Predicción de sarcasmo usando DistilBERT

Trabajo práctico final de la materia LLM e IA generativa de la CEIA del laboratorio de sistemas embebidos de la UBA

Alumno: Rosito Pedro

Planteo del problema

Se buscó realizar *fine-tuning* sobre el modelo preentrenado DistilBERT y verificar si se obtienen buenos resultados para la predicción de sarcasmo en textos obtenidos de titulares de noticias.

El dataset utilizado fue:

“News Headlines Dataset For Sarcasm Detection” [1][2]

Se decidió utilizar DistilBERT [3] por dos motivos:

- Lo que hace único a BERT y por ende a DistilBERT, es su capacidad para captar el contexto bidireccional en una oración, lo que significa que puede entender el significado de una palabra en relación con las palabras que la rodean. Esta mejora en la comprensión contextual ha llevado a un rendimiento sobresaliente en diversas tareas de procesamiento del lenguaje natural. [4]
- DIstilBERT es una versión más pequeña, rápida y barata que se desprende de BERT. Tiene un 40% menos de parámetros que “bert-base-uncased”, corre un 60% más rápido y preserva un 95% de la performance de BERT en el *benchmark* “GLUE language understanding”.

Como en este caso se tiene poca capacidad de cómputo ya que se utilizará Google Collaboratory y, por otro lado, se busca detectar sarcasmo, por lo que se necesita una gran “comprensión” del contexto por parte del modelo, DistilBERT se presenta como una buena alternativa para resolver este problema.

El modelo entrenado deberá ser capaz de predecir si una frase es o no sarcástica. Para esto se buscará obtener un buen desempeño en la métrica accuracy, ya que en el caso de la predicción de sarcasmo los falsos positivos y los falsos negativos son igualmente costosos.

Procedimiento

1.

```
import wandb
wandb.login()
```

Se utiliza Wandb [5] para graficar las métricas y algunos otros parámetros del entrenamiento en tiempo real.

2.

```
from transformers import DistilBertTokenizerFast
tokenizer=DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')
```

Se utiliza el *tokenizer* DistilBertTokenizerFast [6] basado en WordPiece [7] y se obtienen los pesos preentrenados de "distilbert-base-uncased". Se utiliza este tokenizer ya que es el que se encuentra asociado al modelo DistilBERT.

3.

```
from transformers import DataCollatorWithPadding
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

Se define el data_collator que se utilizará para entrenar el modelo. El data_collator generará batches de muestras desde el dataset y completará cada batch (padding) para que las secuencias que contenga sean del largo de la secuencia más larga. Toma el *tokenizer* para saber el token que debe utilizar para realizar el padding.

4.

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=1,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=16,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
)
```

Se definen los argumentos para pasarle a la función de entrenamiento.
output_dir: carpeta en la que se van a guardar los resultados del modelo.
num_train_epochs: cantidad de épocas en las que se va a entrenar el modelo. Se utilizó 1 para evitar que el entrenamiento tomara demasiado tiempo.

per_device_train_batch_size: tamaño de batch por unidad de procesamiento utilizado para entrenar. Se utiliza 32 ya que es un compromiso entre el tiempo de entrenamiento y la convergencia del modelo. Con batchs más chicos el entrenamiento se hace más lento y la mejora en los resultados no es tan grande como para hacer valer la modificación.
per_device_eval_batch_size: tamaño de batch por unidad de procesamiento utilizado para evaluar.
warmup_steps: el objetivo de este parámetro es indicar cuantos pasos utilizará el modelo para “calentar”. El calentamiento consiste en utilizar un *learning rate* muy bajo durante la cantidad de pasos que se indican y luego utilizar el *learning rate* normal para el resto del entrenamiento. Este “calentamiento” se utiliza para evitar un *overfitting* del modelo temprano cuando está expuesto a nuevos datos.
weight decay: el *weight decay* aplicado a todas las capas, excepto al bias y a los pesos de la capa normal del optimizador AdamW.

5. Se añade una clase CustomTrainer que hereda de Trainer y añade a los logs generados por la capa Trainer la métrica accuracy, esto se realiza con el objetivo de poder observar un plot en tiempo real de la métrica, utilizando la plataforma Wandb. [8]

6.

```
model=DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased")

trainer = CustomTrainer(
   model=model,
   args=training_args,
   train_dataset=tokenized_datasets['train'],
   eval_dataset=tokenized_datasets['eval'],
   data_collator=data_collator,
   compute_metrics=compute_metrics
)

trainer.train()
wandb.finish()
```

Se instancia el modelo a partir de los pesos preentrenados de distilbert-base-uncased. Luego se define un trainer instanciando la clase CustomTrainer creada y se le pasan los parámetros definidos anteriormente. Finalmente se llama al método train para entrenar el modelo, una vez finalizado el entrenamiento se cierra la conexión con Wandb.

