Inicialización

In [1]:
```
import numpy as np
import pandas as pd

import torch
import transformers

from tqdm.auto import tqdm

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

Variables globales

In [2]:
```
max_sample_size = 200
```

Cargar datos

In [3]:
```
df_reviews = pd.read_csv('/datasets/imdb_reviews_200.tsv', sep='	')
```

Preprocesamiento para BERT

In [4]:
```
tokenizer = transformers.BertTokenizer.from_pretrained('bert-base-uncased')

ids_list = []
attention_mask_list = []

max_length = 512

for input_text in df_reviews.iloc[:max_sample_size]['review']:
    ids = tokenizer.encode(input_text.lower(), add_special_tokens=True, truncation=True
    padded = np.array(ids + [0]*(max_length - len(ids)))
    attention_mask = np.where(padded != 0, 1, 0)
    ids_list.append(padded)
    attention_mask_list.append(attention_mask)
```

Obtener insertados

In [5]:
```
config = transformers.BertConfig.from_pretrained('bert-base-uncased')
model = transformers.BertModel.from_pretrained('bert-base-uncased')
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializin
g BertModel: ['cls.seq_relationship.bias', 'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.bias', 'cls.predictions.transform.dense.bias', 'cls.seq_relationship.we
ight', 'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.transform.dense.weig
ht', 'cls.predictions.decoder.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trai
ned on another task or with another architecture (e.g. initializing a BertForSequenceCla
ssification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model
that you expect to be exactly identical (initializing a BertForSequenceClassification mo
del from a BertForSequenceClassification model).
```

In [ ]:

In [6]:

```
batch_size = 25    # por lo general, el tamaño del lote es igual a 100, pero lo podemos

embeddings = []

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Uso del dispositivo {device}.')
model.to(device)

for i in tqdm(range(len(ids_list) // batch_size)):

    ids_batch = torch.LongTensor(ids_list[batch_size*i:batch_size*(i+1)]).to(device)
    attention_mask_batch = torch.LongTensor(attention_mask_list[batch_size*i:batch_size

    with torch.no_grad():
        model.eval()
        batch_embeddings = model(ids_batch, attention_mask=attention_mask_batch)

    embeddings.append(batch_embeddings[0][:,0,:].detach().cpu().numpy())
```

```
Uso del dispositivo cpu.

/tmp/ipykernel_28/3344139633.py:11: UserWarning: Creating a tensor from a list of numpy.
ndarrays is extremely slow. Please consider converting the list to a single numpy.ndarra
y with numpy.array() before converting to a tensor. (Triggered internally at  ../torch/c
src/utils/tensor_new.cpp:201.)
  ids_batch = torch.LongTensor(ids_list[batch_size*i:batch_size*(i+1)]).to(device)
```

Modelo

In [7]:

```
features = np.concatenate(embeddings)
target = df_reviews.iloc[:max_sample_size]['pos']

print(features.shape)
print(target.shape)
```

```
(200, 768)
(200,)
```

In [8]:

```
# entrena y prueba tu modelo
# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.5, ra
```

```
# Inicializar el modelo de regresión logística
logistic_model = LogisticRegression(max_iter=1000)

# Entrenar el modelo
logistic_model.fit(X_train, y_train)

# Calcular la exactitud en el conjunto de entrenamiento
train_accuracy = logistic_model.score(X_train, y_train)
print(f"Exactitud en el conjunto de entrenamiento: {train_accuracy:.2f}")
```

```
Exactitud en el conjunto de entrenamiento: 1.00
```

In [ ]: