MLP

pipeline con embeddings preentrenados en español (por ejemplo, FastText de Facebook o Word2Vec entrenado en Wikipedia en español). Esto te permitirá que el modelo entienda mejor:

- Los emojis y símbolos raros (que TF-IDF ignora).
- La semántica de palabras similares (ej. gratis, regalo, oferta → más cerca en el espacio vectorial).
- Los mensajes cortos que de otra forma perderían información.

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils.class_weight import compute_class_weight
import re
from sklearn.feature_extraction.text import TfidfVectorizer
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Embedding, Flatten
from tensorflow.keras.callbacks import EarlyStopping
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.optimizers import Adam, RMSprop, SGD, Adagrad
import fasttext
import fasttext.util
```

In [4]:
```
#!git clone https://github.com/facebookresearch/fastText.git
#!cd fastText
#!pip install fasttext-wheel
```

In [5]:
```
# ===== 1. Cargar datos =====
df = pd.read_excel("datasetv2.xlsx")
```

In [6]:
```
# ===== 2. Preprocesamiento =====
def limpiar_texto(texto):
    texto = texto.lower()
    texto = re.sub(r"http\S+|www\S+|https\S+", " url ", texto)  # URLs
    texto = re.sub(r"\d+", " num ", texto)  # números
    return texto.strip()
```

In [7]:
```
df["clean_message"] = df["message"].astype(str).apply(limpiar_texto)
```

In [11]:
```
# ===== 3. Cargar embeddings FastText =====
# Descargar modelo español: https://fasttext.cc/docs/en/crawl-vectors.html
# fasttext.util.download_model('es', if_exists='ignore')  # solo la 1ra vez
ft = fasttext.load_model("cc.es.300.bin")  # modelo preentrenado en español
```

In [12]:
```
# ===== 4. Convertir mensajes a embeddings =====
def texto_a_vector(texto):
    palabras = texto.split()
    if not palabras:
        return np.zeros(300)
    vectores = [ft.get_word_vector(p) for p in palabras]
    return np.mean(vectores, axis=0)  # promedio de embeddings

X = np.vstack(df["clean_message"].apply(texto_a_vector))
```

In [13]:
```
# ===== 5. Etiquetas =====
encoder = LabelEncoder()
y = encoder.fit_transform(df["target"])  # spam=1, legit=0
```

In [14]:
```
# ===== 6. Train/Test Split =====
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [23]:
```
# ===== 7. Modelo MLP =====
model = Sequential([
    Dense(128, activation="relu"),
    Dropout(0.4),
    Dense(64, activation="relu"),
    Dropout(0.3),
    Dense(1, activation="sigmoid")
])

model.compile(optimizer=Adam(learning_rate=0.0001),
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

In [24]:
```
#agregamos EarlyStopping
# Definimos el callback
early_stop = EarlyStopping(
    monitor='val_loss',         # Métrica a observar
    patience=5,                 # Cuántas epochs esperar sin mejora
    restore_best_weights=True   # Recuperar los mejores pesos
)
```

In [25]:
```
# ===== 8. Entrenamiento =====
history = model.fit(X_train, y_train,
                    validation_data=(X_test, y_test),
                    epochs=50, batch_size=32,callbacks=[early_stop])
```

