

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN



GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE
TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESARROLLO DE UN SISTEMA DE MEJORA DEL
RENDIMIENTO DE GRANDES MODELOS DE
LENGUAJE MEDIANTE TÉCNICAS DE
ENSAMBLE

MIGUEL FERNÁNDEZ VILLASANTE

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Título: Desarrollo de un Sistema de Mejora del Rendimiento de Grandes Modelos de Lenguaje Mediante Técnicas de Ensamble.

Autor: D. Miguel Fernández Villasante.

Tutor: D. Javier Conde Díaz.

Ponente: D.

Departamento:

MIEMBROS DEL TRIBUNAL

Presidente: D.

Vocal: D.

Secretario: D.

Suplente: D.

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de:

Madrid, a de de 20...

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN



GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE
TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Desarrollo de un Sistema de Mejora del Rendimiento de Grandes Modelos de Lenguaje Mediante Técnicas de Ensamble

Miguel Fernández Villasante

23 de junio de 2025

RESUMEN

Un modelo de lenguaje a gran escala (LLM) es una red neuronal con miles de millones de parámetros, entrenada en grandes volúmenes de texto mediante aprendizaje autosupervisado o semisupervisado. Su capacidad para comprender y generar lenguaje natural lo hace clave en aplicaciones como traducción, generación de contenido y análisis de texto. Su importancia radica en su potencial para automatizar procesos, mejorar la eficiencia en diversas industrias y hacer más accesible la información, facilitando la comunicación y el aprendizaje en múltiples idiomas y disciplinas. El *Stacking* es un método de ensamble de modelos en el que se combinan múltiples modelos base (de diferentes tipos o del mismo tipo con distintas configuraciones) y sus predicciones se usan como entrada para un modelo de nivel superior, llamado meta-modelo o meta-learner. Este modelo final aprende a combinar las predicciones de los modelos base para mejorar la precisión general. El presente TFG tiene como objetivo entrenar un meta-modelo de inteligencia artificial que supere en rendimiento a los grandes modelos de lenguaje disponibles en la actualidad. Para ello, se utilizarán estrategias de ensamble de modelos existentes. El modelo propuesto será evaluado con datasets como MMLU, MMLU-pro o TELE-IA.

SUMMARY

A large language model (LLM) is a neural network with billions of parameters, trained on large volumes of text using self-supervised or semi-supervised learning. Its ability to understand and generate natural language makes it key in applications such as translation, content generation, and text analysis. Its importance lies in its potential to automate processes, improve efficiency across various industries, and broaden access to information, facilitating communication and learning in multiple languages and disciplines. Stacking is an ensemble method in which multiple base models (of different types or of the same type with different configurations) are combined, and their predictions are used as input for a higher-level model, called a meta-model or meta-learner. This final model learns to combine the predictions of the base models to improve overall accuracy. This thesis aims to train an artificial intelligence meta-model that outperforms the large language models currently available. To achieve this, ensemble strategies of existing models will be used. The proposed model will be evaluated with datasets such as MMLU, MMLU-pro, or TELE-IA.

PALABRAS CLAVE

Modelos de Lenguaje a Gran Escala, Inteligencia Artificial, Ensamble de Modelos, Stacking, Meta-modelo, Procesamiento del Lenguaje Natural, Evaluación de Modelos, MMLU, TELE-IA

KEYWORDS

Large Language Models, Artificial Intelligence, Model Ensemble, Stacking, Meta-model, Natural Language Processing, Model Evaluation, MMLU, TELE-IA

Agradecimientos

...

