06_red_neuronal_gru

November 20, 2025

1 Red Neuronal con GRU

[1]:
```
from pathlib import Path
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, KFold
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import f1_score
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

import json

DATA_DIR = Path('../data/nlp-getting-started')
TRAIN_PATH = DATA_DIR / 'train.csv'
TEST_PATH = DATA_DIR / 'test.csv'
LOCATION_TO_COUNTRY_PATH = Path('../data/location_to_country.json')
RANDOM_SEED = 27

train_df = pd.read_csv(TRAIN_PATH)
test_df = pd.read_csv(TEST_PATH)

with open(LOCATION_TO_COUNTRY_PATH, 'r', encoding='utf-8') as f:
    location_to_country = json.load(f)

# Categóricas: 'country',  'keyword'. Después las voy a mean encodear.
train_df['country'] = train_df['location'].map(location_to_country).
 ↪fillna('unknown')
test_df['country'] = test_df['location'].map(location_to_country).
 ↪fillna('unknown')

train_df['keyword'] = train_df['keyword'].fillna('missing')
test_df['keyword'] = test_df['keyword'].fillna('missing')
```

```
categorical_features = ['country', 'keyword']

# Numéricas: 'text_length', 'num_hashtags', 'num_mentions',␣
 ↪'num_uppercase_per_word', 'sentiment_score', 'has_url'

# one hot encoding de 'has_url' a mano
train_df['has_url'] = train_df['text'].fillna('').str.contains(r'http[s]?://',␣
 ↪regex=True).astype(int)
test_df['has_url'] = test_df['text'].fillna('').str.contains(r'http[s]?://',␣
 ↪regex=True).astype(int)

train_df['text_length'] = train_df['text'].fillna('').str.len()
test_df['text_length'] = test_df['text'].fillna('').str.len()

train_df['num_hashtags'] = train_df['text'].str.count('#')
train_df['num_mentions'] = train_df['text'].str.count('@')

test_df['num_hashtags'] = test_df['text'].str.count('#')
test_df['num_mentions'] = test_df['text'].str.count('@')

def uppercase_per_word(text):
    text = str(text)

    # Palabras que tengan al menos una letra alfabética
    words = [w for w in text.split() if any(ch.isalpha() for ch in w)]
    if not words:
        return 0.0

    # Solo letras alfabéticas, para evitar que cuenten símbolos raros
    uppercase_letters = sum(ch.isupper() for ch in text if ch.isalpha())
    return uppercase_letters / len(words)


train_df['num_uppercase_per_word'] = train_df['text'].apply(uppercase_per_word)
test_df['num_uppercase_per_word']  = test_df['text'].apply(uppercase_per_word)

analyzer = SentimentIntensityAnalyzer()

def get_sentiment(text):
    if pd.isna(text) or text.strip() == '':
        return 0.5
    compound = analyzer.polarity_scores(text)['compound']
    return (compound + 1) / 2

train_df['sentiment_score'] = train_df['text'].apply(get_sentiment)
test_df['sentiment_score'] = test_df['text'].apply(get_sentiment)
```

```
numeric_features = ['text_length', 'num_hashtags', 'num_mentions',␣
 ↪'num_uppercase_per_word', 'sentiment_score', 'has_url']

embedding_feature = 'text'

# 1. Separar features y target
X = train_df[numeric_features + categorical_features + [embedding_feature]].
 ↪copy()
y = train_df['target'].copy()

# 2. Split estratificado train/validation (80/20)
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=RANDOM_SEED, stratify=y
)

X_train.head()
```

[1]:

	text_length	num_hashtags	num_mentions	num_uppercase_per_word	\
2721	87	1	0	1.000000	
2259	132	0	0	0.083333	
1815	136	0	0	1.333333	
682	139	0	0	1.666667	
7216	121	0	2	0.210526	

	sentiment_score	has_url	country	keyword	\
2721	0.18755	1	unknown	devastated	
2259	0.55135	0	unknown	deluged	
1815	0.27205	1	United Kingdom	crashed	
682	0.50000	1	unknown	blazing	
7216	0.78595	0	United States of America	weapons	

	text
2721	Obama declares disaster for typhoon-devastated…
2259	Businesses are deluged with invzices. Make you…
1815	Neil_Eastwood77: I AM A KNOBHEAD!! Bin Laden f…
682	Morgan Silver Dollar 1880 S Gem BU DMPL Cameo …
7216	@danagould @WaynesterAtl I agree with backgrou…

1.0.1 Mismo Mean Encoding con KFold de antes

[2]:
```
from sklearn.model_selection import KFold

def kfold_target_encoding(train_series, target_series, n_splits=5,␣
 ↪random_state=RANDOM_SEED):
    encoded = pd.Series(np.nan, index=train_series.index, dtype=float)
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=random_state)
    global_mean = target_series.mean()
```

```
    for tr_idx, val_idx in kf.split(train_series):
        fold_df = pd.DataFrame({
            'feature': train_series.iloc[tr_idx],
            'target': target_series.iloc[tr_idx]
        })
        means = fold_df.groupby('feature')['target'].mean()
        encoded.iloc[val_idx] = train_series.iloc[val_idx].map(means)

    encoded.fillna(global_mean, inplace=True)

    full_df = pd.DataFrame({'feature': train_series, 'target': target_series})
    mapping = full_df.groupby('feature')['target'].mean()

    return encoded, mapping, global_mean

mean_encoded_features = []

for col in ['country', 'keyword']:
    train_encoded, mapping, global_mean = kfold_target_encoding(
        X_train[col], y_train
    )
    new_col = f'{col}_target_mean'
    X_train[new_col] = train_encoded
    X_val[new_col]   = X_val[col].map(mapping).fillna(global_mean)
    test_df[new_col] = test_df[col].map(mapping).fillna(global_mean)
    mean_encoded_features.append(new_col)

# Actualizamos lista de numéricas
numeric_features = numeric_features + mean_encoded_features
```

Normalizamos todas las features para darle a la Red neuronal un mejor input. Podría ponerlo en el pipeline, pero como no quiero cambiar hiperparámetros de esto, lo hago aparte.

[3]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_num = scaler.fit_transform(X_train[numeric_features])
X_val_num   = scaler.transform(X_val[numeric_features])
X_test_num  = scaler.transform(test_df[numeric_features])
```

[65]:
```
tweets_len = X_train[text_col].fillna('').str.split().apply(len)
print(tweets_len.describe())
print("Percentiles:")
print(tweets_len.quantile([0.75, 0.90, 0.95, 0.99]))
```

```
count    6090.000000
mean       14.863383
```

```
std          5.747268
min          1.000000
25%         11.000000
50%         15.000000
75%         19.000000
max         31.000000
Name: text, dtype: float64
Percentiles:
0.75    19.0
0.90    22.0
0.95    24.0
0.99    27.0
Name: text, dtype: float64
```

Preparamos el texto para la GRU (tokenizer + padding)

[86]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

text_col = embedding_feature
MAX_WORDS = 10000
MAX_LEN   = 30

tokenizer = Tokenizer(num_words=MAX_WORDS, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train[text_col].fillna(''))

X_train_seq = pad_sequences(
    tokenizer.texts_to_sequences(X_train[text_col].fillna('')),
    maxlen=MAX_LEN
)
X_val_seq = pad_sequences(
    tokenizer.texts_to_sequences(X_val[text_col].fillna('')),
    maxlen=MAX_LEN
)
X_test_seq = pad_sequences(
    tokenizer.texts_to_sequences(test_df[text_col].fillna('')),
    maxlen=MAX_LEN
)
```

Preparado de Tensorflow. Recomendaciones e instalación de internet.

[59]:
```
import os
import tensorflow as tf
from tensorflow.keras import mixed_precision
from tensorflow.keras.layers import Input, Embedding, GRU, Bidirectional,␣
 ↪Dense, Dropout, Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
```

```
os.environ["TF_ENABLE_ONEDNN_OPTS"] = "0"
mixed_precision.set_global_policy("mixed_float16")

gpus = tf.config.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)

print("TF version:", tf.__version__)
print("Physical GPUs:", tf.config.list_physical_devices('GPU'))
print("Logical GPUs:", tf.config.experimental.list_logical_devices('GPU'))
```

```
TF version: 2.20.0
Physical GPUs: [PhysicalDevice(name='/physical_device:GPU:0',
device_type='GPU')]
Logical GPUs: [LogicalDevice(name='/device:GPU:0', device_type='GPU')]
```

Me detecta mi GPU 4070ti, genial

[84]:
```
from tensorflow.keras.layers import GlobalMaxPooling1D

# Rama texto
text_input = Input(shape=(MAX_LEN,), name="text_input")
x = Embedding(MAX_WORDS, 128, name="embedding")(text_input)
x = Bidirectional(GRU(96, return_sequences=True, dropout=0.3,␣
 ↪recurrent_dropout=0.2))(x)
x = GlobalMaxPooling1D()(x)
x = Dropout(0.35)(x)

# Rama numérica
num_input = Input(shape=(n_num_features,), name="num_input")
n = Dense(64, activation='relu', kernel_regularizer=l2(1e-4))(num_input)
n = Dropout(0.25)(n)

combined = Concatenate()([x, n])
h = Dense(96, activation='relu', kernel_regularizer=l2(1e-4))(combined)
h = Dropout(0.3)(h)

output = Dense(1, activation='sigmoid', dtype='float32')(h)

model = Model(inputs=[text_input, num_input], outputs=output)
optimizer = tf.keras.optimizers.Adam(learning_rate=2e-4)
model.compile(optimizer=optimizer, loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=5,
```

```
    restore_best_weights=True
)
```

Entrenamiento de la red neuronal con GRU y features numéricas adicionales.

