IA_Ecommerce_NLP_Redes_Neuronales_Prediccion.ipynb

Hipótesis 1: predecir cuándo será venta y cuándo no lo será analizando un conjunto de datos de la tienda online de Amazon.

Librerías

```
#librerías
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import json

from sklearn.preprocessing import LabelEncoder

#modelo
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

#red neuronal
import tensorflow as tf
from tensorflow.keras import layers, models, Sequential
from tensorflow import keras
from collections import Counter
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.metrics import Precision, Recall
from tensorflow.keras.layers import Dense , Embedding , Bidirectional , LSTM
from tensorflow.keras.preprocessing import sequence

#Bert model
from transformers import BertTokenizer, BertForSequenceClassification
from transformers import pipeline
import torch
from sklearn.model_selection import train_test_split
from tqdm.notebook import tqdm
from torch.optim import AdamW
from torch.cuda.amp import autocast, GradScaler

#metricas
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score, f1_score, confusion_matrix
```

Regresión logística

Hipótesis:

Para predecir la relación entre precio, descuento y cantidad de ventas para ello agrego una variable que determine cuándo es venta y cuándo no lo es:

rating_count < 100 == 0 No venta

rating_count > 100 == 1 Venta

rating_count es la cantidad de resenias que tiene un producto, es decir que representa su popularidad, la cantidad de veces que fue comprado y que los usuarios dieron su opinión por tanto es lo más cercano que podemos obtener a un venta - no venta

```
#cargamos el datase
sellers = pd.read_csv("Amazon.csv")
sellers
```

```
sellers.describe().T
```

```
sellers.info()
```

Limpieza del dataset Amazon.csv

```
sellers['discounted_price'] = sellers['discounted_price'].str.replace("₹", '')
sellers['actual_price'] = sellers['actual_price'].str.replace("₹", '')
sellers['discount_percentage'] = sellers['discount_percentage'].str.replace("%", '')
sellers['actual_price'] = sellers['actual_price'].str.replace(",", '')
sellers['rating_count'] = sellers['rating_count'].str.replace(",", '')
sellers
```

```
#valores nulos
count_nan = sellers.isnull().sum()
```

```
count_nan
```

```
#buscamos los valores nulos
rows_with_null = sellers[sellers.isnull().any(axis=1)]
rows_with_null
sellers.iloc[1463]
```

```
sellers.iloc[1366]
```

```
#buscamos una referencia donde ratin sea de 5 puntos
rating_5 = sellers[sellers['rating']== '5']
rating_5['review_title']
```

```
#se le asigna a los valores NaN el valor: 1 en la columna rating count  -
sellers.fillna(1, inplace=True)
sellers
```

```
#controlamos los valores NaN
count_nan = sellers.isnull().sum()
count_nan
```

Hipótesis: El usuario toma la decisión de comprar por el precio, el descuento, la categoría, la cantidad de resenias de un producto o el puntaje de las resenias?

Para resolver la hipótesis genero un nuevo dataset con estas columnas

```
#elijo las columnas que necesito para evaluar la hipotesis
nuevo_seller = sellers[['category', 'discounted_price', 'actual_price', 'discount_percentage', 'rating', 'rating_count']]
nuevo_seller
```

```
#al parecer hay un caracter pipe en la columna rating!
nuevo_seller[nuevo_seller['rating']=='|']
```

```
nuevo_seller = nuevo_seller[nuevo_seller['rating'] != '|']
nuevo_seller
```

```
nuevo_seller['category'].unique
nuevo_seller['category'].dtype
nuevo_seller
#quitamos las subcategorías y nos quedamos solo con las categorías generales
```

```
nuevo_seller.describe().T
nuevo_seller.info()
nuevo_seller.dtypes
```

```
nuevo_seller['rating'] = nuevo_seller['rating'].astype(float)
nuevo_seller['rating']
```

```
nuevo_seller['rating_count'] = pd.to_numeric(nuevo_seller['rating_count'], errors='coerce')
non_integer_values = nuevo_seller[~nuevo_seller['rating_count'].notna()]['rating_count']

non_integer_values
```

```
#como rating es float no lo modificara
le = LabelEncoder()
for column_name in nuevo_seller.columns:
    if nuevo_seller[column_name].dtype == object:
        nuevo_seller[column_name] = le.fit_transform(nuevo_seller[column_name])
    else:
        pass
```

```
nuevo_seller.dtypes
nuevo_seller.info()
```

```
sns.heatmap(nuevo_seller.corr(), annot=True)
plt.title("Matriz de correlacion")
plt.show()
```

En la matriz de correlación observamos que no hay una relación lineal, ni tampoco inversamente lineal entre ninguna de las variables. Pero no significa que no haya relación.