Índice

Resumen y Palabras Clave	II
Agradecimientos	IV
Lista de acrónimos	VIII
1. Introducción y objetivos	1
1.1. Introducción	1
1.2. Objetivos	3
2. Marco teórico	5
2.1. Evaluación de LLM	5
2.2. Técnicas de ensamble en el aprendizaje automático	7
2.3. Técnicas de ensamble, stacking	9
2.4. Técnicas de ensamble, bagging	11
2.4.1. ¿Qué es el bagging?	11
2.4.2. Ventajas del bagging frente a otras técnicas de ensamble	11
2.5. Mejoras de los LLM	13
2.6. Selección de modelos	14
3. Desarrollo y resultados	16
3.1. Selección de modelos	16
3.1.1. Modelos base	16
3.2. Fase 1	17
3.2.1. Modelos seleccionados	23
3.3. Fase 2	27
3.3.1. Random Forest Classifier	27
3.3.2. K Neighbors Classifier	28
3.3.3. Support Vector Classifier	29
3.3.4. Linear Regression	31
3.3.5. Conclusiones de la Fase 2	33
3.4. Fase 3	34
3.4.1. Metodología	34
3.4.2. Optimización de hiperparámetros	35
3.4.3. Conclusiones de la Fase 3	38
4. Discusión	39
5. Líneas futuras	41
Bibliografía	42
A. Anexo A: Descripción del funcionamiento del <i>One Hot Encoder</i>	46

A.1. One Hot Encoder	46
A.2. Train Test Split	47
B. Anexo 1: Aspectos éticos, económicos, sociales y ambientales	49
C. Anexo 2: Presupuesto económico	50

Índice de figuras

3.1. Modelos ordenados por porcentaje de acierto.	18
3.2. Respuesta más típica para los n mejores modelos, para 11716 filas.	19
3.3. Porcentaje de acierto de los modelos según la categoría	22
3.4. Porcentaje de acierto	23
3.5. Porcentaje de acierto según el modelo/grupo y la categoría	24
3.6. Precisión de los principales grupos de modelos	28
3.7. Precisión de los principales grupos de modelos	29
3.8. Precisión de los principales grupos de modelos	30
3.9. Precisión de los modelos con diferentes valores de los parámetros booleanos .	31
3.10. Precisión de los principales grupos de modelos	32
3.11. Precisión de los modelos con diferentes valores de los parámetros booleanos .	32
3.12. Precisión de la red neuronal en comparación con los modelos base	35

Índice de tablas

3.1. Modelos usados según la categoría para obtener los mejores resultados	20
3.2. Modelos usados según la categoría para obtener los mejores resultados. “T→C” y “C→T” especifican el orden de las ejecuciones: respuesta más típica (T) y categoría (C)	21
3.3. Modelos usados según la categoría para obtener los mejores resultados. “T→C” y “C→T” especifican el orden de las ejecuciones: respuesta más típica (T) y categoría (C)	25
3.4. Modelos usados según la categoría para obtener los mejores resultados	26
3.5. Mejor combinación de hiperparámetros y la precisión lograda.	28
3.6. Mejor combinación de hiperparámetros y la precisión lograda.	29
3.7. Mejor combinación de hiperparámetros y la precisión lograda.	31
3.8. Mejor combinación de hiperparámetros y la precisión lograda.	33
3.9. Resultados de la Fase 2	33
3.10. Precisión obtenida según la función de activación utilizada.	35
3.11. Precisión obtenida según la tasa de abandono utilizada.	36
3.12. Precisión obtenida según la configuración de capas ocultas.	36
3.13. Precisión obtenida según el tamaño de lote utilizado.	36
3.14. Precisión obtenida según el ratio de aprendizaje.	37
3.15. Precisión obtenida según el número de épocas.	37
3.16. Mejor combinación de hiperparámetros y la precisión lograda.	38
4.1. Resultados de todas las metodologías empleadas.	40
C.1. Presupuesto económico	50

Lista de acrónimos

LLM *Large Language Model* (Modelo de Lenguaje Grande): Se refiere a modelos de inteligencia artificial entrenados con grandes volúmenes de datos textuales y que poseen miles de millones de parámetros, como GPT y BERT.

IA Inteligencia Artificial: Campo de estudio que busca desarrollar sistemas capaces de realizar tareas que requieren inteligencia humana, como el procesamiento del lenguaje natural y la toma de decisiones.

MoE *Mixture of Experts* (Mezcla de Expertos): Arquitectura de red neuronal que combina múltiples modelos especializados (expertos) para mejorar el rendimiento en tareas específicas, activando solo un subconjunto de expertos para cada entrada.

SMoE *Sparse Mixture of Experts* (Mezcla Dispersa de Expertos): Arquitectura de red neuronal que activa solo un subconjunto de expertos especializados para cada entrada, mejorando la eficiencia y la escalabilidad del modelo.

