Evaluación LLM

Lenin G. Falconí

2025-01-14

1. Métricas para clasificación

```
from pprint import pprint
from evaluate import load

accuracy = load("accuracy")
precision = load("precision")
recall = load("recall")
f1 = load("f1")

real_labels = [0,1,0,1,1]
predicted_labels = [0,0,0,1,1]
acc_val = accuracy.compute(references=real_labels, predictions=predicted_labels)
precision_val = precision.compute(references=real_labels, predictions=predicted_labels)
recall_val = recall.compute(references=real_labels, predictions=predicted_labels)
f1_val = f1.compute(references=real_labels, predictions=predicted_labels)

print(f"acc: {acc_val}")
print(f"precision: {precision_val}")
print(f"recall: {recall_val}")
print(f"f1: {f1_val}")
```

```
2025-01-15 12:03:18.284257: E tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342] Unable t
2025-01-15 12:03:18.284363: E tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] Unable to
2025-01-15 12:03:18.284424: E tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518] Unable
2025-01-15 12:03:18.318302: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow bina
To enable the following instructions: SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild Tensor
acc: {'accuracy': 0.8}
precision: {'precision': 1.0}
recall: {'recall': 0.6666666666666666}
f1: {'f1': 0.8}
```

Se presenta la descripción y las features de `Accuracy`

```
print("Accuracy description")
print(accuracy.description)
print("Accuracy Features")
print(accuracy.features)
```

```
Accuracy description

Accuracy is the proportion of correct predictions among the total number of cases processed. It can b
Accuracy = (TP + TN) / (TP + TN + FP + FN)
 Where:
TP: True positive
TN: True negative
```

```
FP: False positive
FN: False negative

Accuracy Features
{'predictions': Value(dtype='int32', id=None), 'references': Value(dtype='int32', id=None)}
```

Se presenta las features y la descripción de la métrica `Precision`

```
print("Description")
print(precision.description)
print("Features")
print(precision.features)
```

```


Description

Precision is the fraction of correctly labeled positive examples out of all of the examples that were
Precision = TP / (TP + FP)
where TP is the True positives (i.e. the examples correctly labeled as positive) and FP is the False

Features
{'predictions': Value(dtype='int32', id=None), 'references': Value(dtype='int32', id=None)}
```

Se presenta las features y la descripcion de `Recall`

```
print("Description")
print(recall.description)
print("Features")
print(recall.features)
```

```


Description

Recall is the fraction of the positive examples that were correctly labeled by the model as positive.
Recall = TP / (TP + FN)
Where TP is the true positives and FN is the false negatives.

Features
{'predictions': Value(dtype='int32', id=None), 'references': Value(dtype='int32', id=None)}
```

2. Ejercicio Métricas de Clasificación en Pipeline

```
import torch
from transformers import pipeline, AutoTokenizer, AutoModelForSequenceClassification
from evaluate import load

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_name = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)
model = model.to(device)
classifier = pipeline("text-classification", model=model, tokenizer=tokenizer, device=0)

new_data = ["this movie was terrible", "best movie ever"]

predictions = classifier(new_data)
predicted_labels = [1 if pred["label"]=="POSITIVE" else 0 for pred in predictions]
print(predicted_labels)
```

```
# tokenizar dato de entrada
new_input = tokenizer(new_data, return_tensors="pt", padding=True, truncation=True, max_length=64)
new_input = new_input.to(device)
with torch.no_grad():
    outputs = model(**new_input)

predicted = torch.argmax(outputs.logits, dim=1).tolist()
print(predicted)
# etiquetas ground truth

real = [0,1]

accuracy = load("accuracy")
precision = load("precision")
recall = load("recall")
f1 = load("f1")

acc_val = accuracy.compute(references=real, predictions=predicted)
precision_val = precision.compute(references=real, predictions=predicted)
recall_val = recall.compute(references=real, predictions=predicted)
f1_val = f1.compute(references=real, predictions=predicted)

print(f"acc: {acc_val}")
print(f"precision: {precision_val}")
print(f"recall: {recall_val}")
print(f"f1: {f1_val}")
```

```
Device set to use cuda:0
[0, 1]
[0, 1]
acc: {'accuracy': 1.0}
precision: {'precision': 1.0}
recall: {'recall': 1.0}
f1: {'f1': 1.0}
```