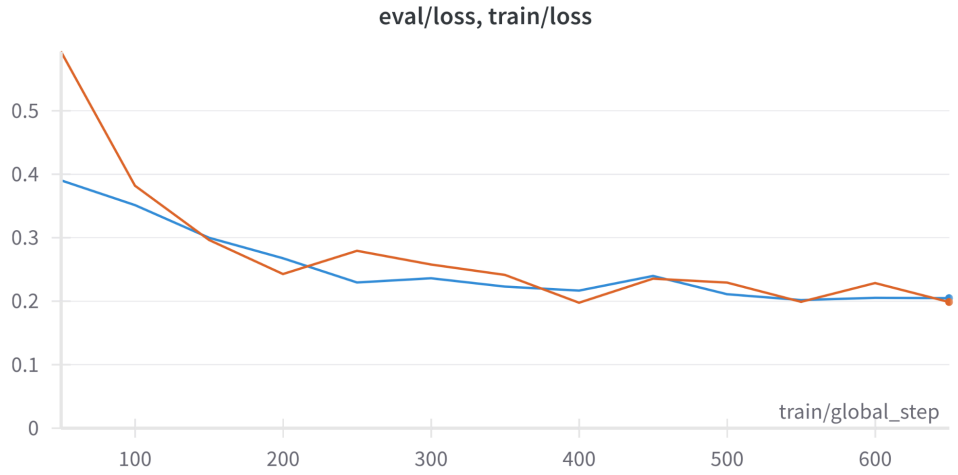
7. Por último, se realiza la evaluación del modelo y se obtienen los resultados finales.

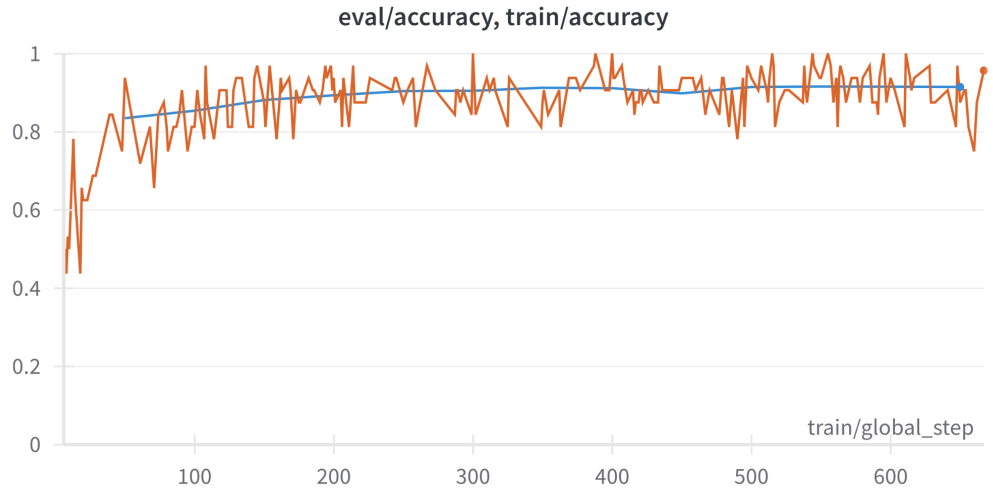
Resultados

Se entrenó el modelo utilizando diferentes parámetros obteniéndose los siguientes resultados (las curvas naranjas representan el entrenamiento y las curvas celestes la validación):

Primer entrenamiento:
per_device_train_batch_size=32, warmup_steps=50, épocas=1

	precision	recall	f1-score	support
0	0.918	0.938	0.928	2996
1	0.918	0.893	0.905	2346
accuracy			0.918	5342
macro avg	0.918	0.915	0.916	5342
weighted avg	0.918	0.918	0.918	5342