CART *Classification and Regression Trees* (Árboles de Clasificación y Regresión): Algoritmo de aprendizaje supervisado que utiliza árboles de decisión para realizar tareas de clasificación y regresión, dividiendo los datos en función de características específicas.

FLOPS *Floating Point Operations Per Second* (Operaciones de Punto Flotante por Segundo): Unidad de medida que indica la cantidad de operaciones matemáticas que un sistema puede realizar en un segundo, utilizada para evaluar el rendimiento de hardware y algoritmos.

MMLU *Massive Multitask Language Understanding* (Comprensión de Lenguaje Multitarea Masiva): Conjunto de tareas diseñado para evaluar la capacidad de los modelos de lenguaje en una amplia gama de tareas de comprensión del lenguaje natural.

GPU *Graphics Processing Unit* (Unidad de Procesamiento Gráfico): Componente de hardware especializado en el procesamiento paralelo, utilizado para acelerar cálculos complejos en aplicaciones como gráficos por ordenador y aprendizaje automático.

API *Application Programming Interface* (Interfaz de Programación de Aplicaciones): Conjunto de reglas y protocolos que permiten la interacción entre diferentes software, facilitando la comunicación y el intercambio de datos entre aplicaciones.

1. Introducción y objetivos

1.1. Introducción

En los últimos años, los modelos de lenguaje de gran tamaño (*Large Language Models*, LLM) han revolucionado el campo de la inteligencia artificial, demostrando capacidades sin precedentes en tareas como la traducción automática, la generación de contenido y el análisis de texto. Estos modelos, entrenados con miles de millones de parámetros y grandes volúmenes de datos, han establecido un nuevo paradigma en la comprensión y generación de lenguaje natural por parte de máquinas. Su impacto ha sido significativo en diversas áreas, desde la automatización de procesos hasta hacer la información más accesible, permitiendo que personas y organizaciones aprovechen estas tecnologías para resolver problemas complejos y mejorar la eficiencia en sus actividades. Además, la rápida evolución de los LLM ha impulsado el desarrollo de nuevas aplicaciones en sectores tan variados como la atención al cliente, la investigación científica y la creatividad digital.

El avance de los LLM ha sido posible gracias a mejoras en la arquitectura de redes neuronales, como los transformadores, y al acceso a infraestructuras computacionales cada vez más potentes. La introducción de arquitecturas como BERT, GPT y sus sucesores ha permitido a los modelos capturar relaciones contextuales profundas en el texto, superando ampliamente a los enfoques tradicionales basados en reglas o en modelos estadísticos más simples. Sin embargo, el crecimiento exponencial en el tamaño de los modelos y los conjuntos de datos también ha planteado desafíos en términos de coste computacional, consumo energético y sostenibilidad. Por ello, la comunidad científica ha comenzado a explorar alternativas que permitan optimizar el rendimiento de los modelos sin necesidad de aumentar indefinidamente su tamaño, como la especialización de modelos, el aprendizaje por transferencia y las técnicas de ensamble.

A pesar de su éxito, los LLM presentan limitaciones en tareas específicas que requieren razonamiento complejo y precisión, como es la resolución de preguntas tipo test. Estas tareas son especialmente desafiantes porque combinan la necesidad de comprender el contexto, analizar múltiples opciones y seleccionar la respuesta correcta. En este contexto, los modelos actuales suelen cometer errores debido a su incapacidad para razonar de manera estructurada o integrar información de manera eficiente. Además, estas limitaciones se hacen más evidentes en dominios especializados, donde el conocimiento requerido puede no estar completamente representado en los datos de entrenamiento de los modelos. La falta de interpretabilidad y la tendencia a generar respuestas plausibles pero incorrectas son también obstáculos importantes para su adopción en entornos críticos.

La resolución de preguntas tipo test no solo implica la comprensión literal del enunciado, sino también la capacidad de inferir información implícita, descartar distractores y, en ocasiones, aplicar conocimientos de sentido común o de dominio específico. Por ejemplo, en contextos educativos, las preguntas pueden estar diseñadas para evaluar habilidades cognitivas de orden

superior, como el análisis, la síntesis o la evaluación crítica. En estos casos, los modelos de lenguaje deben ir más allá de la simple recuperación de información y demostrar capacidades de razonamiento avanzado, lo que representa un reto significativo para las arquitecturas actuales.

El interés en mejorar el rendimiento de los modelos en este tipo de tareas ha crecido en los últimos años, ya que su resolución tiene aplicaciones prácticas en áreas como la educación, facilitando la evaluación de conocimientos y la construcción de entornos educativos virtuales con capacidades inteligentes. Por ejemplo, sistemas de tutoría virtual basados en inteligencia artificial podrían proporcionar retroalimentación personalizada a los estudiantes, ayudándolos a identificar y superar sus debilidades. Asimismo, en sectores como la medicina o el derecho, la capacidad de responder preguntas complejas con precisión podría transformar la forma en que los profesionales acceden y utilizan el conocimiento especializado. Además, la automatización de exámenes y pruebas de certificación mediante IA puede contribuir a una evaluación más objetiva y eficiente, reduciendo la carga de trabajo de los docentes y mejorando la equidad en los procesos de evaluación.

En el ámbito empresarial, la capacidad de los modelos para responder preguntas tipo test puede ser aprovechada en procesos de selección de personal, formación continua y certificación de competencias. Las organizaciones pueden beneficiarse de sistemas automáticos que evalúen el conocimiento de sus empleados de manera rápida y precisa, facilitando la identificación de necesidades formativas y la toma de decisiones estratégicas en recursos humanos. Por otro lado, en el sector público, la integración de modelos de IA en plataformas de servicios ciudadanos puede mejorar la accesibilidad y la calidad de la información ofrecida a la población.

En los últimos años, se han propuesto diversas estrategias para abordar estas limitaciones, incluyendo el uso de técnicas de *prompt engineering*, el ajuste fino (*fine-tuning*) en conjuntos de datos específicos y la integración de fuentes de conocimiento externas. El *prompt engineering* ha permitido adaptar los modelos a tareas concretas mediante la formulación cuidadosa de las instrucciones, mientras que el ajuste fino ha demostrado ser eficaz para especializar los modelos en dominios particulares. Sin embargo, estas aproximaciones no siempre logran superar las barreras inherentes a la arquitectura de los modelos o a la calidad de los datos disponibles. Por ello, resulta fundamental explorar enfoques alternativos que permitan aprovechar las ventajas de distintos modelos y técnicas, con el objetivo de obtener resultados más robustos y precisos.

En este contexto, las estrategias de ensamble, como el método de *stacking*, han demostrado ser una herramienta poderosa para mejorar el rendimiento de los sistemas de aprendizaje automático. Estas estrategias permiten combinar múltiples modelos base, aprovechando sus fortalezas individuales y mitigando sus debilidades, para obtener un modelo final más robusto y preciso. El *stacking*, en particular, consiste en entrenar un modelo de nivel superior que aprende a integrar las predicciones de varios modelos base, optimizando así la toma de decisiones en tareas complejas.

El uso de técnicas de ensamble no es exclusivo del procesamiento del lenguaje natural, sino que ha sido ampliamente adoptado en otras áreas de la inteligencia artificial, como la visión por computador y la predicción de series temporales. En estos campos, los métodos de ensamble han permitido alcanzar resultados de vanguardia en aplicaciones industriales. Su éxito radica en la capacidad de reducir el sesgo y la varianza de los modelos individuales, mejorando la generalización y la robustez frente a datos no vistos. Aplicar estos principios al ámbito de los LLM y las tareas de razonamiento complejo representa una oportunidad para avanzar en la frontera del conocimiento en inteligencia artificial.

En este Trabajo de Fin de Grado se propone abordar estas limitaciones mediante el uso de

estrategias de ensamble de modelos, específicamente el método de *stacking*. Este enfoque no solo permite mejorar el rendimiento general de los modelos en tareas de razonamiento complejo, sino que también abre nuevas posibilidades para explorar cómo diferentes arquitecturas y técnicas pueden complementarse entre sí. Además, se busca que los resultados obtenidos contribuyan al avance de la inteligencia artificial, proporcionando una base sólida para futuras investigaciones en este campo. La validación experimental de este enfoque permitirá evaluar su eficacia en comparación con los modelos individuales y sentar las bases para su aplicación en escenarios reales.