3. Perplexity

```
import torch
from evaluate import load
from transformers import AutoModelForCausalLM, AutoTokenizer

# revisando si la GPU esta disponible
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model_name = "gpt2"
model = AutoModelForCausalLM.from_pretrained(model_name).to(device)

tokenizer = AutoTokenizer.from_pretrained(model_name)

# configurando el padding token a eos_token
tokenizer.pad_token = tokenizer.eos_token
# Preparar el texto semilla
prompt = "Latest research findings in Antartica show"
input_ids = tokenizer.encode(prompt, return_tensors="pt").to(device)
attention_mask = torch.ones(input_ids.shape, device=device)
# Generacion de texto
output = model.generate(input_ids,
```

```
                        max_length=45,
                        num_return_sequences=1)
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
print(generated_text)

# Probando el Perplexity Score
# se requiere tokenizar el texto generado
inputs = tokenizer(generated_text,
                   return_tensors="pt",
                   padding=True,
                   truncation=True).to(device)
inputs['attention_mask'] = torch.ones(inputs['input_ids'].shape, device=device)
# cargando el perplexity score
perplexity = load("perplexity", module_type="metric")

# results = perplexity.compute(predictions=generated_text, model_id="gpt2")
results = perplexity.compute(model=model,
                             input_ids=inputs['input_ids'],
                             attention_mask=inputs['attention_mask'],
                             pad_token_id=tokenizer.pad_token_id)
print(results)
print(results["mean_perplexity"])
```

```
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected be
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
The attention mask is not set and cannot be inferred from input because pad token is same as eos toke
Latest research findings in Antartica show that the presence of a single molecule in the brain is ass
```

4. Ejercicio para BLEU

```
from pprint import pprint
from evaluate import load
bleu = load("bleu")
pprint(bleu.description)

input_text = "Latest research findings in Antarctica show"
references = [["Latest research findings in Antartica show significant ice loss due to climate change

generated_text = "Latest research findings in Antarctica show that the ice sheet is melting faster th
result = bleu.compute(predictions=[generated_text], references=references)
pprint(result)
```

```
('BLEU (Bilingual Evaluation Understudy) is an algorithm for evaluating the '
 'quality of text which has been machine-translated from one natural language '
 'to another.\n'
 "Quality is considered to be the correspondence between a machine's output "
 'and that of a human: "the closer a machine translation is to a professional '
 'human translation, the better it is"\n'
 '- this is the central idea behind BLEU. BLEU was one of the first metrics to '
 'claim a high correlation with human judgements of quality, and remains one '
 'of the most popular automated and inexpensive metrics.\n'
 '\n'
 'Scores are calculated for individual translated segments-generally '
 'sentences-by comparing them with a set of good quality reference '
 'translations.\n'
```

```
 'Those scores are then averaged over the whole corpus to reach an estimate of '
 "the translation's overall quality.\n"
 'Neither intelligibility nor grammatical correctness are not taken into '
 'account.\n')
{'bleu': 1.0,
 'brevity_penalty': 1.0,
 'length_ratio': 1.2142857142857142,
 'precisions': [1.0, 1.0, 1.0, 1.0],
 'reference_length': 14,
 'translation_length': 17}
```