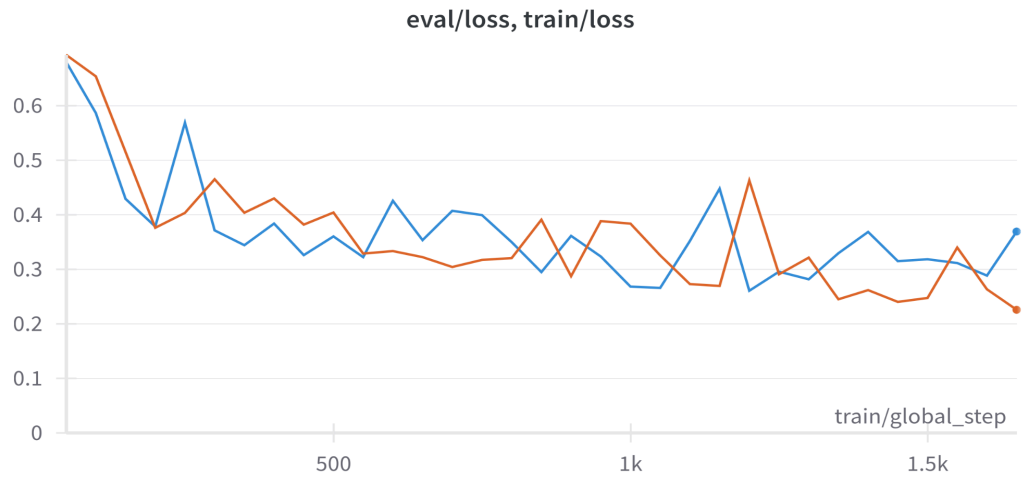


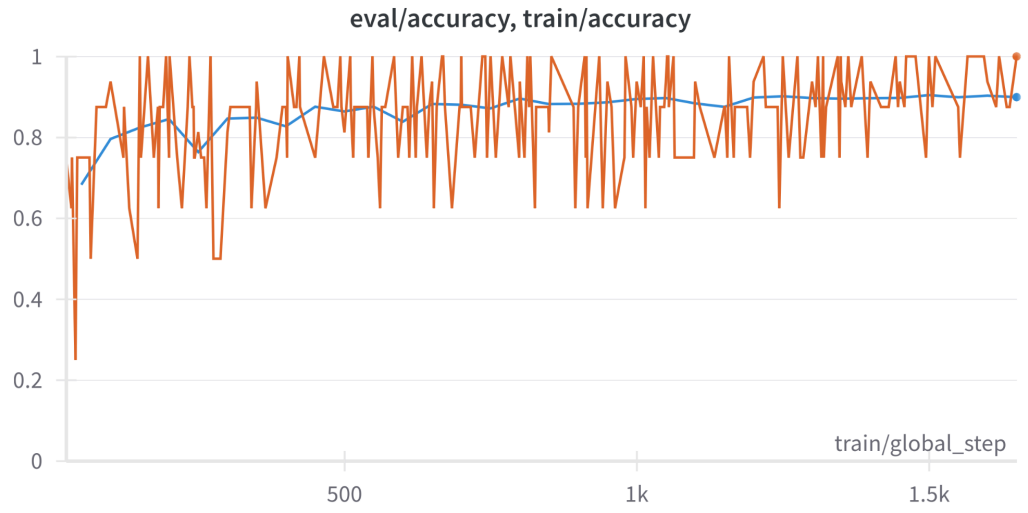


En los siguientes entrenamientos se buscó aumentar el número de épocas y disminuir el tamaño del batch de entrenamiento para ver si se obtenían mejores resultados, pero debido a las limitaciones del colab gratuito, ninguno llegó a terminar de ejecutarse, pese a haber introducido callbacks de *early stopping* por accuracy y eval loss. Se muestra únicamente el entrenamiento que mejores resultados obtuvo.

per_device_train_batch_size=8, warmup_steps=500, épocas=2

	precision	recall	f1-score	support
0	0.897	0.937	0.917	2996
1	0.915	0.863	0.888	2346
accuracy			0.904	5342
macro avg	0.906	0.900	0.902	5342
weighted avg	0.905	0.904	0.904	5342





Conclusiones

A partir de los resultados observados se concluye que:

- El modelo obtuvo buenos resultados para la predicción de sarcasmo.
- El utilizar batches de entrenamiento de menor tamaño y aumentar el número de épocas, no mejoró el rendimiento del modelo. Esto se debe probablemente al hecho de que, por el tamaño y la forma del dataset, el modelo tiende a hacer overfitting. Este hecho se observa en la gran variación en el accuracy de entrenamiento y en cómo el train loss disminuye mientras que el eval loss no disminuye de la misma manera y en algunos casos aumenta.
- Se utilizó la técnica de early stopping sin obtener buenos resultados.

De lo anterior también se desprende que parece razonable utilizar batches de entrenamiento un poco más grandes y tal vez se podría combinar con técnicas de *data augmentation* para obtener resultados aún mejores.

Referencias

1. Misra, Rishabh and Prahal Arora. "Sarcasm Detection using News Headlines Dataset." AI Open (2023).
2. Misra, Rishabh and Jigyasa Grover. "Sculpting Data for ML: The first act of Machine Learning." ISBN 9798585463570 (2021).
3. https://arxiv.org/pdf/1910.01108
4. https://es.wikipedia.org/wiki/BERT_(modelo_de_lenguaje)
5. https://wandb.ai
6.https://huggingface.co/transformers/v3.0.2/model_doc/distilbert.html#distilberttokenizerfast
7. https://arxiv.org/pdf/1609.08144v2
8. https://discuss.huggingface.co/t/metrics-for-training-set-in-trainer/2461/4