Regresión Logística

```
reg_log = LogisticRegression()
```

```
#Agregamos la nueva columna que será nuestro target "venta"
nuevo_seller['venta'] = (nuevo_seller['rating_count'] > 100).astype(int)
nuevo_seller
```

```
nuevo_seller.to_csv('nuevo_seller.csv', index=False)
```

```
X = nuevo_seller.drop('venta', axis=1).values
y = nuevo_seller['venta'].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
reg_log = LogisticRegression()
reg_log.fit(X_train, y_train)
```

```
y_pred = reg_log.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

```
confusion = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=['Venta', 'No Venta'], yticklabels=['Venta', 'No Venta'])
plt.title('Confusion Matrix-10')
plt.xlabel('Predicted Labels')
plt.ylabel('Test Labels')
plt.show()
```

Puede predecir cuándo no será venta

```
mae = mean_absolute_error(y_test, y_pred)

# Calcular el Error Cuadrático Medio (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calcular el Coeficiente de Determinación (R^2)
r2 = r2_score(y_test, y_pred)
```

```
mae
```

```
mse
```

```
r2
```

Puede predecir con gran precisión cuándo no será venta

Red neuronal secuencial con KERAS
Probaremos el mismo dataset pero en una red neuronal con Keras

```
# utilizamos el mismo dataset
df_seller = pd.read_csv("nuevo_seller.csv")
df_seller
```

```
#dividimos los datos en conjustos de entrenamiento y testing
train_data, test_data, train_labels, test_labels = train_test_split(
    df_seller.drop('venta', axis=1),
    df_seller['venta'],
    test_size=0.2,
    random_state=42
)
```

```
# 2. ESCALADO (ESTO ES LO NUEVO)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
train_data_scaled = scaler.fit_transform(train_data)
test_data_scaled = scaler.transform(test_data)

# 3. crear el modelo
modelo = LogisticRegression()

# 4. entrenar CON DATOS ESCALADOS
modelo.fit(train_data_scaled, train_labels)

# 5. predecir CON DATOS ESCALADOS
predicciones = modelo.predict(test_data_scaled)



#armamos la red neuronal con 3 capas dense, 2 funciones de activacion Relu
#y una última capa con una función de activación sigmoide que será la que determine venta - no venta
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape = (train_data.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])



# compilamos el modelo con un optimizador adam y usamos binnary crossentropy para que pueda
#clasificar en venta - no venta
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',  # Ajusta la función de pérdida según tu problema
    metrics=['accuracy']
)

model.summary()



# Entrenamos al modelos con 10 epochs y un batch size de 32
history_imdb = model.fit(train_data, train_labels, epochs=10, batch_size=32, validation_split=0.2)

# Plot accuracy and loss
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(history_imdb.history['accuracy'], label='Train Accuracy')
plt.plot(history_imdb.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy (IMDB)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_imdb.history['loss'], label='Train Loss')
plt.plot(history_imdb.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss (IMDB)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()



# Evaluar el modelo en el conjunto de prueba
test_loss, test_acc = model.evaluate(test_data, test_labels)
print(f'Accuracy -10: {test_acc}')

# Hacer predicciones en el conjunto de prueba
predictions = model.predict(test_data)
predicted_labels = (predictions > 0.5).astype(int)

# Mostrar la matriz de confusión
conf_matrix = confusion_matrix(list(test_labels), list(predicted_labels))
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=['Venta', 'No Venta'], yticklabels=['Venta', 'No Venta'])
plt.title('Confusion Matrix-10')
plt.xlabel('Predicted Labels')
plt.ylabel('Test Labels')
plt.show()

# Mostrar el informe de clasificación
print("Classification Report-10:")
print(classification_report(test_labels, predicted_labels))
```

Predice con gran facilidad cuando no será venta. Las métricas de precisión demasiado altas lo que podría significar que al probar con nuevos datos no pueda predecir con la misma precisión.

Hipótesis 2: Clasificar los comentarios de una tienda virtual en positivos - negativos para conocer el humor y las emociones de los usuarios

BERT

