Detección de entidades en informes clínicos mediante el uso de LLMs

Gonzalo Sanchez Montesinos
Daniel Bordeianu

Noviembre 2024

Resumen

Este documento es la memoria del trabajo realizado por Daniel Bordeianu y Gonzalo Sánchez para dar respuesta al problema **Detección de entidades en informes clínicos mediante el uso de LLMs** propuesto en el **Concurso de Modelización de Problemas de Empresas**.
El reconocimiento de entidades nombradas (NER) es fundamental dentro del procesamiento del lenguaje natural. En este sentido, los modelos de lenguaje de gran escala (LLMs) permiten, mediante técnicas de **prompt engineering**, realizar tareas de reconocimiento de entidades nombradas, sin necesidad de entrenamientos adicionales.

Este documento pretende recopilar la fundamentación teórico-matemática del problema, una guía de código que permita comprender el código que hemos desarrollado para resolverlo, así como un análisis de los resultados obtenidos.

El capítulo 1 recoge el fundamento matemático, el 2 cubre la generación de informes clínicos utilizando LLMs, el 3 el reconocimiento de entidades nombradas, el 4 la obtención de métricas y el 5 las conclusiones obtenidas. Finalmente, se incluye la bibliografía.

1. Fundamentos Matemáticos del Procesamiento del Lenguaje Natural y Estado del Arte

1.1. Fundamentos Matemáticos del Procesamiento del Lenguaje Natural

Pese a que existan múltiples técnicas para realizar un análisis del lenguaje natural, se pueden clasificar en dos grandes bloques: las técnicas lingüísticas formales y las técnicas empiricistas o probabilistas.

Un modelo probabilístico de lenguaje define una distribución de probabilidad a partir de los valores observados en un corpus de documentos. Recordando algunos conceptos de Probabilidad, se tiene que, la probabilidad de que un elemento (por ejemplo, una palabra) p_i aparezca en un corpus, donde ya han aparecido una serie de palabras antes, depende de todas las anteriores:

$$\textbf{Regla de la Cadena: } P(p_1 \cap p_2 \cap ... \cap p_n) = P(p_1)P(p_2|p_1) \cdot ... \cdot P(p_n|p_1 \cap ... \cap p_{n-1})$$

Despejando:

$$P(p_n|p_1 \cap ... \cap p_{n-1}) = \frac{P(p_1 \cap p_2 \cap ... \cap p_n)}{P(p_1)P(p_2|p_1) \cdot ... \cdot P(p_{n-1}|p_1 \cap ... \cap p_{n-2})}$$

Puesto que puede haber combinaciones de palabras que no estén presentes en el corpus, esta hipótesis es impracticable. Por lo tanto, se asume que la probabilidad de un elemento depende solo de los n-1 elementos anteriores:

$$\textbf{Hipótesis de Markov: } P(p_i|p_1 \cap ... \cap p_{i-1}) = P(p_{i-(n-1)} \cap ... \cap p_{i-1})$$

El modelo probabilístico de los n-gramas utiliza la hipótesis de Markov para indicar que la dependencia es con los n-1 elementos anteriores.

No obstante, si una combinación de elementos no está en el corpus, la probabilidad que le asignará el n-grama será cero. Para solucionar este problema, se utilizan técnicas de alisado.

El alisado de Laplace, siendo AB un bigrama, asignaría como valor a la probabilidad condicional $P(B|A)$:

$$P(B|A) = \frac{frec(A,B) + t}{frec(A) + t \cdot m}$$

donde $frec$ denota la frecuencia de aparición en el corpus, t es el número de observaciones virtuales adicionales, y n el número de monogramas (palabras, en nuestro ejemplo) existentes en el corpus.
Es importante observar que las probabilidades siguen sumando 1.
Obviamente, la estimación óptima por máxima verosimilitud de la probabilidad condicionada $P(p_n|p_{n-1})$ es:

$$\frac{frec(w_{n-1}, w_n)}{frec(w_{n-1}}$$

Otra aproximación es la de los modelos "back-off", que estiman la probabilidad condicionada de una palabra fijándose no solo en las n-1 anteriores; sino que, en caso de ser esta probabilidad cero o menor que un determinado umbral, considera la probabilidad de la palabra condicionada a las n-2 anteriores; y así sucesivamente.
Un ejemplo de esto es el alisado por interpolación lineal. Por simplicidad, se asumen bigramas:

$$P_{Int}(w_2|w_1) = \alpha P(w_2|w_1) + (1 - \alpha)P(w_2)$$

donde $\alpha \in \mathbb{R}$ regula el peso que se le da a la probabilidad condicional y a la probabilidad incondicional.