```
Epoch 1/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7464 - loss: 0.6370 - val_
accuracy: 0.8635 - val_loss: 0.5235
Epoch 2/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8688 - loss: 0.4283 - val_
accuracy: 0.8836 - val_loss: 0.3329
Epoch 3/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8890 - loss: 0.3162 - val_
accuracy: 0.8942 - val_loss: 0.2800
Epoch 4/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9010 - loss: 0.2773 - val_
accuracy: 0.9025 - val_loss: 0.2556
Epoch 5/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9085 - loss: 0.2551 - val_
accuracy: 0.9167 - val_loss: 0.2383
Epoch 6/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9161 - loss: 0.2378 - val_
accuracy: 0.9202 - val_loss: 0.2252
Epoch 7/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9183 - loss: 0.2216 - val_
accuracy: 0.9249 - val_loss: 0.2146
Epoch 8/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9206 - loss: 0.2137 - val_
accuracy: 0.9267 - val_loss: 0.2062
Epoch 9/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9260 - loss: 0.2038 - val_
accuracy: 0.9303 - val_loss: 0.2003
Epoch 10/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9285 - loss: 0.1967 - val_
accuracy: 0.9309 - val_loss: 0.1951
Epoch 11/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9307 - loss: 0.1905 - val_
accuracy: 0.9314 - val_loss: 0.1921
Epoch 12/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9296 - loss: 0.1869 - val_
accuracy: 0.9320 - val_loss: 0.1883
Epoch 13/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9342 - loss: 0.1831 - val_
accuracy: 0.9338 - val_loss: 0.1849
Epoch 14/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9366 - loss: 0.1825 - val_
accuracy: 0.9338 - val_loss: 0.1829
Epoch 15/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9333 - loss: 0.1767 - val_
accuracy: 0.9344 - val_loss: 0.1818
Epoch 16/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9339 - loss: 0.1742 - val_
accuracy: 0.9374 - val_loss: 0.1791
Epoch 17/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9379 - loss: 0.1694 - val_
accuracy: 0.9374 - val_loss: 0.1772
Epoch 18/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9393 - loss: 0.1663 - val_
accuracy: 0.9374 - val_loss: 0.1763
Epoch 19/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9419 - loss: 0.1658 - val_
accuracy: 0.9374 - val_loss: 0.1754
Epoch 20/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9409 - loss: 0.1592 - val_
accuracy: 0.9385 - val_loss: 0.1743
```

```
Epoch 21/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9410 - loss: 0.1620 - val_
accuracy: 0.9356 - val_loss: 0.1741
Epoch 22/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9428 - loss: 0.1559 - val_
accuracy: 0.9397 - val_loss: 0.1717
Epoch 23/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9415 - loss: 0.1551 - val_
accuracy: 0.9397 - val_loss: 0.1705
Epoch 24/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9421 - loss: 0.1523 - val_
accuracy: 0.9385 - val_loss: 0.1701
Epoch 25/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9434 - loss: 0.1485 - val_
accuracy: 0.9403 - val_loss: 0.1704
Epoch 26/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9415 - loss: 0.1510 - val_
accuracy: 0.9409 - val_loss: 0.1696
Epoch 27/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9459 - loss: 0.1469 - val_
accuracy: 0.9397 - val_loss: 0.1680
Epoch 28/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9438 - loss: 0.1484 - val_
accuracy: 0.9397 - val_loss: 0.1665
Epoch 29/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9466 - loss: 0.1424 - val_
accuracy: 0.9397 - val_loss: 0.1666
Epoch 30/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9472 - loss: 0.1423 - val_
accuracy: 0.9409 - val_loss: 0.1662
Epoch 31/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9492 - loss: 0.1399 - val_
accuracy: 0.9403 - val_loss: 0.1653
Epoch 32/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9497 - loss: 0.1393 - val_
accuracy: 0.9397 - val_loss: 0.1642
Epoch 33/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9496 - loss: 0.1386 - val_
accuracy: 0.9403 - val_loss: 0.1649
Epoch 34/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9500 - loss: 0.1356 - val_
accuracy: 0.9409 - val_loss: 0.1635
Epoch 35/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9527 - loss: 0.1361 - val_
accuracy: 0.9409 - val_loss: 0.1643
Epoch 36/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9514 - loss: 0.1324 - val_
accuracy: 0.9403 - val_loss: 0.1620
Epoch 37/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9506 - loss: 0.1337 - val_
accuracy: 0.9433 - val_loss: 0.1629
Epoch 38/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9521 - loss: 0.1306 - val_
accuracy: 0.9427 - val_loss: 0.1614
Epoch 39/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9529 - loss: 0.1298 - val_
accuracy: 0.9409 - val_loss: 0.1610
Epoch 40/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9546 - loss: 0.1288 - val_
accuracy: 0.9433 - val_loss: 0.1599
```