5. Ejercicio BLEU Traducción

```
from pprint import pprint
from evaluate import load
bleu = load("bleu")

input_sentences = ["Hola, ¿cómo estás?", "Estoy genial, gracias"]
references = [["Hello, how are you?", "Hi, how are you?"],
              ["I'm great, thanks", "I'm great, thank you"]]
result = bleu.compute(predictions=input_sentences, references=references)
pprint(result)
pprint(bleu.description)
```

```
{'bleu': 0.0,
 'brevity_penalty': 0.8948393168143697,
 'length_ratio': 0.9,
 'precisions': [0.3333333333333333, 0.0, 0.0, 0.0],
 'reference_length': 10,
 'translation_length': 9}
('BLEU (Bilingual Evaluation Understudy) is an algorithm for evaluating the '
 'quality of text which has been machine-translated from one natural language '
 'to another.\n'
 "Quality is considered to be the correspondence between a machine's output "
 'and that of a human: "the closer a machine translation is to a professional '
 'human translation, the better it is"\n'
 '- this is the central idea behind BLEU. BLEU was one of the first metrics to '
 'claim a high correlation with human judgements of quality, and remains one '
 'of the most popular automated and inexpensive metrics.\n'
 '\n'
 'Scores are calculated for individual translated segments-generally '
 'sentences-by comparing them with a set of good quality reference '
 'translations.\n'
 'Those scores are then averaged over the whole corpus to reach an estimate of '
 "the translation's overall quality.\n"
 'Neither intelligibility nor grammatical correctness are not taken into '
 'account.\n')
```

brevity penalty Es un factor que se aplica en la puntuación BLEU para penalizar las traducciones que son más cortas que las oraciones de referencia. Si la traducción es significativamente más corta que la referencia, la puntuación BLEU se reduce. Esto ayuda a evitar que las traducciones excesivamente breves obtengan puntuaciones artificialmente altas.

length ratio Es la relación entre la longitud de la traducción y la longitud de la referencia. Se calcula dividiendo la longitud (el número de palabras) de la traducción por la longitud de la referencia. Un valor cercano a 1 indica que la longitud de la traducción y la referencia es similar, mientras que valores muy altos o muy bajos indican una diferencia significativa en la longitud.

translation length Es el número de palabras en la oración traducida. Es uno de los factores que se utilizan para calcular la relación de longitud y la penalización por brevedad.

reference length número de palabras en la oración de referencia

precisions proporción de n-gramas que aparecen en la referencia hasta 4 n-gramas.

6. ROUGE

```
from evaluate import load

rouge = load('rouge')

predictions = ["The cat sat on the mat."]
references = ["The cat is sitting on the mat."]

results = rouge.compute(predictions=predictions,
                        references=references)
pprint(results)
```

```
{'rouge1': 0.7692307692307692,
 'rouge2': 0.5454545454545454,
 'rougeL': 0.7692307692307692,
 'rougeLsum': 0.7692307692307692}
```

```
predictions = ["As we learn more about the frequency and size distribution of exoplanets, we are disc
references = ["The more we learn about the frequency and size distribution of exoplanets, the more co

results = rouge.compute(predictions=predictions,
                        references=references)
pprint(results)
```

```
{'rouge1': 0.7906976744186046,
 'rouge2': 0.5365853658536585,
 'rougeL': 0.7441860465116279,
 'rougeLsum': 0.7441860465116279}
```

```
references = ["""Un autómata finito (AF) o máquina de estado finito es un modelo computacional
que realiza cómputos en forma automática sobre una entrada para producir una salida."""]
predictions = ["""Un autómata finito (AF) es un modelo computacional que procesa entradas
de manera automática para generar una salida."""]
results = rouge.compute(predictions=predictions, references=references)
pprint(results)
```

```
{'rouge1': 0.64, 'rouge2': 0.4166666666666667, 'rougeL': 0.6, 'rougeLsum': 0.64}
```

Se observa que `rouge1` y `rougeL` dan resultados similares en los ejemplos. `rouge2`, por su parte, parece más bajo que las otras métricas. Esto se debe a la comparación a nivel de n-gramas realizada por la métrica.

7. METEOR y BLEU

```
from evaluate import load

bleu_metric = load('bleu')
meteor_metric = load('meteor')
```

```
predictions = ["He thought it right and necessary to become a knight-errant, roaming the world in arr
references = ["He believed it was proper and essential to transform into a knight-errant, traveling t

results_bleu = bleu_metric.compute(predictions=predictions, references=references)
results_meteor = meteor_metric.compute(predictions=predictions, references=references)
pprint(f"BLEU: {results_bleu}")
pprint(f"Meteor: {results_meteor}")
```

```
[nltk_data] Downloading package wordnet to /home/leningfe/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]     /home/leningfe/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /home/leningfe/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
("BLEU: {'bleu': 0.25928256340208583, 'precisions': [0.7, 0.3684210526315789, "
 "0.2222222222222222, 0.11764705882352941], 'brevity_penalty': "
 "0.9048374180359595, 'length_ratio': 0.9090909090909091, "
 "'translation_length': 20, 'reference_length': 22}")
"Meteor: {'meteor': 0.6531090723751274}"
```

Se observa que los scores de BLEU son más bajos comparados con METEOR

8. Exact Match

```
from evaluate import load
exact_match = load("exact_match")

predictions = ["The cat sat on the mat.",
               "Theaters are great.",
               "Like comparing oranges and apples."]
references = ["The cat sat on the mat?",
              "Theaters are great.",
              "Like comparing apples and oranges."]

results = exact_match.compute(predictions=predictions,
                              references=references)

pprint(f"Exact Match: {results}")
```

```
"Exact Match: {'exact_match': 0.3333333333333333}"
```

Se observa que la métrica busca establecer relaciones de identidad entre las oraciones generadas y las de referencia. Por ejemplo, si en la oración del medio, que es la única idéntica en ambos sets, alteramos y eliminamos el punto final, el `EM` se hace 0

9. Similitud de texto

9.1. Bert Score

```
from evaluate import load
bertscore = load("bertscore")
predictions = ["The burrow stretched forward like a narrow corridor for a while, then plunged abruptl
references = ["The rabbit-hole went straight on like a tunnel for some way, and then dipped suddenly

results = bertscore.compute(predictions=predictions,
```

```
                             references=references,
                             model_type="roberta-large")
pprint(f"Bert-Score: {results}")

# para meteor
meteor_score = load("meteor")
results_meteor = meteor_score.compute(predictions=predictions,
                                       references=references)
pprint(f"Meteor-Score: {results_meteor}")
```

```
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and i
("Bert-Score: {'precision': [0.9340652227401733], 'recall': "
 "[0.9245126247406006], 'f1': [0.9292643666267395], 'hashcode': "
 "'roberta-large_L17_no-idf_version=0.3.12(hug_trans=4.48.0)'}")
[nltk_data] Downloading package wordnet to /home/leningfe/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]     /home/leningfe/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /home/leningfe/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
"Meteor-Score: {'meteor': 0.37180012567275916}"
```

9.2. Similitud de Textos con Glove y Sim-score

```
import torch.nn.functional as F
from torchtext.vocab import GloVe
glove = GloVe(name='6B', dim=100)

sentence1 = "The cat is on the mat"
sentence2 = "The dog is on the mat"


# Function to get sentence embedding by averaging word embeddings
def get_sentence_embedding(sentence, glove):
    sentence_words = sentence.lower().split()
    word_embeddings = [glove[word] for word in sentence_words if word in glove.stoi]
    if word_embeddings:
        return torch.mean(torch.stack(word_embeddings), dim=0)
    else:
        return torch.zeros(glove.dim)  # Return zero vector if no embeddings are found

# Get the sentence embeddings for both sentences
embedding_sentence1 = get_sentence_embedding(words_sentence1, glove)
embedding_sentence2 = get_sentence_embedding(words_sentence2, glove)

cosine_similarity = F.cosine_similarity(embedding_sentence1,
                                        embedding_sentence2, dim=0)

pprint(f"Cosine similarity between the sentences: {cosine_similarity.item():.4f}")
```

Cosine similarity between the sentences: 0.9948