El desarrollo de este trabajo implica una revisión exhaustiva del estado del arte en modelos de lenguaje, técnicas de ensamble y métodos de evaluación en tareas tipo test. Se analizarán los principales retos asociados a la integración de modelos heterogéneos, así como las métricas más adecuadas para medir el rendimiento en contextos de razonamiento complejo. Asimismo, se explorarán posibles limitaciones y riesgos asociados al uso de modelos de IA en la toma de decisiones, incluyendo aspectos éticos, de transparencia y de equidad.

En las siguientes secciones, se explorarán los fundamentos teóricos de los modelos de lenguaje, las técnicas de ensamble y su aplicación en tareas de razonamiento complejo. También se describirá cómo este trabajo se enmarca en el contexto de los avances recientes en el procesamiento del lenguaje natural, destacando su relevancia y potencial impacto en aplicaciones prácticas. Finalmente, se presentarán los resultados obtenidos y se discutirán las posibles líneas de investigación futura que surgen a partir de este trabajo.

En resumen, la combinación de modelos de lenguaje avanzados y técnicas de ensamble representa una vía prometedora para superar las limitaciones actuales en la resolución de preguntas tipo test. Este trabajo pretende aportar una contribución significativa en este ámbito, sentando las bases para el desarrollo de sistemas de inteligencia artificial más precisos, robustos y adaptables a las necesidades de los usuarios. A medida que la inteligencia artificial continúa evolucionando, es fundamental seguir explorando nuevas estrategias y enfoques que permitan maximizar su potencial y abordar los desafíos más complejos de la sociedad actual.

1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es desarrollar y evaluar un modelo de inteligencia artificial que supere en rendimiento a los modelos actuales en la resolución de preguntas tipo test, utilizando estrategias de ensamble. Para alcanzar este objetivo general, se plantean los siguientes pasos:

- **Estudiar el estado del arte:** realizar una revisión exhaustiva de la literatura sobre la evaluación de LLM y las técnicas de ensamble, con el fin de identificar las metodologías más relevantes y comprender las limitaciones actuales.
- **Familiarizarse con la tecnología:** explorar y analizar las herramientas, frameworks y recursos necesarios para trabajar con LLM y estrategias de ensamble, asegurando una implementación eficiente del proyecto.
- **Seleccionar los modelos base:** identificar y seleccionar los modelos de lenguaje más adecuados para incluir en el estudio, considerando su rendimiento, arquitectura y disponibilidad.
- **Analizar las limitaciones de los modelos seleccionados:** determinar los principales desafíos que enfrentan los LLM en la resolución de preguntas tipo test, además de

obtener las fortalezas y debilidades de cada uno.

- **Diseñar y entrenar el meta-modelo:** implementar un meta-modelo basado en el método de *stacking*, utilizando las predicciones de los modelos base como entrada, y entrenarlo para optimizar su capacidad de razonamiento y precisión.
- **Evaluar el meta-modelo:** validar el rendimiento del modelo propuesto utilizando conjuntos de datos estándar como MMLU, MMLU-pro y TELE-IA, midiendo su precisión, capacidad de razonamiento y generalización en diferentes dominios.
- **Analizar los resultados:** comparar el rendimiento del meta-modelo con los modelos de lenguaje actuales, identificando fortalezas, debilidades y posibles áreas de mejora.
- **Contribuir al avance del estado del arte:** proponer conclusiones y recomendaciones basadas en los resultados obtenidos, destacando cómo las estrategias de ensamble pueden aplicarse a otros problemas complejos en el procesamiento del lenguaje natural.

Estos objetivos están diseñados para guiar el desarrollo del trabajo de manera estructurada, asegurando que cada fase contribuya al cumplimiento del objetivo general. Además, se espera que los resultados obtenidos sirvan como base para futuras investigaciones en el campo de la inteligencia artificial y el procesamiento del lenguaje natural.

2. Marco teórico

En este trabajo, el marco teórico recoge los conceptos clave para entender el enfoque propuesto y su importancia dentro del procesamiento del lenguaje natural y la inteligencia artificial.

2.1. Evaluación de LLM

Métricas de evaluación de los LLM

La evaluación de un LLM es una tarea tan importante como su construcción, permite evaluar su precisión y mitigar sesgos en sus respuestas [1]. Las métricas de esta evaluación se dividen en tres categorías principales [1]:

- **Métricas cualitativas y éticas:** evalúan la calidad subjetiva y la alineación con valores sociales, es decir, se pretende identificar y reducir los sesgos, evitando reproducir estereotipos [1]. Un ejemplo de estos sesgos se muestra en el estudio de la IA escribiendo cartas de recomendación, donde describía a los hombres por sus capacidades profesionales, mientras que las mujeres eran descritas por su apariencia física [2].
- **Métricas de caso de uso:** evalúan hasta qué punto el modelo contribuye a alcanzar los objetivos concretos de las aplicaciones para las que está destinado y si sus respuestas están alineadas con sus metas [2].
- **Métricas de alucinación:** evalúan si las respuestas generadas por el modelo son correctas desde el punto de vista factual, es decir, si la información proporcionada es verídica y se ajusta a los hechos comprobables [2].

Para una evaluación integral, además de las métricas, se emplean la retroalimentación del usuario y la comparación con conjuntos de datos estándar (*benchmarking*) [1]. Un enfoque efectivo para el mantenimiento y la mejora continua de los LLM es el desarrollo orientado a métricas, que implica seguir medidas clave durante largos periodos de tiempo, obteniendo información del desempeño del modelo y encaminar sus futuras mejoras [1].

Benchmarks

Los *benchmarks* son conjuntos de datos y protocolos utilizados para probar y comparar el rendimiento de diferentes LLM en diversas tareas [3]. Estos buscan simular escenarios de procesamiento de lenguaje del mundo real y abarcan una variedad de tareas, como clasificación de texto, traducción automática, comprensión de lectura y generación de diálogos [3].

Categorías de los *benchmarks*: los *benchmarks* se pueden clasificar en tres categorías principales según el tipo de tareas que evalúan [3]:

- **Benchmarks para tareas de lenguaje generales:** dedicados a evaluar el rendimiento general de los LLM en una amplia gama de tareas, como clasificación de texto, traducción automática y comprensión de lectura [3]. MMU es un ejemplo de este tipo [3].
- **Benchmarks para tareas específicas o de dominio:** dedicados a evaluar el rendimiento de los LLM en aplicaciones y dominios más especializados, como la generación de código o la respuesta a preguntas médicas [3].
- **Benchmarks para tareas multimodales:** dedicados a evaluar modelos que procesan y generan contenido a partir de múltiples modalidades, como texto e imágenes [3].

Los *benchmarks* son herramientas esenciales y en constante evolución para medir sistemáticamente el rendimiento de los LLM [3]. Permiten no solo comparar modelos y rastrear mejoras, sino también identificar sus fortalezas y limitaciones en un espectro cada vez más amplio de tareas y dominios [3].

LLM-as-a-judge

LLM-as-a-judge (LLM-como-juez) es una técnica que pretende averiguar si la respuesta generada contiene alucinaciones¹ [1]. Se trata de un enfoque innovador para la evaluación de modelos de lenguaje, donde un LLM actúa como juez para evaluar las respuestas que ha generado otro [1, 4]. Una de sus mayores ventajas es la eficiencia y escalabilidad, pues la evaluación humana es costosa y difícilmente escalable [4]. Este enfoque es especialmente útil en tareas donde no existe una verdad universal, como la generación de resúmenes o diálogos [1].

Sin embargo, los *LLM-as-a-judge* contiene alguna limitación clave. Estos LLM pueden exhibir sesgos de autopreferencia, es decir, que pueden evaluar sus propias salidas más favorablemente que la de otros modelos, lo que compromete la objetividad [1, 4]. Este proceso, además de ser computacionalmente costoso por sí solo, puede requerir el uso de múltiples jueces para reducir los sesgos autoprioritarios [1].

Además de los sesgos de autopreferencia, los *LLM-as-a-judge* pueden verse afectados por otros sesgos, como el sesgo de orden (la posición de los candidatos influye en el juicio), el sesgo de longitud (preferencia por respuestas más largas o más cortas). Estos y la vulnerabilidad a manipulaciones adversarias pueden alterar los juicios del LLM, afectando a sus resultados [4].

Las evaluaciones de los resultados generados por los LLM pueden producirse en formato de puntuación, clasificación o selección [4]. Los atributos que puede juzgar incluyen: utilidad, inofensividad, fiabilidad, relevancia, viabilidad y calidad general [4].

La evaluación de los LLM es esencial para su precisión y para mitigar sesgos [1]. Esta se apoya en métricas cualitativas, de caso de uso y de alucinación, complementadas con benchmarks estandarizados [1]. Un avance clave es el enfoque *LLM-as-a-judge*, donde un LLM evalúa a otros, ofreciendo eficiencia y escalabilidad [4]. La evolución continua de la evaluación es fundamental para la mejora de los LLM [3].

¹El término “alucinaciones” se refiere a aquellas respuestas generadas por el modelo que parecen plausibles o correctas, pero son incorrectas o inventadas. Es decir, el modelo “alucina” información que no está respaldada por hechos reales o los datos de entrenamiento.

2.2. Técnicas de ensamble en el aprendizaje automático

El aprendizaje de ensamble o *ensemble learning* es una técnica fundamental en el campo de *machine learning* y la Ciencia de Datos. Permite mejorar la precisión y la robustez de las predicciones al combinar las decisiones de múltiples modelos en lugar de depender de uno solo. Estos modelos individuales, denominados “modelos base” o “predictores”, pueden ser clasificadores, regresores o algoritmos no supervisados, cuya diversidad y combinación estratégica conducen a un modelo final con un rendimiento superior. La lógica detrás de esta idea es que el conocimiento colectivo supera la capacidad de un único experto. [5, 6, 7]

Ventajas del aprendizaje de ensamble

La preferencia por las técnicas de ensamble sobre los modelos individuales se puede justificar por varias razones fundamentadas en las limitaciones de los modelos simples:

- **Mejora en la generalización y precisión:** una de las ventajas más significativas es la capacidad de obtener un mejor rendimiento de generalización. Al combinar las predicciones de varios modelos, el ensamble reduce el riesgo de errores de los individuos, logrando una menor tasa de error. Esto se atribuye a la reducción del sesgo (*bias*) y la varianza (*variance*) de las predicciones. [5]
- **Reducción del sobreajuste y fomento de la diversidad:** los métodos de ensamble tienden a sobreajustar menos que los modelos individuales, ya que la diversidad entre los modelos base ayuda a mitigar la tendencia a memorizar el ruido de los datos de entrenamiento [5, 6]. Para fomentar esta diversidad, se emplean técnicas como el muestreo de datos (entre otros, el *bagging*), la manipulación del espacio de características (como el *random forest*) o la introducción de ruido aleatorio en las salidas (por ejemplo, *output smearing*). Además, métodos como el *dropout* entrenan modelos con pesos compartidos, lo que permite reducir el sobreajuste. [5]
- **Manejo de la heterogeneidad de datos y modelos:** las técnicas de ensamble pueden combinar eficazmente modelos de diferentes familias o múltiples instancias del mismo algoritmo. Esta flexibilidad permite explotar las fortalezas de distintos algoritmos, resultando en un alto rendimiento en gran variedad de problemas, incluso cuando los modelos base tienen fortalezas y debilidades diferentes. [5, 6]

Estrategias de ensamble principales

Hay numerosas estrategias de ensamble, cada una con sus propias características y beneficios. Algunos ejemplos notables son los siguientes:

- **Stacking (*stacked generalization*):** Es una técnica que utiliza un meta-modelo para aprender a combinar las predicciones de varios modelos base. Estos generan predicciones individuales que luego actúan como características de entrada para el meta-modelo. Mediante la estrategia de *winner-takes-all* (el vencedor se lleva todo), que se resume en que la respuesta más típica es la adecuada, el meta-modelo produce la respuesta final. [7]
- **Boosting:** entrena modelos base de manera secuencial, donde cada modelo sucesivo corrige los errores o las clasificaciones incorrectas de los modelos anteriores. Se enfatizan los ejemplos mal clasificados por los modelos previos, permitiendo una mejora pro-

gresiva del porcentaje de acierto [6]. Ejemplos populares incluyen *AdaBoost*, *Gradient Boosting* y *XGBoost*. [7]