Si se considerasen trigramas:

$$P_{Int}(w_i|w_{i-1}, w_{i-2}) = \lambda_3 P(w_i|w_{i-1}, w_{i-2}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_1 P(w_i)$$

donde $\lambda_1, \lambda_2, \lambda_3 \geq 0$ y $\lambda_1 + \lambda_2 + \lambda_3 = 1$
Los valores de los parámetros pueden estar prefijados, o pueden hallarse con algoritmos para maximizar la verosimilitud. También se puede hacer que dependan del contexto, si existen muchos trigramas con la palabra w_i, entonces se le podría dar a λ_3 un valor alto, en otro caso, se le podría dar un valor bajo.

1.2. Estado del arte

En primer lugar, los tipos más importantes de Machine Learning (ML) son el supervisado, no supervisado, semi-supervisado y por refuerzo. Los modelos de ML solo pueden aprender con los datos con los que se los entrena. No obstante, se ha observado en los LLMs más importantes un nuevo tipo de ML: **in-context learning**. Consiste en añadir unos pocos ejemplos de "entrenamiento" en el prompt para que el LLM "aprenda" a realizar una nueva tarea, que se "olvida" en cuanto el LLM manda su respuesta (esto es, los pesos del modelo no se modifican).

En función del número de ejemplos que se indiquen en el prompt, se consideran: **zero-shot learning** (no se proporcionan ejemplos en el prompt), **one-shot learning** (solo se proporciona un ejemplo), **few-shot learning** (se proporcionan varios ejemplos para que mejore la respuesta del LLM).

Como se muestra en [3], los modelos suficientemente grandes (entreandos con una cantidad y diversidad suficiente de datos) son capaces de realizar nuevas tareas gracias al ICL.

Lo anterior justifica que el enfoque desde el que se aborde el problema sea este mismo. Se emplean técnicas de prompt engineering para, utilizando LLMs, reconocer entidades nombradas en textos médicos.

Las técnicas utilizadas están basadas en el empleo del paquete spacy-llm, que integra dentro de los *pipelines* de spaCy distintos LLMs, entre los que se encuentran distintas versiones de ChatGPT. El paquete tiene definidas distintas **tareas**, que definen problmeas de NLP que se envían al LLM mediante prompts. Las tareas también permiten dar una estructura a la respuesta del LLM.

Algunas de las tareas que ofrece este paquete son: *spacy.EntityLinker.v1, spacy.Summarization.v1, spacy.NER.v3.*

La tarea en la que nos interesamos en este trabajo es spacy.NER.v3. Esta es la tarea que servirá de base para el desarrollo del capítulo 3.

La mencionada tarea sirve para introducir una técnica frecuente de prompt engineering, **Chain-of-Thought Prompting**, que consiste en ofrecer al LLM ejemplos que no incluyan solamente la respuesta, sino también cómo llegar hasta ella. Un ejemplo que se puede consultar en [5] es el siguiente:
Pregunta de Ejemplo: Roger tiene 5 pelotas de Tenis. Compra 2 latas más de pelotas de tenis. Cada lata tiene 3 pelotas de tenis. ¿Cuántas pelotas de tenis tiene ahora?
Respuesta de Ejemplo: Roger empezó con 5 pelotas. 2 latas de 3 pelotas cada una son 6 pelotas de tenis. 6 + 5 = 11. La respuesta es 11
Aquí iría una pregunta con una estructura similar a la anterior que el LLM deba responder. Lo hará

siguiendo el formato de la respuesta anterior, lo que hará que sea menos probable que se equivoque.

Otras técnicas de prompt engineering se pueden consultar en [6], no se detallan aquí por no estar directamente relacionadas con la construcción de nuestra solución.