```
#cargamos nuestro set de datos
data = []

with open('amazon_resenias.json', 'r') as archivo:
    for line in archivo:
        try:
            data.append(json.loads(line))
        except json.JSONDecodeError as e:
            print(f"Error decoding JSON: {e}")

df = pd.DataFrame(data)
df
```

```
df.columns
```

```
#limpiamos el dataset de las columnas que no necesitamos
df.drop(['reviewerID', 'asin', 'reviewerName', 'helpful','summary', 'unixReviewTime', 'reviewTime'],axis='columns',inplace=True)
```

```
#renombramos las columnas
df= df.rename(columns={"reviewText":"Review","overall": "Rating"})
```

```
#creamos una función que las resenias con un rating menor o igual a 3 se normalice a 0 como negativo
#caso contrario las resenias cuya puntuación sean mayores a 3 tendrán un valor de 1 como resenia positiva
#Esta columna será nuestro target para enseniarle al modelo
def normalizar(df) :
    if df['Rating'] <= 3.0 :
        puntuacion = 0 # negativo
    else :
        puntuacion = 1 # positivo

    return puntuacion
```

```
#aplicamos la función
df['Rating'] = df.apply(normalizar , axis = 1)
```

```
#contamos la cantidad de palabras que hay en cada resenia
df['review_len'] = [len(text.split()) for text in df.Review]
```

```
#obtenemos la resenia con la mayor cantidad de palabras
df['review_len'].max()
```

```
#para que el modelo no sea tan pesado disminuimos el dataset a las resenias
#que no superen las 50 palabras
df = df[~(df['review_len'] > 30)]
```

```
#nuestro dataset se ha reducido a 80468 resenias es aceptable para entrenar y probar el modelo
df.info()
```

```
df['review_len'].max()
```

```
df
```

```
#cantidad de palabras en una resenia
from matplotlib import pyplot as plt
df['review_len'].plot(kind='hist', bins=20, title='review_len')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
#hay mas cantidad de resenias positivas que negativas por lo que el modelo aprendera
#rápidamente a identificar las resenias positivas
from matplotlib import pyplot as plt
df['Rating'].plot(kind='hist', bins=20, title='Rating')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
#al parecer hay una buena distribución de cantidad de palabras entre resenias positivas y negativas
from matplotlib import pyplot as plt
df.plot(kind='scatter', x='Rating', y='review_len', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
# Cargar el modelo preentrenado y el tokenizador de BERT para clasificación de secuencias
modelo_bert = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)
tokenizador_bert = BertTokenizer.from_pretrained("bert-base-uncased")



# Dividir el conjunto de datos en conjuntos de entrenamiento y prueba
train_texts, test_texts, train_labels, test_labels = train_test_split(df["Review"].values.tolist(), df["Rating"].values.tolist(), test_size=0.2, ra



# Tokenizar y preparar los datos
def tokenize_batch(texts):
    return tokenizador_bert(texts, padding=True, truncation=True, return_tensors="pt")

train_encodings = tokenize_batch(train_texts)
test_encodings = tokenize_batch(test_texts)

# Convertir etiquetas a tensores de PyTorch
train_labels = torch.tensor(train_labels)
test_labels = torch.tensor(test_labels)

# Crear conjuntos de datos de PyTorch
train_dataset = torch.utils.data.TensorDataset(train_encodings["input_ids"], train_encodings["attention_mask"], train_labels)
test_dataset = torch.utils.data.TensorDataset(test_encodings["input_ids"], test_encodings["attention_mask"], test_labels)

# Configuración del entrenamiento
batch_size = 50
epochs = 3
optimizer = torch.optim.AdamW(params=modelo_bert.parameters(), lr=5e-5)
loss_fn = torch.nn.CrossEntropyLoss()



# Specify the device (CPU or GPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Initialize model, tokenizer, and optimizer
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2).to(device)
optimizer = AdamW(model.parameters(), lr=5e-5)
scaler = GradScaler()

# Training loop with mixed precision
for epoch in tqdm(range(epochs), desc="Epochs"):
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    model.train()

    for batch in tqdm(train_loader, desc="Batches", leave=False):
        optimizer.zero_grad()
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)

        with autocast():
            outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
            loss = outputs.loss