[87]:
```
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

history = model.fit(
    {'text_input': X_train_seq, 'num_input': X_train_num},
    y_train,
    validation_data=(
        {'text_input': X_val_seq, 'num_input': X_val_num},
        y_val
    ),
    epochs=40,
    batch_size=batch_size,
    callbacks=[early_stop],
    verbose=2
)
```

```
Epoch 1/40
191/191 - 7s - 38ms/step - accuracy: 0.6782 - loss: 0.6270 - val_accuracy:
0.7715 - val_loss: 0.5328
Epoch 2/40
191/191 - 7s - 36ms/step - accuracy: 0.7378 - loss: 0.5532 - val_accuracy:
0.7827 - val_loss: 0.4886
Epoch 3/40
191/191 - 7s - 36ms/step - accuracy: 0.7798 - loss: 0.4962 - val_accuracy:
0.8168 - val_loss: 0.4319
Epoch 4/40
191/191 - 7s - 36ms/step - accuracy: 0.8415 - loss: 0.3903 - val_accuracy:
0.8286 - val_loss: 0.4227
Epoch 5/40
191/191 - 7s - 36ms/step - accuracy: 0.8828 - loss: 0.3071 - val_accuracy:
0.8070 - val_loss: 0.4553
Epoch 6/40
191/191 - 7s - 36ms/step - accuracy: 0.9097 - loss: 0.2493 - val_accuracy:
0.8030 - val_loss: 0.5036
Epoch 7/40
191/191 - 7s - 36ms/step - accuracy: 0.9337 - loss: 0.2005 - val_accuracy:
0.7873 - val_loss: 0.5853
```

Ah overfittea zarpado. No llega a 2 epochs y ya empieza a subir el loss de validación. Tengo que mejorarlo

[88]:
```
from sklearn.metrics import f1_score
import numpy as np

y_val_proba = model.predict(
    {'text_input': X_val_seq, 'num_input': X_val_num},
    batch_size=256
).ravel()

thresholds = np.linspace(0.2, 0.8, 61)
best_th, best_f1 = 0.5, 0

for th in thresholds:
    y_pred = (y_val_proba >= th).astype(int)
    f1 = f1_score(y_val, y_pred)
    if f1 > best_f1:
        best_f1, best_th = f1, th

print(f"Mejor threshold: {best_th:.2f} con F1 = {best_f1:.4f}")
```

```
6/6             1s 138ms/step
Mejor threshold: 0.52 con F1 = 0.7922
```

[89]:
```
# Predicción en validación
y_val_pred_proba = model.predict(
    {'text_input': X_val_seq, 'num_input': X_val_num},
    batch_size=batch_size
).ravel()

# Convertimos probabilidades a clases binarias (threshold 0.5)
y_val_pred = (y_val_pred_proba >= 0.5).astype(int)

# Calculamos F1
f1_val = f1_score(y_val, y_val_pred)
print(f"F1-score en validación: {f1_val:.4f}")


y_train_pred_proba = model.predict(
    {'text_input': X_train_seq, 'num_input': X_train_num},
    batch_size=batch_size
).ravel()

y_train_pred = (y_train_pred_proba >= 0.5).astype(int)
f1_train = f1_score(y_train, y_train_pred)

print(f"F1 en entrenamiento: {f1_train:.4f}")
print(f"F1 en validación: {f1_val:.4f}")
```

```
48/48             0s 10ms/step
```

```
F1-score en validación: 0.7900
191/191             2s 9ms/step
F1 en entrenamiento: 0.8711
F1 en validación: 0.7900
```

Bueno, qué decir, la red neuronal con GRU entrenada con las features numéricas adicionales logró un F1 en validación de 0.79. Es lo mejor hasta ahora. No entendí todo la verdad, me encantaría profundizar más en redes neuronales y entender cada hiperparámetro, seguro podría mejorarlo mucho pero por ahora me quedo contento con este resultado.

[90]:
```python
y_test_pred_proba = model.predict(
    {'text_input': X_test_seq, 'num_input': X_test_num}
).ravel()
y_test_pred = (y_test_pred_proba >= 0.5).astype(int)

submission = pd.DataFrame({
    'id': test_df['id'],
    'target': y_test_pred
})

from pathlib import Path
submissions_dir = Path('../resultados')
submissions_dir.mkdir(parents=True, exist_ok=True)
submission_path = submissions_dir / 'red_neuronal_gru.csv'
submission.to_csv(submission_path, index=False)
print(f"Submission guardada en: {submission_path}")
```

```
102/102             1s 9ms/step
Submission guardada en: ../resultados/red_neuronal_gru.csv
```