2. Generación de textos clínicos

Esta sección trata la generación de informes clínicos utilizando la API de OPENAI. Los informes clínicos generados se pueden consultar en la carpeta de GOOGLE DRIVE compartida, en REPORTS_GENERATED.
Por su parte, el código empleado para la generación de los informes clínicos, que se cubrirá en esta parte de la memoria, se puede encontrar en la carpeta CODE, en el cuaderno **reports_generation.ipynb**

El fragmento de código crucial es el siguiente, que, empleando la API de OpenAI y dándole contexto, pide a ChatGPT-4o que genere una serie de informes clínicos:

```
def generate_reports(language, country, required_entity, number_of_reports=5):
  client = OpenAI(api_key=userdata.get('apiKey'))

  system_content_text = f"You are a doctor from {country}. You are writing typical linical
  ↪  reports in {language}."

  user_content_text = f"Write {number_of_reports} clinical reports. Mark its endindg with the
  ↪  string \"----\". You must include: {required_entity}."
  completion =   client.chat.completions.create(
      model="gpt-4o",
      messages=[
            {
              "role": "system",
              "content": [
                {
                  "type": "text",
                  "text": system_content_text
                }
              ]
            },

          {"role": "user", "content": user_content_text}

      ]
  )
  return completion.choices[0].message.content
```

Los parámetros de lengua y país permiten generar informes en diversos idiomas. En los ejemplos que hemos generado y que están en Google Drive aparecen ejemplos en inglés, francés y, por supuesto, español.
Asimismo, el parámetro required entity permite especificar entidades que deban aparecer en el informe, como por ejemplo, un tipo de enferdad. Permite, además, incluir un ejemplo de estructura, para asegurarse de que siga una estructura concreta. En los ejemplos proporcionados en Google Drive, hay dos tipos de estructuras distintas: una más rígida (que se ajusta al formato: nombre, día, motivo de la visita...); y una mucho más laxa, habiéndole pedido al LLM que evitase estructuras del tipo "Síntomas: ...lista de síntomas".
Esta variedad de informes generados, responde a nuestro intento por obtener resultados significativos en las secciones posteriores.

Para aumentar la legibilidad, aunque los informes clínicos se almacenarán en un JSON que se utilizará en los apartados posteriores, se almacenan en un .docx. Se aconseja que sean los .docx los archivos que se consulten en Google Drive, pues los JSON no son especialmente legibles, mucho menos las partes en francés o español, debido a la utilización de tildes. El código para pasar a .docx es el siguiente:

```
from docx import Document
def convert2docxOneReport(report_text, doc):
  paragraphs = report_text.split("\n")

  for paragraph in paragraphs:
      bold_text_split = paragraph.split("**")
      paragraph_adder = doc.add_paragraph()
      for i, part in enumerate(bold_text_split):
          if i % 2 == 0:
              # Regular text
              paragraph_adder.add_run(part)
          else:
              # Bold text
              bold_run = paragraph_adder.add_run(part)
              bold_run.bold = True
  doc.add_page_break()


def convert2docx(list_of_reports):
  #Creating the document
  doc = Document()
  for i,report in enumerate(list_of_reports):
    doc.add_heading(f"Report{i}", level=1)
    convert2docxOneReport(report, doc)
  doc.save("Reports.docx")

```

Importante: Para ejecutar el código del cuaderno deben utilizar su propia API KEY para la API de OpenAI.

3. Implementación de técnicas de prompt engineering para NER

Esta sección se basa en una de las referencias que se proporcionaba en el enunciado, concretamente en [7].
El código que obtiene las entidades de los informes generados en la sección anterior se puede encontrar en la carpeta CODE, en el cuaderno **ner_spacy.ipynb**. Asimismo, los resultados de hacer NER sobre los informes generados en la sección anterior se pueden consultar en la carpeta **NER_GPT_3.5**.