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()



# Model evaluation
model.eval()
predictions = []

test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

with torch.no_grad():
    for batch in tqdm(test_loader, desc="Batches", leave=False):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)

        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        preds = torch.argmax(logits, dim=1)
        predictions.extend(preds.cpu().numpy())

# Convert predictions to numpy array
predictions = np.array(predictions)

# Calculate accuracy
accuracy = accuracy_score(test_labels.numpy(), predictions)
print(f"Accuracy: {accuracy:.4f}")
# Generate and print classification report
classification_report_str = classification_report(test_labels.numpy(), predictions)
print("Classification Report:\n", classification_report_str)

# Generate and plot confusion matrix
conf_matrix = confusion_matrix(test_labels.numpy(), predictions)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Puede predecir con bastante precisión cuando son resenias positivas más que negativas pero ello se debe a que al disminuir el dataset a 30 palabras observamos que las resenias positivas son mayores que las negativas lo que explica que el modelo refleje estos valores. Si utilizamos mayor cantidad de palabras consume más recursos y por ende se interrumpe la conexión del entorno

TENSOR FLOW KERAS

```
#dividimos los datos
X_train , X_val , Y_train , Y_val = train_test_split(df['Review'] , df['Rating'] , train_size = 0.80 , random_state = 42)
```

```
#usamos corpus para extraer las palabras más comunes y poder tokenizar
corpus = [word for text in df['Review'] for word in text.split()]
words_count = Counter(corpus)
sorted_words = words_count.most_common()
sorted_words
```

```
# definimos los parámetros para el tokenizador
VOCAB_SIZE = len(sorted_words)
EMBEDDING_DIM = 300
MAX_LEN = np.max(df['review_len'])
```

```
#armamos la funcion tokenizadora
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

def function_tokenizer(resenias_train) :
    tokenizer = Tokenizer(num_words=VOCAB_SIZE , oov_token='')
    tokenizer.fit_on_texts(resenias_train)
    seqs = tokenizer.texts_to_sequences(resenias_train)
    pad_seqs = pad_sequences(seqs , maxlen = MAX_LEN , padding='post')

    return pad_seqs
```

```
#aplicamos la tokenización
X_train = function_tokenizer(X_train)
X_val = function_tokenizer(X_val)
```

```
#creamos un modelo secuencial con una capa embebida donde pasamos los parametros
#2 capas bidireccionales
#2 capas dense una con activación relu y otra con sigmoide
model = Sequential([
    Embedding(VOCAB_SIZE + 1 , EMBEDDING_DIM , input_length=MAX_LEN) ,
    Bidirectional(LSTM(265 , return_sequences=True)) ,
    Bidirectional(LSTM(128)) ,
    Dense(64 , activation='relu') ,
    Dense(1 , activation='sigmoid')
])
```

```
#compilamos con entropía binaria
model.compile(
    loss = 'binary_crossentropy' ,
    optimizer = 'adam' ,
    metrics=['accuracy', Precision(name = 'precision'), Recall(name = 'recall')]
    )
```

```
model.summary()
```

```
history = model.fit(
    X_train ,
    Y_train ,
    epochs = 8,
    batch_size = 30,
    validation_data=(X_val , Y_val)
)
```

```
sns.set()
plt.figure(figsize=(12,6))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
```

```
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

```
y_val_pred = model.predict(X_val)
y_val_pred = y_val_pred.round()
```

```
lstm_cm = confusion_matrix(Y_val , y_val_pred)
sns.heatmap(lstm_cm, annot=True,fmt='3g')
plt.show()
```

No ha podido aprender con la precisión necesaria

Simple RNN

```
model_red = models.Sequential([
    layers.Embedding(VOCAB_SIZE + 1 , EMBEDDING_DIM , input_length=MAX_LEN),
    layers.SimpleRNN(32),
    layers.Dense(1, activation='sigmoid')
])

model_red.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy', Precision(name = 'precision'), Recall(name = 'recall')]
)

model_red.summary()
```

```
history = model_red.fit(
    X_train ,
    Y_train ,
    epochs = 10,
    batch_size = 64,
    validation_data=(X_val , Y_val)
)
```

```
sns.set()
plt.figure(figsize=(12,6))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

No ha podido aprender