```
Epoch 41/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9496 - loss: 0.1290 - val_
accuracy: 0.9427 - val_loss: 0.1597
Epoch 42/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9545 - loss: 0.1278 - val_
accuracy: 0.9444 - val_loss: 0.1616
Epoch 43/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9533 - loss: 0.1270 - val_
accuracy: 0.9421 - val_loss: 0.1589
Epoch 44/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9533 - loss: 0.1240 - val_
accuracy: 0.9433 - val_loss: 0.1588
Epoch 45/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9549 - loss: 0.1234 - val_
accuracy: 0.9450 - val_loss: 0.1582
Epoch 46/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9564 - loss: 0.1210 - val_
accuracy: 0.9450 - val_loss: 0.1583
Epoch 47/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9564 - loss: 0.1210 - val_
accuracy: 0.9462 - val_loss: 0.1582
Epoch 48/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9551 - loss: 0.1203 - val_
accuracy: 0.9456 - val_loss: 0.1580
Epoch 49/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9574 - loss: 0.1169 - val_
accuracy: 0.9450 - val_loss: 0.1576
Epoch 50/50
212/212 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9583 - loss: 0.1179 - val_
accuracy: 0.9462 - val_loss: 0.1563
```

In [26]:

```python
# ===== 9. Evaluación =====
loss, acc = model.evaluate(X_test, y_test)
print(f"Accuracy con embeddings: {acc:.4f}")
```

```
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9462 - loss: 0.1563
Accuracy con embeddings: 0.9462
```

In [27]:

```python
y_pred = (model.predict(X_test) > 0.5).astype("int32")

print("\nClassification Report:\n", classification_report(y_test, y_pred, target
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

Classification Report:
               precision    recall  f1-score   support

       legit       0.96      0.95      0.95       981
        spam       0.93      0.94      0.94       711

    accuracy                           0.95      1692
   macro avg       0.94      0.95      0.94      1692
weighted avg       0.95      0.95      0.95      1692


Confusion Matrix:
 [[933  48]
 [ 43 668]]
```

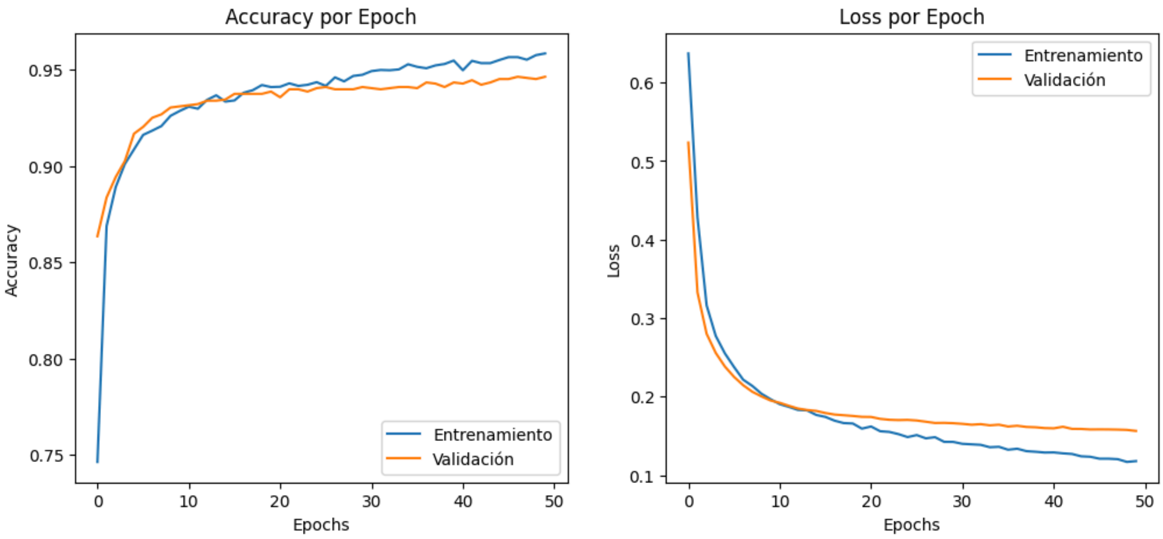
In [28]:

```
# ========================
# 8. Gráfico de entrenamiento
# ========================
plt.figure(figsize=(12,5))

# Accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Accuracy por Epoch')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Loss por Epoch')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



In [ ]: