Máster Universitario en IA – UTAMED (PN).
Solís López, Rafael

1. Introducción y objetivos

Como estudiante del Máster Universitario en IA ofertado en UTAMED y con el fin de participar un concurso patrocinado por el Ayuntamiento de Málaga denominado "**Málaga -Smart Urban System**", consistente en la elaboración de ideas para favorecer la gestión de opiniones, quejas y comentarios de los diferentes ciudadanos de forma que puedan aportar valor, vamos a elaborar una **PoC** (Proof Of Concept) consistente en **extraer opiniones de diferentes fuentes de internet relacionadas con la ciudad, analizar posteriormente su polaridad y clasificarlas en un determinado número de temáticas (tópicos)** para poder realizar una clasificación de las mismas que puedan ser empleadas para poder efectuar consultas estratégicas.

En una **segunda fase**, la propuesta plantea **conectar el modelo de clasificación / análisis de sentimiento un modelo LLM** (Large Lenguage Model) para que actúe como **agente intermediario entre el modelo y los usuarios** de forma bidireccional[1], ofreciendo respuestas automáticas y aclarando consultas ciudadanas e incorporando las nuevas consultas al sistema ir ampliando el corpus y enriqueciendo el modelo planteado.

2. Estructura del proyecto.

Todo el análisis y procesamiento computacional ha sido efectuado a través de notebooks empleando Python y librerías específicas en el entorno colaborativo Google Colab. Se han elaborado 3 notebooks y un script, cada uno con un contenido específico el cual detallamos a continuación:

- **extraccion.ipynb** (extracción de la información de las diferente fuentes y generación de los ficheros csv que componen el corpus a procesar).
- **modelado.ipynb.** (procesamiento del corpus , generación del análisis de sentimientos y clasificación en tópicos, estudio gráfico y validación del modelo a través de métricas).

[1] *En la PoC el agente recibe un mensaje del usuario y llamada los modelos generados previamente para analizar la polaridad y el tópico. Para mejorar el flujo debería implementarse una base de datos vectorial que pudiese almacenar los nuevos mensajes para ir alimentando y reentrenando los modelos para mejorar su ajuste.*

- **chat_ia.ipynb** (implementación de un agente de IA que permita realizar consultas relacionadas con el proyecto haciendo uso del modelo generado en el paso anterior).
- **Prdproccesor.py.** Script de Python con una clase propia creada para realizar el preprocesamiento. Se ha generado en un archivo independiente para no duplicar código y poder realizar su llamada desde los otros notebooks.

Tanto el código (carpeta code) , como los ficheros csv que componen el corpus (carpeta data) y los modelos generados para analizar la polaridad, modelizar tópicos y vectorizar (carpeta models). han sido subidos a un respositorio en Github (https://github.com/rsolis-utamed/pln_practica/tree/main).

3. Datos y preprocesamiento.

3.1. Obtención de los datos.

Para la elaboración del **corpus** (conjunto de textos empleados para el estudio) nos hemos dirigido a diferentes fuentes en la web.

En primera instancia se buscaron quejas e incidencias en organismos oficiales (Consulta Quejas y Sugerencias Ayto Málaga), no obteniéndose resultados debido al carácter confidencial de dicha información.

En una segunda búsqueda se localizaron varios foros y webs que poseen información sobre comentarios, quejas e incidencias relativas a la ciudad de Málaga, dentro de este bloque de recursos seleccionamos 3 que consideramos que poseían información relevante y además permitían realizar llamadas programáticas para extraer la información.

Recurso	URL	Descripción
cotilleando	https://www.cotilleando.com/threads/malaga-da-asco.139225/",	Foro enfocado principalmente en temas de sociedad, crónica social y debates de opinión personal. El hilo específico titulado "Málaga da asco" es un espacio de queja y debate crítico.
minube	https://www.minube.com/rincon/malaga-a13124	Plataforma social de viajes y turismo donde los usuarios recomiendan lugares, restaurantes y experiencias.
forocoches	https://forocoches.com/foro/showthread.php?t=9441037	Foro de debate general. En este hilo específico los usuarios suelen debatir sobre la realidad socioeconómica de la ciudad.

Para efectuar la extracción de la información de los diferentes recursos se ha empleado la librería **BeautifulSoup**[2] que nos permite navegar a través de las etiquetas de un documento HTML para extraer y procesar información.

Una vez extraídos los comentarios de cada fuente se han procesado a través de la librería pandas que permite encapsular la información en **dataframes**[3] para posteriormente generar un csv por cada fuente.

3.2. Procesamiento del corpus.

El **preproceamiento** lingüístico es la etapa en la que se limpia, normaliza y transforma el texto. Para realizar este paso se ha generado una clase en Python en la que se recoge se recibe un texto y se le aplican una serie de transformaciones en una secuencia ordenada de pasos:

1. Paso a **minúsculas** del texto.
2. **Eliminación** de **etiquetas html**, **caracteres especiales y signos de puntuación**. Para ello se emplea una librería específica de Python denominada "re" (regular expressions).
3. **Tokenizado.** Es el proceso mediante el cual el texto es divido en unidades elementales de procesamiento, en nuestro caso cada comentario se divide en palabras, generándose un array con tokens por texto. Para realizar esta acción hemos hecho uso de la librería genérica para PLN de Python.
4. **Lemantización.** Proceso que consiste en reducir las palabras a su forma canónica correcta analizando el contexto gramatical (por ejemplo "viajó" pasaría a "viajar"). La librería spaCy facilita posee métodos específicos que permiten realizan esta acción para diferentes lenguas.
5. **Eliminación de stopwords**. En esta fase se intenta eliminar del corpus toda palabra que no aporta significado semántico relevante al análisis (por ejemplo: artículos y preposiciones). La librería **nltk** posee un módulo específico para este tratamiento.

---

[2] Para procesar las peticiones http se ha empleado además la librería request.
[3] La librería de Pandas permite generar y manipular dataframes.

Una vez obtenemos el texto preprocesado es necesario llevar a cabo una última acción antes de poder integrar la información a algún modelo de clasificación para que procese los datos y pueda aprender de la información recibida, esta sería la **vectorización** de los tokens, es decir, el paso de una matriz de string a una **matriz numérica** fácilmente procesable por una computadora.

En nuestro proyecto hemos implementado un tipo de vectorización denominado **TF-IDF** que nos permite generar matrices numéricas considerando la **frecuencia de una palabra** en un documento (**TF**) y asignando **mayor peso a palabras que aparecen en pocos documentos (IDF)**, ya que éstas contienen un valor diferenciador mayor.

4. Análisis de sentimientos

4.1. Creación del modelo.

Para detectar la polaridad latente en los textos hemos confeccionado un modelo de análisis de sentimientos bajo un entorno no supervisado, dado que no disponíamos de etiquetas previas. Para el etiquetado automático de los textos hemos empleados dos librerías:

- **TextBlob**. Librería construida sobre nltk que ofrece una interfaz sencilla de trabajo. El diccionario que posee integrado está en inglés por lo que su precisión para el etiquetado de textos en otros idiomas no arroja resultados precisos. Para mejorar el rendimiento del modelo se ha empleado la librería "**googletrans**" para pasar por argumento al método de predicción de polaridad el texto traducido a inglés.
- **PySentimiento**. Modelo entrenado específicamente en español. Utiliza un modelo basado en **Transformer (BERT)**. Dentro de los modelos posibles hemos selecionado el modelo preentrenado "**finiteautomata**" especializado en noticias, correos y reseñas.

A continuación, mostramos una tabla comparativa.

Característica	TextBlob	PySentimiento
Idioma Nativo	Inglés (requiere traducción)	Español (y otros)
Tecnología	Diccionario de reglas (Léxico)	Redes Neuronales (Transformers)
Precisión en Español	Baja (pierde matices)	Muy Alta (especializado)
Facilidad	Muy alta (una línea de código)	Alta (pero requiere instalar librería)
Detección de Negatividad	Deficiente en español sin traducir	Muy Alta (especializado)

Posteriormente hemos implementado un modelo de clasificación que integra un algoritmo **Naive Bayes** a través de la librería de Python **Scitkit-learn**. Este algoritmo calcula **la probabilidad de que un texto pertenezca a una clase dada basado en la experiencia previa** (corpus preprocesado y vectorizado).

4.2. Validación.

Los datos de partida se componen de 50 mensajes extraídos de 3 fuentes diferentes (MiNube, Cotilleando y ForoCoches) sin etiquetas previas. En una primera fase se ha generado un dataframe con los textos originales y etiquetas generadas por ambos modelos (TextBlob y PySentimiento).

Para entrenar el modelo se ha dividido el set de datos en 80% de la información para su entrenamiento y el 20% restante para la contrastación de las predicciones generadas por el modelo entrenado.

Ambos modelos arrojan un f1-score[4] de aproximadamente el 0.5, lo que nos indica que el **rendimiento de los modelos es mediocre en ambos casos**.

Por otro lado, dado que partimos de **etiquetas generadas sintéticamente**, en una fase posterior se ha generado un set de datos balanceado con 15 mensajes (5 neutros, 5 positivos y 5 negativos) y se les ha asignado manualmente la polaridad

4 $F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$ Medida empleada para estudiar el rendimiento del modelo en caso de clases desbalanceadas. Su rango de valores está comprendido entre 0: rendimiento nulo y 1: rendimiento perfecto.

para otorgarle etiquetas reales y se han generado métricas de rendimiento la cuales mostramos a continuación.

- **Rendimiento del modelo TextBlob:**

F1 Score: 0.16667

	precision	recall	f1-score	support
Negativo	0.00	0.00	0.00	5
Neutro	0.00	0.00	0.00	5
Positivo	0.33	1.00	0.50	5
accuracy			0.33	15
macro avg	0.11	0.33	0.17	15
weighted avg	0.11	0.33	0.17	15

El modelo posee un ajuste f1-score muy bajo (0.166667), con un sesgo hacia el lado positivo, poseyendo un recall de 1 para la categoría positivo y 0 en el resto.

- **Rendimiento del modelo PySentimiento:**

F1 Score: 0.29132

	precision	recall	f1-score	support
Negativo	0.50	0.20	0.29	5
Neutro	0.42	1.00	0.59	5
Positivo	0.00	0.00	0.00	5
accuracy			0.40	15
macro avg	0.31	0.40	0.29	15
weighted avg	0.31	0.40	0.29	15

El modelo muestra un desempleo deficiente (f1-score de 0.29132) aunque superior al otro modelo. Realizando un análisis por categorías:

  - **Negativo:** Tiene una precisión del 0.50, lo que significa que cuando el modelo dice que algo es negativo, solo acierta la mitad de las veces. Su recall (0.20) es muy bajo, indicando que se le escapan el 80% de los mensajes negativos reales.
  - **Neutro:** Tiene un recall de 1.00, lo que significa que identificó todos los mensajes neutros, pero a costa de clasificar casi todo como .
  - **Positivo:** El desempeño es nulo (0.00). El modelo no fue capaz de identificar ni un solo mensaje positivo correctamente.

5. Modelado de tópicos

5.1. Creación del modelo.

El modelado de tópicos es una tarea que nos permite descubrir temas subyacentes en grandes volúmenes de datos de forma automática. En nuestro caso particular hemos elegido un enfoque **LDA (Latent Dirichlet Allocation)** que se fundamente en un **modelo bayesiano que nos permite simular cómo podrían haber sido generados los textos**.

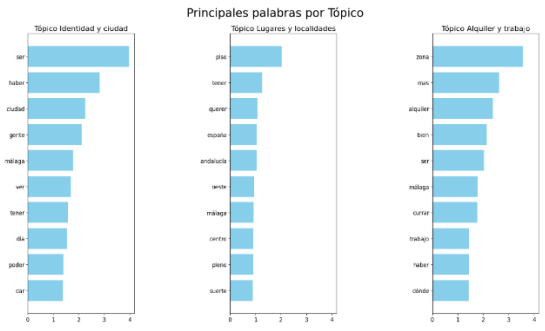
El proceso parte de las siguientes **premisas**:

- **Cada texto/documento está compuesto por una mezcla de tópicos.**
- **Cada tópico está definido por una distribución de palabras**.

5.2. Validación.

Para efectuar la validación del modelo hemos analizado tanto la métrica de **perplejidad** (mientas menor mejor entiende el modelo la estructura de tópicos) como la **coherencia** (interpretabilidad de los tópicos). Hemos probado diferentes **hiperparámetros** como son el **número de tópicos y el total de palabras a considerar en el diccionario**, finalmente se han seleccionado 3 tópicos y un diccionario de 1500 palabras.

Tras analizar el conjunto de palabras de cada tópico se le asignó un nombre representativo. Seguidamente, podemos ver en el gráfico el nombre asignado a cada tópico y las 10 palabras más representativas de cada uno.



test_df.xlsx

6. Creación de un flujo de diálogo con un LLM.

Una vez generado los modelos predictores de polaridad, vectorizador y asignador de tópicos se ha diseñado un notebook "chat_ia" en el que se definen una serie de funciones que hacen uso de los modelos compilados en pasos previos y son facilitados a un modelo **LLM**, en nuestro caso el modelo "**Gemini 2.5. Flash**". Para facilitar el flujo de comunicación entre las funcionalidades implementadas, el asistente de chat y el usuario final se ha construido un **agente** haciendo uso de la librería "**LangChain**".

A continuación, detallamos los principales elementos estructurales del código:

- **Funciones definidas** para llamar a los modelos compilados (preprocessed_vectorizer, det_sentiment, et_topic)
- **PromtTemplate.** Plantilla donde definimos el **rol del sistema y las instrucciones para llevar a cabo** la clasificación de polaridad y tópicos haciendo uso de las funciones específicas.
- **Tools**. Array con el nombre de todas las funciones mapeadas que puede emplear el agente para lograr los objetivos.
- **Llm**. Modelo **LLM empleado** para la conversación.
- **Variable que recoge la el mensaje escrito por el usuario.**
- **Invocación del agente para que genere los resultados**.

7. Conclusiones

Una vez definido el proceso de extracción de la información, generación del corpus, preprocesado y vectorización, diseño de los modelos de análisis de polaridad y modelado de tópicos, y atendiendo a las métricas obtenidas y la visualización de los resultados, podemos plasmar una serie de consideraciones.

- **Corpus**. Tanto el **número de textos** obtenidos (50) como la **representatividad y calidad** de los mismos **no es suficiente** para generar modelos de alto rendimiento.
- **Análisis de sentimiento**. Hemos implementado dos modelos, obteniendo un f1-score inferior a 0.5, por lo que los **resultado generados son muy deficientes actualmente para llevarlos a producción**.

- **Modelado de tópcios**. Al igual que sucede con el análisis de polaridad, el **rendimiento del modelo no es adecuado**. No se ha logrado reducir considerablemente la perplejidad ajustando el número de tópicos y el total de palabras a considerar. Además, la coherencia es baja, existen palabras relevantes que se repiten en varios tópicos.
- **Modelo LLM**. El agente integrado, empleando técnicas de prompting y llamada a los modelos posee un grado de calidad **muy condicionado al de los modelos diseñados**.

Podemos concluir que la PoC puede ser un buen punto de partida para elaborar un proyecto escalable de mayor rigor y precisión.

8. Próximos pasos

En función de las conclusiones expuestas podemos definir una serie de próximos pasos naturales que sería necesario definir:

- Aumento y mejora de la calidad del corpus entrenado para entrenar los modelos.
- Exploración de un mayor número de alternativas en cuanto a modelos y algoritmos a implementar para analizar la polaridad.
- Estudio en mayor profundidad de los posibles tópicos que podría presentar el modelo, atendiendo a la coherencia, representatividad de sus palabras y su capacidad clasificatoria.
- Integración de un sistema que permita al agente ia implementado almacenar las consultas lanzadas por los usuarios para ir enriqueciendo y evolucionando el modelo. Para ello deben revisarse las pipelines y plantear sistemas de almacenamiento (bases de datos vectoriales).
- Planteamiento de futuro escalado y metodología de desarrollo y mantenimiento del proceso.
- Consideración de la posible manipulación de información sensible que pueda ser sujeta a GDPR.

9. Bibligrafía

- Bird, S., Klein, E., y Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media.
- Apuntes de clase.