Las partes clave del código de esta sección son la función que obtiene las entidades de un informe clínico concreto:

```
def get_entities(nlp, report):
    doc = nlp(report)
    entities = {}
    for ent in doc.ents:
        if ent.label_ not in entities:
            entities[ent.label_] = []
            entities[ent.label_].append(ent.text)
        else:
            entities[ent.label_].append(ent.text)
    return entities
```

así como la que crea el nlp a partir del fichero de configuración:

```
def create_nlp():
  nlp = assemble("config.cfg")
  return nlp
```

Este fichero es, si se quiere utilizar chatGPT 3.5, de la forma:

```
[nlp]
lang = "en"
pipeline = ["llm"]

[components]

[components.llm]
factory = "llm"

[components.llm.task]
@llm_tasks = "spacy.NER.v3"
labels = ["Disease", "Diagnosis", "Medication and dosing", "Symptom", "Procedure", "Names of
↪  Patiens", "Patient's Sex", "Patient's Age", "Patient's Weight", "Patient's Height",
↪  "Admission date", "Medical discharge date", "Tests"]

[components.llm.model]
```

```
@llm_models  = "spacy.GPT-3-5.v1"
config = {"temperature": 0.0}
```

Nota: se ha probado con distintas temperaturas: 0.0, 0.3
Si se usa GPT-4:

```
[nlp]
lang = "en"
pipeline = ["llm"]

[components]

[components.llm]
factory = "llm"

[components.llm.task]
@llm_tasks = "spacy.NER.v3"
labels = ["Disease", "Diagnosis", "Medication and dosing", "Symptom", "Procedure", "Names of Patiens", "Patient's Sex", "Patient's Age", "Patient's Weight", "Patient's Height", "Admission date", "Medical discharge date", "Tests"]

[components.llm.model]
@llm_models  = "spacy.GPT-4.v2"
config = {"temperature": 0.0}
```

Importante: Para ejecutar el código del cuaderno deben utilizar su propia API KEY para la API de OpenAI.

Se incluye en el cuaderno código que toma automáticamente los informes clínicos sobre los que hacer NER de nuestro repositorio de GitHub:

```
import requests
def download_file(url,local_filename):
  response = requests.get(url)
  with open(local_filename, 'wb') as file:
      file.write(response.content)

#descargar fichero config.cfg

url = "https://raw.githubusercontent.com/gonsan33/Concurso-ModelizaciOn-Empresas-24/main/GMV/CONFIG_FILES/config_GPT4.cfg"
local_filename = "content/config.cfg"
download_file(url,local_filename)
url = "https://raw.githubusercontent.com/gonsan33/Concurso-ModelizaciOn-Empresas-24/main/GMV/REPORTS_GENERATED/medical_reports.json"
local_filename = "content/medical_reports.json"
download_file(url,local_filename)
```

4. Validación de los resultados: obtención automatizada de métricas

El código para la obtención automatizada de métricas se puede encontrar en la carpeta CODE, en el cuaderno **metrics_evaluation_ner.ipynb**.
Este es una modificación del cuaderno **ner_spacy.ipynb**.Primero, se obtienen las entidades con nombre de una lista de informes clínicos y luego se utiliza la librería **nervaluate** para comparar con "la verdad", que se le proporciona al sistema
A este respecto, la función más importante es:

Obtiene las métricas. Estas se almacenan posteriormente en un JSON, que se puede consultar en la carpeta TEST_REPORTS_AND_METRICS

```
from nervaluate import Evaluator
def get_metrics(predicted_labels, ground_truth_labels, nlp):
  evaluator = Evaluator(ground_truth_labels, predicted_labels, tags =
  ↪  list(nlp.get_pipe("llm").labels) , loader="default")

  # call the evaluator to get model evaluation result
  return (evaluator.evaluate())
```

En el cuaderno se incluyen funciones que pasan el formato en el que el usuario da las etiquetas verdaderas al formato que utiliza la librería. Este se puede consultar en [8].

La librería mencionada evalúa según distintos métodos: Strict Evaluation Method, Exact Evaluation Method, Partial Evaluation Method. La evaluación estricta considera que una predicción del modelo es correcta si y solo si la etiqueta y el string predichos por el modelo coinciden exactamente (límite superior e inferior del string) con la verdad. La evaluación exacta busca solamente que los dos strings coincidan, aunque las etiquetas no lo hagan. Por el contrario, la evaluación parcial tiene en cuenta la posibilidad de que los strings no sean exactamente iguales, sino que se solapen.

Se elige este último método a la hora de evaluar los resultados obtenidos. Este método es similar al método IOB (inside, outside, begin), que se expone en [9], y que da crédito al modelo cuando obtiene un *partial match*.

