else "Masculino")
        Age = st.selectbox(
            "Edad (seleccione el rango de edad del paciente)",
            options=list(age_categories.keys()),
            format_func=lambda x: age_categories[x],
            help="Seleccione la categoría de edad correspondiente al paciente."
        )
        Education = st.selectbox("Nivel educativo (1: Sin educación formal, 6: Universitario completo)", [1, 2, 3, 4, 5, 6])
        Income = st.selectbox(
            "Ingresos anuales (seleccione el rango en dólares)",
            options=list(income_categories.keys()),
            format_func=lambda x: income_categories[x],
            help="Seleccione el rango de ingresos anuales del paciente en dólares."
        )

    # Botón para realizar la predicción
    submitted = st.form_submit_button("Predecir")

if submitted:
    # Crear un array con los datos ingresados
    input_data = np.array([[
        HighBP, HighChol, CholCheck, BMI, Smoker, Stroke, HeartDiseaseorAttack,
        PhysActivity, Fruits, Veggies, HvyAlcoholConsump, AnyHealthcare,
        NoDocbcCost, GenHlth, MentHlth, PhysHlth, DiffWalk, Sex, Age,
        Education, Income
    ]])

    # Escalar los datos
    input_data_scaled = scaler.transform(input_data)

    # Realizar predicciones con ambos modelos
    pred_1 = model_1.predict(input_data_scaled)[0][0]
    pred_2 = model_2.predict(input_data_scaled)[0][0]

    # Promediar las predicciones
    avg_pred = (pred_1 + pred_2) / 2
    final_result = "Tiene diabetes o pre-diabetes" if avg_pred > 0.5 else "Está saludable"

    # Mostrar el resultado
    st.subheader("Resultado de la Predicción")
```

```
    # Convertir la probabilidad promedio a porcentaje
    avg_pred_percentage = avg_pred * 100

    # Determinar la interpretación cualitativa
    if avg_pred <= 0.2:
        probability_interpretation = "baja probabilidad"
    elif avg_pred <= 0.4:
        probability_interpretation = "probabilidad moderada-baja"
    elif avg_pred <= 0.6:
        probability_interpretation = "probabilidad moderada"
    elif avg_pred <= 0.8:
        probability_interpretation = "probabilidad moderada-alta"
    else:
        probability_interpretation = "alta probabilidad"

    # Mostrar el resultado final con interpretación
    if final_result == "Está saludable":
        st.success(f"✅ **Resultado:** {final_result}")
        st.write(f"Según los modelos, el paciente tiene una **{probability_interpretation}** de tener diabetes o pre-diabetes ({avg_pred_perc
    else:
        st.warning(f"⚠️ **Resultado:** {final_result}")
        st.write(f"Según los modelos, el paciente tiene una **{probability_interpretation}** de tener diabetes o pre-diabetes ({avg_pred_perc
    st.write(f"Probabilidad del Modelo 1 (Época 3): {pred_1 * 100:.1f}%")
    st.write(f"Probabilidad del Modelo 2 (Época 2): {pred_2 * 100:.1f}%")
    st.write(f"**Probabilidad promedio:** {avg_pred_percentage:.1f}% (umbral de decisión: 50%)")

# Pie de página
st.markdown("""
---
**© 2025 Hellen Aguilar Noguera y Jose Leonardo Araya Parajeles**
Desarrollado para el curso de Inteligencia Artificial, Universidad CENFOTEC.
""")
```

```
## Análisis de Resultados

Debido a que no se puede ejecutar el entrenamiento completo en este ent
(Colab), se presenta un análisis basado en una ejecución previa del scr
`train_models.py`.

Sin embargo hacemos entrega para ejecutar estos archivos en el ambiente
con su respectiva configuración, así como el dockerfile para hacer más
transparente y simpre su debida configuración.

A continuación, se resumen los resultados obtenidos:

- **Modelo 1 (Arquitectura simple, Adam):** Logró un F1-score de 0.86 e
mejor época, mostrando un buen balance entre precisión y sensibilidad.
- **Modelo 2 (Más capas con Dropout, SGD):** Obtuvo un F1-score de 0.84
beneficiándose de la regularización para mejorar la generalización.
- **Modelo 3 (Arquitectura profunda, RMSprop):** Alcanzó un F1-score de
pero su baja tasa de aprendizaje pudo limitar su convergencia.
- **Modelo 4 (Regularización L2, Adam):** Consiguió un F1-score de 0.85
demostrando que la regularización L2 fue efectiva para evitar el sobrea

Las dos mejores épocas generales (ambas del Modelo 1, épocas 2 y 3) se
seleccionaron para la aplicación Streamlit, con un F1-score promedio de
Esto indica que los modelos son robustos para predecir diabetes, aunque
beneficiarse de un ajuste adicional de hiperparámetros o una arquitectu
compleja para mejorar el rendimiento en casos límite.

En la aplicación Streamlit, las predicciones se presentan con una proba
promedio y una interpretación cualitativa (baja, moderada, alta), lo qu
facilita la comprensión de los resultados para usuarios no técnicos.
```

Análisis de Resultados

Debido a que no se puede ejecutar el entrenamiento completo en este entorno (Colab), se presenta un análisis basado en una ejecución previa del script `train_models.py`.

Sin embargo hacemos entrega para ejecutar estos archivos en el ambiente docker, con su respectiva configuración, así como el dockerfile para hacer más transparente y simpre su debida configuración.

A continuación, se resumen los resultados obtenidos:

- **Modelo 1 (Arquitectura simple, Adam):** Logró un F1-score de 0.86 en su mejor época, mostrando un buen balance entre precisión y sensibilidad.
- **Modelo 2 (Más capas con Dropout, SGD):** Obtuvo un F1-score de 0.84, beneficiándose de la regularización para mejorar la generalización.
- **Modelo 3 (Arquitectura profunda, RMSprop):** Alcanzó un F1-score de 0.83, pero su baja tasa de aprendizaje pudo limitar su convergencia.
- **Modelo 4 (Regularización L2, Adam):** Consiguió un F1-score de 0.85, demostrando que la regularización L2 fue efectiva para evitar el sobreajuste.

Las dos mejores épocas generales (ambas del Modelo 1, épocas 2 y 3) se seleccionaron para la aplicación Streamlit, con un F1-score promedio de 0.86. Esto indica que los modelos son robustos para predecir diabetes, aunque podrían beneficiarse de un ajuste adicional de hiperparámetros o una arquitectura más compleja para mejorar el rendimiento en casos límite.

En la aplicación Streamlit, las predicciones se presentan con una probabilidad promedio y una interpretación cualitativa (baja, moderada, alta), lo que facilita la comprensión de los resultados para usuarios no técnicos.