UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Clasificación multi-etiqueta de películas por género

TRABAJO LINGÜISTICA CONPUTACIONAL

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Autores: Andrés Pacheco, Miquel Gómez

Resumen

En el Procesamiento del Lenguaje Natural (PLN), existen distintas tareas que pasan por la compresión de textos para la posterior clasificación de estos. En concreto, en este trabajo investigamos distintas formas de clasificar por género películas en base a su descripción.

Como premisa, partimos de una serie de películas bien catalogadas, y nuestro trabajo consiste en catalogar un conjunto de test en los géneros correspondientes. La especialidad que tiene esta tarea es el hecho de que, una misma película, puede pertenecer a más de un género.

Para atacar este problema, se han hecho diversos experimentos, y se ha hecho uso de modelos desde *machine learning* clásico, pasando por *transformers* y llegando hasta modelos que rompen el estado del arte como *LLMs* de grandes capacidades.

Se muestran ejemplos y especificaciones de la implementación durante el paper, así como el coste y rendimiento de los distintos modelos. Se hace una reflexión del porqué ciertos modelos funcionan mejor que otros, y en qué situaciones es más conveniente su uso.

Palabras clave: Clasificación de textos; Películas; Transformers; LLMs; Machine Learning

---

Índice general

Índice general IV

1 Análisis del problema 1
- 1.1 Tecnologías . . . 1
  - 1.1.1 Modelos clásicos . . . 1
  - 1.1.2 Transformers . . . 2
  - 1.1.3 LLMs (Grandes Modelos de lenguaje) . . . 2
- 1.2 Dataset . . . 3
- 1.3 Manejo del dataset . . . 3
  - 1.3.1 Procesado del dataset . . . 3
- 1.4 Evaluación . . . 4

2 Implementación 5
- 2.1 Modelos clásicos . . . 5
- 2.2 Transformers . . . 6
- 2.3 LLMs . . . 6

3 Coste de las soluciones 9

4 Resultado de las soluciones 11
- 4.1 Clásicos . . . 11
- 4.2 Transformers . . . 12
- 4.3 LLMs . . . 12

5 Conclusiones 15

CAPÍTULO 1
Análisis del problema

Este proyecto ataca un problema de clasificación multi-etiqueta (que no multi-clase), en el que se pretenden asignar a un conjunto de películas un conjunto de géneros (etiquetas). Esto implica que la clasificación se hará de forma que, para cada película, se asigne UNO o MÁS géneros, acorde con lo que el nombre y la descripción de esta película indique.

Hay que recalcar la complicación de este problema, ya que, a pesar de que se puede abordar con clasificadores corrientes, hay que adaptarlos con el fin de que den más de una salida. En la sección de implementación se plantea cómo se ha gestionado este problema. Ahora, se analizan las tecnologías empleadas, el dataset usado y su evaluación.

1.1 Tecnologías

A continuación se describen brevemente las tres categorías de modelos evaluadas: modelos clásicos, transformers y LLMs. Para cada una, se resumen sus características, ventajas y limitaciones más relevantes para la tarea de clasificación multi-etiqueta.

1.1.1. Modelos clásicos

Los modelos clásicos que se usarán para el desarrollo de esta tarea son los siguientes: Regresión Logística, Random Forest, XGBoost y SVC.

Estos modelos, al igual que la gran mayoría de técnicas tradicionales, operan sobre vectores de características. En nuestro caso, estos serán extraídos de las descripciones de las películas. Para pasar de texto a vectores, se requiere un preprocesado explícito, como eliminar palabras repetitivas o reducir palabras a sus raíces.

Para tareas de este estilo, se implementan con el esquema one-vs-rest, de forma que se producen salidas multi-etiqueta a partir de modelos básicos. Más detalles en la implementación.

Como ventaja principal, podemos decir que todos estos modelos son rápidos de entrenar y baratos en cómputo; además suelen ser interpretables y fáciles de desplegar en infraestructuras propias sin dependencia de terceros. Además, su rendimiento es razonable cuando el dataset está bien representado por las características extraídas.

Por otro lado, sus limitaciones son claras: primero que todo, tenemos su dependencia con la representación del texto elegida; luego, requieren ingeniería de características cuidadosa; no capturan bien relaciones semánticas complejas ni el contexto amplio de la

trama; por último, su capacidad para generalizar a formulaciones lingüísticas nuevas es inferior a la de modelos basados en representaciones profundas.

1.1.2. Transformers

Para este trabajo se han planteado modelos de transformers que siguen dos arquitecturas: BERT [1], RoBERTa [2].

Los modelos basados en transformers como BERT, RoBERTa, emplean tokenizadores propios y representaciones contextualizadas que permiten entender mejor la semántica y las dependencias dentro del texto; pueden entrenarse/fine-tunearse para salida multi-etiqueta con relativa facilidad. Esto reduce la necesidad de ingeniería manual de características y suele mejorar la capacidad para clasificar o identificar clases en los textos.

Estos modelos NO necesitan de procesado del texto por nuestra parte, como sí sucede en los clásicos, por lo que su uso es más sencillo y menos delicado. Además de eso, entre sus ventajas destacan un rendimiento superior en tareas de NLP; flexibilidad para transfer learning; y buena adaptación al fine-tuning con relativamente pocos ejemplos etiquetados.

Como inconvenientes principales, están el consumo de más recursos y necesidad de GPU, así como de un gran coste temporal. Encima de eso, la gestión de hiperparámetros, el fine-tuning y la inferencia a gran escala, pueden encarecer el proyecto y requieren cuidado en la regularización y en el balanceo de clases para evitar sobreajuste.

1.1.3. LLMs (Grandes Modelos de lenguaje)

Por último, como LLMs se ha empleado la API de Gemini con su versión *gemini-2.5-pro* por su versatilidad y facilidad de implementación.

A nivel conceptual, hay que mencionar que para los LLM los *inputs* tampoco son vectores de características, sino más bien son un *prompt* en texto plano. Los LLMs modernos ofrecen la posibilidad de realizar la tarea directamente vía *prompting* [3] y/o con salida estructurada vía esquemas, por lo que a menudo no requieren entrenamiento ni fine-tuning adicional para obtener predicciones útiles.

Para usar estos modelos, basta con un prompt bien diseñado y ejemplos en contexto. Esto simplifica el pipeline de producción y acelera la experimentación, ya que el modelo incorpora un amplio conocimiento lingüístico y semántico. Como usaremos modelos gestionados por otros proveedores, ellos mismos se encargan del tokenizado de forma similar a como se hace con los transformers, cosa que simplifica aún más la tarea.

No obstante, usar los modelos más punteros es caro, ya que se hace mediante terceros, aquí van algunas desventajas: costes por token que pueden crecer con volúmenes grandes; latencia/red en la inferencia; límites de cuota; dependencia del proveedor y preocupaciones de privacidad/legislación al enviar datos a terceros. Además, la repetibilidad y control de versiones puede ser más compleja que en modelos locales.

En consecuencia los LLMs suelen ofrecer la mejor calidad *"out-of-the-box"* para este tipo de tareas sin aplicar fine-tune, pero su uso debe justificarse frente al coste y a las restricciones operativas que implican los servicios externos.

1.2 Dataset

El dataset original que se ha ofrecido al alumnado de la asignatura, contiene información de más de 9.400 películas. Estas vienen anotadas con su título (que las identifica), una descripción de la película y los géneros a los que pertenece (el objetivo). Más concretamente, vemos lo siguiente para cada película:

1. *movie_name*: Título de la película.
2. *genre*: Lista con Él o Los géneros de la película. P.ej. Drama, Comedy, Crime, etc..
3. *description*: Breve descripción o resumen de la trama.

A continuación, unos ejemplos de cómo se verían las entradas de dataset.

movie_name	**genre**	**description**
The Shawshank Redemption	Drama, Crime	Imprisoned in the 1940s ...
12 Angry Men	Drama	The defense and ...
The Dark Knight	Drama, Action, ...	Batman raises the ...

Tabla 1.1: Entradas ejemplo del dataset.

En total, hay 18 etiquetas de género distintas. Es así que cada película puede tener cualquier combinación de estas asignada, pero mínimo una.

1.3 Manejo del dataset

El dataset originalmente ha sido dividido en un 90 % de datos para su uso en entrenamiento y el resto de 'test'. Decimos 'test' porque es el conjunto de datos objetivo que ha de ser etiquetado por nosotros, y sobre el que se hará la evaluación final.

Ya que se han probado diferentes métodos de clasificación, se deben evaluar de forma consistente antes de atacar el conjunto de test final. Es así como hemos dividido este conjunto en otros dos sub-conjuntos, uno que utilizaremos como entrenamiento 'de verdad' y el otro como validación.

La partición se ha hecho sobre ese 90 % original, a proporción de 85 % y 15 % respectivamente, dejando alrededor de 7.200 datos para puro entrenamiento, y unos 1.270 para validar los distintos métodos de clasificación.

1.3.1. Procesado del dataset

Una vez particionado el dataset, se ha procesado para ser digerido por los modelos. Exceptuando el caso de los LLMs y los Transformers, los clasificadores, en general, no están diseñados para recibir directamente las palabras en lenguaje humano. Así pues, se ha procesado el dataset de distintas maneras según la necesidad de cada modelo.

1. **Modelos clásicos**: Para cada muestra, se ha juntado en un mismo texto el título y la descripción de la película para luego ser *tokenizado*. El tokenizado se ha hecho de forma similar a las prácticas de la asignatura eliminando tokens innecesarios o redundantes, signos de puntuación y, finalmente, aplicación de stemming. A partir de ahí, se han de convertir a vectores de características mediante alguna técnica como puede ser conteo.

Luego, para el manejo de salidas, se han binarizado usando la codificación one-hot, de forma que cada película tiene asociada un vector de 18 0s o 1s, estando a 1s en las componentes a los géneros que tiene asignados.

2. **Transformers**: A diferencia de los modelos clásicos, los transformers sí que son capaces de digerir tokens normales. La forma en la que lo hacen es usando también un tokenizador, sin embargo, lo hacen usando uno propio en vez que uno de uso general. De esta forma, pueden aprovechar mejor lo aprendido durante su preentrenamiento, y se pueden fine-tunear mejor.

   Además de eso, cuando juntamos el título con la descripción de la película, añadimos un separador entremedias `'[SEP]'`, para permitir que el modelo entienda que hay dos cosas distintas en el texto [4]. Como los tokens adquieren significado de sus vecinos, al introducir un separador, permitimos que el transformer entienda que son dos contextos distintos.

3. **LLMs**: De forma similar al Transformer, los LLMs pueden digerir texto humano a través de tokenizadores. La ventaja es que encima, si lo hacemos a través de alguna API, este trabajo se puede lograr sin necesidad de tokenización por nuestra parte. Solo necesitan un prompt con la tarea y toda la información necesaria.

   En concreto, usaremos una opción que nos proporcionan algunas API, en las que las salidas de los modelos de Lenguaje son 'estructuradas'. De esta forma, se les puede especificar el formato en el que queremos que den las predicciones, y así mejorando el procesado entero de predicción y procesado.

1.4 Evaluación

A partir de las salidas de los modelos, se binariza en formato one-hot si no lo está ya, y se calculan las métricas presentadas en el dataset: *accuracy*, *f1*, *precision*, *recall* y *hamming_loss*.

Se compararán los modelos con ellos, y utilizaremos en el *benchmark* los que mejor funcionen.

CAPÍTULO 2
Implementación

Como ya se ha mencionado, para abordar la tarea se han probado distintos modelos, a continuación un listado de ellos según las clasificaciones que se han considerado.

1. **Modelos clásicos**: Regresión Logística, XGBoost, Random Forest, SVC.
2. **Transformers**: BERT, ROBERTA
3. **LLMs**: Gemini-2.5-pro.

A continuación detallamos con más detalle cada uno de los modelos y el cómo se han usado, en cada uno de ellos, explicando como se ha atacado esta tarea multi-etiqueta.

2.1 Modelos clásicos

Para los cuatro modelos probados, hemos hecho uso de la implementación por parte de *scikit-learn* [5]. En concreto, sus implementaciones son las siguientes: Regresión Logística [6], XGBoost [7], Random Forest [8], SVC [9].

A pesar de esto, para hacer uso de estos modelos, hace falta extraer las características de los textos a analizar. En concreto, se han tokenizado los textos, eliminado *stopwords*, eliminado *signos de puntuación*, y aplicado *stemming*.

A partir de la lista de tokens resultantes, se han probado las distintas codificaciones como *Counter* [10] y *Tfidf* [11]. En pocas palabras, lo que hacen es contar la cantidad de veces que cada token aparece en el texto; con eso, el Tfidf además pondera los conteos según la cantidad de documentos en los que aparece cada token.

Se han dado como input los vectores extraídos por estos métodos a los modelos, consiguiendo las salidas o *logits*, una salida para cada una de las 18 posibles etiquetas. Podríamos definir las decisiones que se toman para cada modelo, se hace 'one versus the rest', donde los modelos deciden cada posible etiqueta de forma binaria. Esto en verdad quiere decir que no solo hay un modelo, sino que se entrena un modelo para cada una de las 18 etiquetas. Al hacer las predicciones, se lanzan todos a la vez, consiguiendo 18 resultados que se binarizan.

A partir de aquí, se han pasado las salidas para cada objetivo por una función *sigmoide* [12], consiguiendo probabilidades de clasificación binaria: si el valor está por encima del 50 % se selecciona el género (1), sino no (0).

De esa forma, al saber a qué género corresponde cada componente del vector, asignamos dichos géneros a cada película procesada.

2.2 Transformers

Para el uso de Transformers se prosigue como se ha mencionado: para obtener el texto se junta el título con la descripción de la película, mediante el separador `'[SEP]'`. Se codifican las entradas con los respectivos tokenizadores de cada transformer, y se obtienen las salidas.

Ahora, a diferencia de los métodos clásicos, NO necesitamos entrenar un modelo por cada clase, un mismo transformer puede tener como capa de salida 18 neuronas, de forma que cada una sirve para predecir un género. Luego, estos valores se pasan por una *sigmoide* y de nuevo, si el valor está por encima del 50 % se selecciona el género (1), sino no (0).

Las implementaciones de los transformers que se han usado son de modelos de Hugging Face [13]. En concreto, para cada una de las arquitecturas, se han probado los siguientes modelos / pesos / checkpoints.

1. **BERT**: Google/bert-base [14]
2. **ROBERTA**: FacebookAI/roberta-base [15], Microsoft/deberta-v3 [16], classla/xlm-roberta-base-multilingual-text-genre-classifier [17], FacebookAI/roberta-large-mnli [18]

2.3 LLMs

Por último, los modelos de lenguaje, se han implementado siguiendo el esquema que se especifica en la documentación del servicio [19]. Como pre-requisito se necesita una cuenta de Google AI Studio y una API-KEY para poder acceder a sus servicios.

Como se ha indicado anteriormente, estos servicios hacen uso de sus propios tokenizadores y *embeddings*, por lo que nosotros únicamente necesitamos hacer lo siguiente: mediante el prompt, indicarle al modelo de lenguaje cuál es la tarea, ejemplos de cómo realizarla, y en qué formato hay que devolver los resultados.