A continuación se incluyen tablas de los resultados obtenidos:

GLOBAL			
MODEL	F1	RECALL	PRECISION
GPT-4 temp = 0,0	0,79	0,908	0,699
GPT-4 temp = 0,3	0,84	0,966	0,743
GPT-3,5 temp = 0,	0,786	0,868	0,719
GPT-3,5 temp = 0,	0,786	0,738	0,719

Figura 1: Métricas Globales

MEDICATION AND DOSING			
MODEL	F1	RECALL	PRECISION
GPT-4 temp = 0,0	0,727	1	0,571
GPT-4 temp = 0,3	0,545	0,75	0,429
GPT-3,5 temp = 0,	0,542	0,813	0,406
GPT-3,5 temp = 0,	0,542	0,813	0,406

Figura 2: Medication and Dosing

PATIENT'S AGE			
MODEL	F1	RECALL	PRECISION
GPT-4 temp = 0,0	1	1	1
GPT-4 temp = 0,3	1	1	1
GPT-3,5 temp = 0,	1	1	1
GPT-3,5 temp = 0,	1	1	1

Figura 3: Patient's Age

ADMISSION DATE			
MODEL	F1	RECALL	PRECISION
GPT-4 temp = 0,0	0,941	1	0,889
GPT-4 temp = 0,3	0,941	1	0,889
GPT-3,5 temp = 0,	0,941	1	0,889
GPT-3,5 temp = 0,	0,941	1	0,889

Figura 4: Admission Date

DISEASE			
MODEL	F1	RECALL	PRECISION
GPT-4 temp =	0	0	0
GPT-4 temp =	0	0	0
GPT-3,5 tem	0	0	0
GPT-3,5 tem	0	0	0

Figura 5: Disease

SYMPTOM			
MODEL	F1	RECALL	PRECISION
GPT-4 temp	0,704	0,731	0,679
GPT-4 temp	0,704	0,731	0,679
GPT-3,5 tem	0,64	0,615	0,667
GPT-3,5 tem	0,64	0,615	0,666

Figura 6: Symptom

PATIENT'S SEX			
MODEL	F1	RECALL	PRECISION
GPT-4 temp	1	1	1
GPT-4 temp	1	1	1
GPT-3,5 tem	1	1	1
GPT-3,5 tem	1	1	1

Figura 7: Sex

PATIENT'S WEIGHT			
MODEL	F1	RECALL	PRECISION
GPT-4 temp	1	1	1
GPT-4 temp	1	1	1
GPT-3,5 tem	1	1	1
GPT-3,5 tem	1	1	1

Figura 8: Weight

MEDICAL DISCHARGE DATE			
MODEL	F1	RECALL	PRECISION
GPT-4 temp	1	1	1
GPT-4 temp	1	1	1
GPT-3,5 tem	0,941	1	0,889
GPT-3,5 tem	0,889	1	0,8

Figura 9: Medical Discharge Date

PROCEDURE			
MODEL	F1	RECALL	PRECISION
GPT-4 temp	0,5	1	0,333
GPT-4 temp	0,462	1	0,3
GPT-3,5 tem	0,429	1	0,273
GPT-3,5 tem	0,429	1	0,273

Figura 10: Procedure

PATIENT'S AGE			
MODEL	F1	RECALL	PRECISION
GPT-4 temp	1	1	1
GPT-4 temp	1	1	1
GPT-3,5 tem	1	1	1
GPT-3,5 tem	1	1	1

Figura 11: Age

PATIENT'S HEIGHT			
MODEL	F1	RECALL	PRECISION
GPT-4 temp	1	1	1
GPT-4 temp	1	1	1
GPT-3,5 tem	1	1	1
GPT-3,5 tem	1	1	1

Figura 12: Height

TESTS			
MODEL	F1	RECALL	PRECISION
GPT-4 temp	0,722	0,813	0,65
GPT-4 temp	0,778	0,875	0,7
GPT-3,5 tem	0,588	0,625	0,556
GPT-3,5 tem	0,625	0,6,25	0,625

Figura 13: Tests

En cuanto al análisis de las métricas, se tiene que:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1Score = \frac{2 \cdot PrecisionRecall}{Precision + Recall}$$

Donde TP, FP y FN son verdaderos positivos, falsos positivos y falsos negativos, respectivamente.
La precisión indica el ratio de verdaderos positivos entre el total de positivos. Es decir, la relación entre las entidades que el modelo identifica como tales de forma correcta, y el total de entidades que el modelo identifica.
En general, las 4 configuraciones de LLMs utilizados tienen una precisón cercana al 0.7, un número considerable. Además, no se aprecian grandes diferencias entre los resultados de GPT 3.5 para las distintas temperaturas. Sí que se aprecian diferencias entre el rendimiento de GPT 3.5 y el de GPT4, considerablemente mejor.

Por su parte, el recall es el ratio entre el total de entidades que el modelo identifica, y el total de entidades que debería haber identificado (es decir, el total de entidades presentes en el texto).

En general, los valores de recall son muy altos para todas las configuraciones, lo que significa que el número de entidades presentes en el texto es del orden del número de entidades que identifica correctamente el modelo.

Por su parte, el F1 score indica el balance entre Precisión y Recall, en 3 de las 4 configuraciones está en torno al 50 %, mientras que en la otra (GPT-4 con temperatura 0.0) es cercana al 75 %.

En cuanto al análisis por entidad, es útil guiarse por la siguiente tabla:

Recall	Precisión	Interpretación	Entidades
Alto	Alto	Esta entidad es manejada bien por el modelo.	Symptom, Patient's Sex, Patient's Weight, Medical Discharge Date, Patient's Age, Admission Date, Patient's Age, Patient's Hight, Tests, (Diagnosis)
Bajo	Alto	El modelo no siempre puede extraer esta entidad, pero cuando lo hace es con alta confianza.	
Alto	Bajo	El modelo extrae esta entidad bien, sin embargo, es con baja confianza ya que a veces se extrae como otro tipo.	Procedure
Bajo	Bajo	Este tipo de entidad es manejado de manera deficiente por el modelo, ya que generalmente no se extrae. Cuando se extrae, no es con alta confianza.	Names of Patients, Disease

Cuadro 1: Interpretación del rendimiento del modelo por entidades según los valores de Recall y Precisión. Análisis basado en [10]

5. Conclusiones

En las métricas obtenidas en las secciones anteriores se aprecia cómo el modelo maneja bastante bien todas las entidades, excepto los nombres y las enfermedades, que no aparecen en suficiente cantidad en los datos.
Debido a la evaluación positiva del modelo en todos los demás campos, se opta por pedir dos nuevos informes, estos mucho más largos y menos estructurados, en los que volver a evaluar al modelo. Por lo superior que es, según los datos anteriores, GPT-4 a GPT-3.5, se opta directamente por utilizar el primero. Los resultados obtenidos se pueden consultar en el JSON long_metrics_GPT4_temp0.3, aunque los resultados empeoran de manera considerable. La precisión se reduce a 0.216, el recall a 0.727, y F1 a 0.333. Estos resultados mejoran si se añaden ejemplos al fichero de configuración.

6. Bibliografía

Referencias

[1] Stuart Russel, Peter Norvig, *Artificial Intelligence, a modern approach.*, Chapters 13,14,15,20,22.

[2] https://www.hopsworks.ai/dictionary/in-context-learning-icl

[3] Allan Raventós, Mansheej Paul, Feng Chen, Surya Ganguli *Pretraining task diversity and the emergence of non-Bayesian in-context learning for regression*

[4] Spacy llm Documentation: https://spacy.io/usage/large-language-models

[5] Chain of Thought: https://www.promptingguide.ai/techniques/cot

[6] Prompt Engineering Techniques: https://www.promptingguide.ai/techniques

[7] spacy-llm: Integrating LLMs into structured NLP pipelines: https://medium.com/@pankaj_pandey/spacy-llm-integrating-llms-into-structured-nlp-pipelines-7134dd05ebc2

[8] nervaluate — The Ultimate way for Benchmarking NER Models https://rumn.medium.com/nervaluate-the-ultimate-way-for-benchmarking-ner-models-b29e83fbae95

[9] Named Entity Recognition — Clinical Data Extraction https://vishal-aiml164.medium.com/named-entity-recognition-clinical-data-extraction-9b089d91b27b

[10] Evaluation metrics for custom named entity recognition models https://learn.microsoft.com/en-us/azure/ai-services/language-service/custom-named-entity-recognition/concepts/evaluation-metrics