Si vemos más concretamente la implementación, para especificar el formato de las salidas utilizamos clases y enums. De esta forma, indicamos al milímetro cómo nos han de ser devueltas las salidas.

```python
# Define the schema for structured output
class MovieGenre(enum.Enum):
    ACTION = "Action"
    ADVENTURE = "Adventure"
    ANIMATION = "Animation"
    COMEDY = "Comedy"
    CRIME = "Crime"
    DRAMA = "Drama"
    FAMILY = "Family"
    FANTASY = "Fantasy"
    HISTORY = "History"
    HORROR = "Horror"
    MUSIC = "Music"
    MYSTERY = "Mystery"
    ROMANCE = "Romance"
    SCIENCE_FICTION = "Science Fiction"
    TV_MOVIE = "TV Movie"
    THRILLER = "Thriller"
```

```
    WAR = "War"
    WESTERN = "Western"


class MovieClassification(TypedDict):
    movie_name: str
    genres: List[MovieGenre]


class MovieBatchClassification(TypedDict):
    movies: List[MovieClassification]
```

En este caso, las salidas para cada película serán, el nombre de esta y una lista de géneros (lista la cual está restringida a la indicada con el enum). Con eso, podemos crear un prompt con las instrucciones, en concreto, el prompt usado tiene esta forma:

```
def generate_prompt(df_train: pd.DataFrame, df_batch: pd.DataFrame, genres: str)
    ↪ -> str:
    prompt = f"""
    Classify the following movie in any of these genres. More than one genre can
    ↪ be assigned.
    Genres: {genres}

    ================================================
    """

    for _, row in df_train.iterrows():
        prompt += f"Movie title: {row['movie_name']}\n"
        prompt += f"Plot: {row['description']}\n"
        prompt += f"Actual genres: {row['genre']}\n"
        prompt += "--------------------\n"

    prompt += """
    ================================================
    Now, classify the following movies returning a structured JSON
    response with the movie names and their genres.
    """

    for _, row in df_batch.iterrows():
        prompt += f"Movie title: {row['movie_name']}\n"
        prompt += f"Plot: {row['description']}\n\n"

    return prompt
```

Lo que estamos haciendo en esta función es: generar un texto en el que indicamos la tarea a resolver (linea 2 a linea 7), damos ejemplos etiquetados a partir del dataset de entrenamiento (linea 9 a linea 13) y le damos los datos que tiene que clasificar (1inea 15 a linea 25).

Una vez esté el prompt generado, es solo cuestión de llamar a la API añadiendo el formato de la respuesta.

```
import google.generativeai as genai

genai.configure(api_key='...')

# Create model with structured output
model = genai.GenerativeModel(
    'gemini-2.5-pro',
```

```
    generation_config=genai.GenerationConfig(
        response_mime_type="application/json", # Indicate structured output
        response_schema=MovieBatchClassification # Indicate exact output format
    )
)

# Generate prompt
prompt = generate_prompt(train_movies, val_movies, genres_str)

# Call API to get predictions
response = model.generate_content(prompt)
batch_result = json.loads(response.text)
```

A pesar de que se han omitido detalles, con esto tendríamos la estructura general de cómo extraer predicciones de un LLM muy potente.

CAPÍTULO 3
Coste de las soluciones

Todos los experimentos han sido ejecutados en una de nuestras máquinas personales. En concreto, se ha lanzado todo en una máquina con un Ryzen 9 9900, 64GB de ram y una GPU RTX 5070 de NVidia. Esto implica que no se ha hecho gasto alguno más que para el uso de la API de Gemini.

Según el conteo obtenido con la librería *tiktoken* [20], se han usado aproximadamente 1.35 millones de tokens de entrada y 0.15 millones de salida $\approx$ 1,5 millones de tokens, lo que coincide con el reporte final que nos ofrece la web.

En total, por estos 1.5 millones de tokens, se nos han cargado $\approx$ 30€ por el servicio. Ahora, hay que mencionar que el coste real para los alumnos ha sido de 0€. Esto ya que Google AI Studio, ofrece un crédito de 300€ al iniciar en su plataforma sin cargo alguno para el usuario. Gracias Google por 'patrocinar' esta competición.

CAPÍTULO 4
Resultado de las soluciones

A continuación presentamos los resultados que han obtenido los modelos planteados en cada categoría.

4.1 Clásicos

Los modelos clásicos se comportan todos de forma similar ante la tarea. Podemos destacar que todos los modelos han dado su mejor resultados al entrenarse con el *TfIDf*, pero el XGBoost ha dado mejores resultados con el *Counter*, algo peculiar.

También, dado el desbalanceo de clases, hemos comprobado que al modificar el threeshold de decisión su puntuación en validación mejoraba. Ajustando este valor para que sea distinto de 0.5, usando una búsqueda logaritmica, conseguimos los mejores valores.

El repunte en esta sección se lo lleva *Regressión Logistica*, con un score de 56,2 f1 en validación.

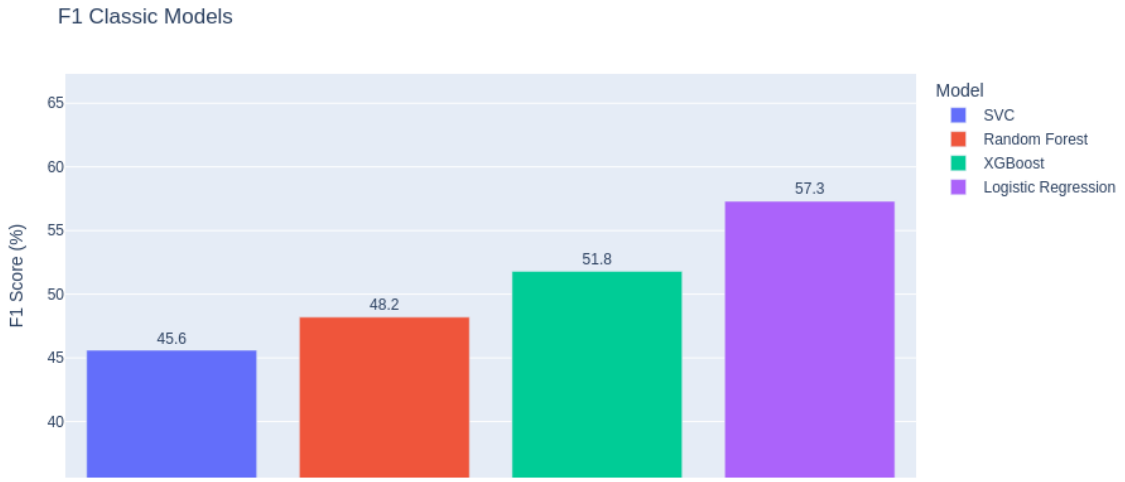

Figura 4.1: Clásicos f1 score

4.2 Transformers

Para los transformers probados, todos han dado buenas respuestas. Todos los competidores conseguido superar en la validación al baseline, sin embargo, el que mejores resultados obtenido ha sido por parte de *roberta-large-mnli*.

El modelo de facebook deja detrás al resto con un 66,7 f1 en validación,

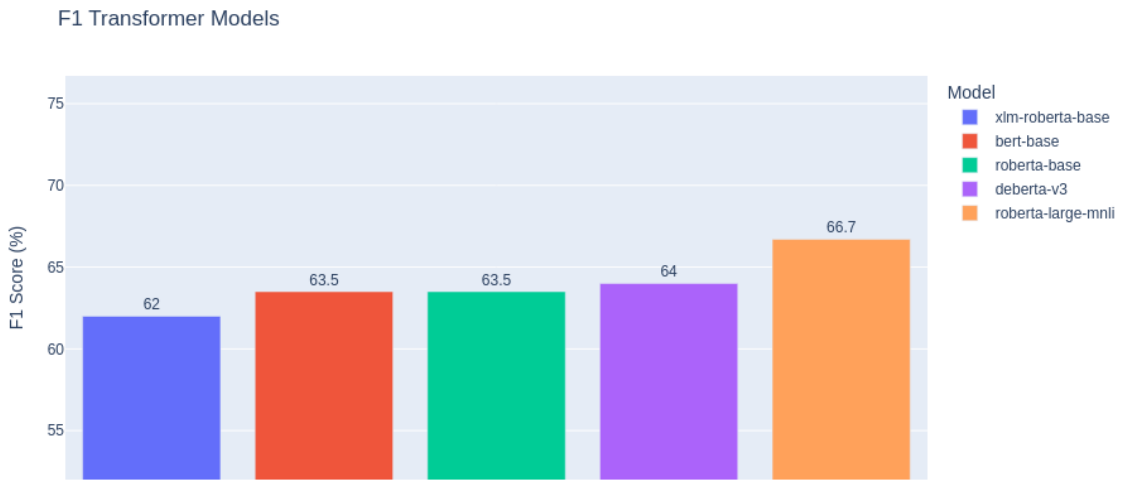

Figura 4.2: Transformers f1 score

4.3 LLMs

Por último, tenemos a Gemini. Como se esperaba de él, ha roto todos los esquemas, consiguiendo en validación una puntuación de 83,4 f1 en validación. Las capacidades de este modelo para tareas de este estilo se encuentra órdenes de magnitud por encima de los transformers clásicos.

A cambio de un coste y tiempo de inferencia mucho más grande, conseguimos unos resultados de libro.

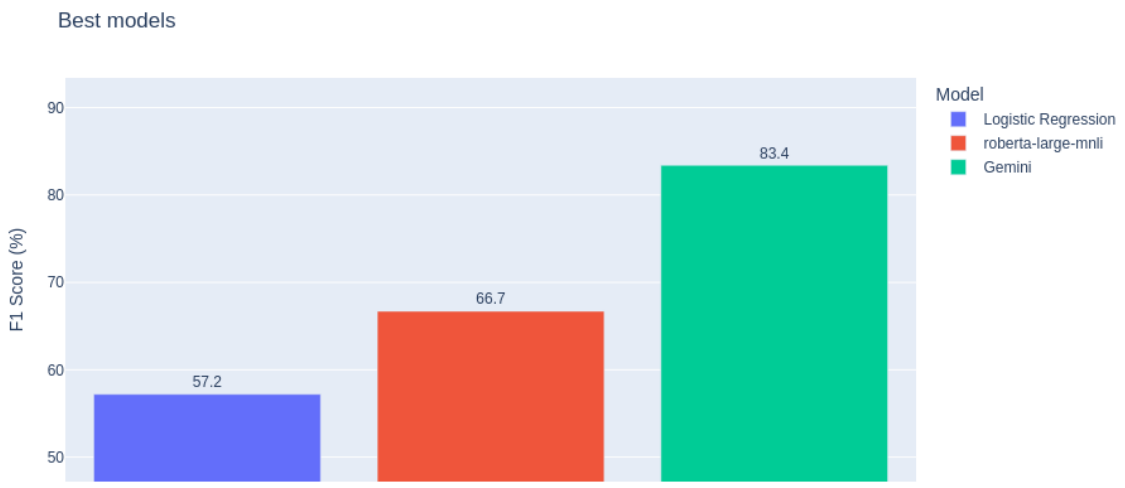

Figura 4.3: Mejores modelos

CAPÍTULO 5
Conclusiones

Tras la realización de este trabajo, hemos probado formas de abordar la tarea de la clasificación multi-clase con modelos de tres distintas categorías. Todas, con sus ventajas y desventajas han mostrado defender su punto en las respuestas, proporcionando en mayor o menor medida soluciones buenas al problema.

En concreto, podemos ver que si NO disponemos de grandes capacidades computacionales, tendremos que conformarnos con el uso de modelos clásicos, como mejor opción Regresión logística.

Por otro lado, si disponemos de GPUs de capacidad decente, podríamos atacar el problema mediante modelos más modernos como los transformers. Para esta tarea destacamos la arquitectura RoBERTa, en concreto la implementación de FaceBook: *roberta-large-mnli*.

Por último, si solo queremos los mejores resultados posibles en la tarea, nos da igual depender de servicios externos y el coste que tengan nuestras respuestas, deberemos ir con modelos de lenguaje tan grandes y capaces como podamos encontrar. Un ejemplo de estos es Gemini, con su versión *gemini-2.5-pro*.

Bibliografía

[1] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 11 de oct. de 2018. URL: https://arxiv.org/abs/1810.04805.

[2] Yinhan Liu et al. *ROBERTA: A robustly optimized BERT pretraining approach*. 26 de jul. de 2019. URL: https://arxiv.org/abs/1907.11692.

[3] Pengfei Liu et al. *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing*. 28 de jul. de 2021. URL: https://arxiv.org/abs/2107.13586.

[4] Hosein Mohebbi, Ali Modarressi y Mohammad Taher Pilehvar. *Exploring the Role of BERT Token Representations to Explain Sentence Probing Results*. 3 de abr. de 2021. URL: https://arxiv.org/abs/2104.01477.

[5] Fabian Pedregosa et al. *scikit-learn: machine learning in Python — scikit-learn 1.7.2 documentation*. 1 de ene. de 2025. URL: https://scikit-learn.org/stable/.

[6] Fabian Pedregosa, Gaël Varoquaux y Alexandre Gramfort. *LogisticRegression — scikit-learn documentation*. 1 de ene. de 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[7] Tony Chen y Tong He. *Using the scikit-learn Estimator Interface — XGBoost documentation*. 1 de ene. de 2025. URL: https://xgboost.readthedocs.io/en/stable/python/sklearn_estimator.html.

[8] Fabian Pedregosa y Gaël Varoquaux. *RandomForestClassifier — scikit-learn documentation*. 1 de ene. de 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[9] Fabian Pedregosa y Gaël Varoquaux. *SVC — scikit-learn documentation*. 1 de ene. de 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[10] Fabian Pedregosa y Gaël Varoquaux. *CountVectorizer — scikit-learn documentation*. 1 de ene. de 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.

[11] Fabian Pedregosa y Gaël Varoquaux. *TfidfVectorizer — scikit-learn documentation*. 1 de ene. de 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

[12] Wikipedia contributors. *Sigmoid function*. 2 de oct. de 2025. URL: https://en.wikipedia.org/wiki/Sigmoid_function.

[13] *Hugging Face – The AI community building the future.* URL: https://huggingface.co/.

[14] *google-bert/bert-base-uncased · Hugging Face*. URL: https://huggingface.co/google-bert/bert-base-uncased.

[15] *FacebookAI/roberta-base · Hugging Face*. URL: https://huggingface.co/FacebookAI/roberta-base.

[16] *microsoft/deberta-v3-base · Hugging Face*. URL: https://huggingface.co/microsoft/deberta-v3-base.

[17] *classla/xlm-roberta-base-multilingual-text-genre-classifier · Hugging Face*. URL: https://huggingface.co/classla/xlm-roberta-base-multilingual-text-genre-classifier.

[18] *FacebookAI/roberta-large-mnli · Hugging Face*. URL: https://huggingface.co/FacebookAI/roberta-large-mnli.

[19] *Gemini API in Vertex AI quickstart*. URL: https://cloud.google.com/vertex-ai/generative-ai/docs/start/quickstart?usertype=adc.

[20] *tiktoken*. 15 de dic. de 2022. URL: https://pypi.org/project/tiktoken/0.1.1/.