- **Bagging (*bootstrap aggregating*):** es una técnica que genera múltiples subconjuntos de datos de entrenamiento mediante “muestreo con reemplazo” (*bootstrapping*). Cada modelo base se entrena de forma independiente con un subconjunto de datos diferente [7]. Las predicciones finales se combinan mediante votación o promediado. *Random Forest* es una mejora de los árboles de decisión que utiliza esta estrategia. Esta técnica es especialmente útil para reducir la varianza. [5]
- **Ensamblados explícitos e implícitos (*explicit/implicit ensembles*):**
 - Los ensambles implícitos entrenan un único modelo de tal manera que se comporta como un ensamble de múltiples redes neuronales sin necesidad de aumentar el costo computacional significativamente. Ejemplos notables incluyen *Dropout* y *DropConnect*, que crean diversidad al omitir aleatoriamente unidades y conexiones, respectivamente, durante el entrenamiento. [5]
 - Los ensambles explícitos, por otro lado, no comparten parámetros y combinan directamente las predicciones de modelos completamente separados. *Snapshot Ensembles* captura “instantáneas” de un modelo en diferentes puntos de su trayectoria de optimización y posteriormente se hace una media de todas estas imágenes. [5]
- **Estrategias de fusión de decisiones (*decision fusion strategies*):** Más allá de las arquitecturas de ensamble, la forma en que se combinan las salidas de los modelos base es crucial para el rendimiento final. Estas estrategias incluyen *Majority Voting*, *Unweighted Model Averaging*, *Bayes Optimal Classifier*, *Super Learner*, *Consensus* y *Query-By-Committee*. [5, 7]

Aplicaciones y desafíos

Las técnicas de ensamble han sido empleadas con éxito en una amplia gama de aplicaciones, desde la clasificación de imágenes y videos hasta el sector de la salud y se han implementado en problemas de clasificación, regresión y *clustering*. En cuanto a ejemplos específicos, se incluye el reconocimiento facial, la detección de objetos, segmentación semántica o procesamiento del lenguaje natural (incluyendo reconocimiento de voz y análisis de sentimientos). La presencia de tantas tareas diferentes demuestra su versatilidad. [5]

A pesar de sus innegables beneficios, la implementación de modelos de ensamble conlleva ciertas consideraciones. Generalmente, se requiere más tiempo y recursos computacionales para el entrenamiento, especialmente cuando se utilizan conjuntos de datos grandes o un número elevado de modelos base. Además, la interpretación de los resultados puede ser más compleja debido a la combinación de múltiples modelos.

En definitiva, las técnicas de ensamble se han consolidado como una herramienta clave en *machine learning* gracias a su capacidad para aprovechar las fortalezas de varios modelos y mejorar la precisión de las predicciones. Además de ayudar a reducir el sesgo, la varianza y el sobreajuste, su flexibilidad para adaptarse a distintos tipos de problemas las convierte en una opción muy recomendable en la mayoría de los escenarios de modelización predictiva.

2.3. Técnicas de ensamble, stacking

Tras haber explorado las técnicas de ensamble de modelos de forma general, se profundizará en la estrategia de *stacking* (apilamiento) o *ensemble-stacking*. Esta técnica se enmarca dentro del aprendizaje por ensamble y tiene como objetivo combinar las predicciones de múltiples modelos individuales, denominados modelos base o *base learners*, para obtener una predicción final con mejor rendimiento. [8, 9]

¿Qué es el stacking?

El *stacking* se diferencia de otros métodos de ensamble por su estructura de dos niveles. Primero, se entrenan varios modelos base sobre el mismo conjunto de datos. Luego, en lugar de combinar directamente sus predicciones (como en el *Bagging*) o corregirlas de forma secuencial (como en el *Boosting*), estas predicciones se utilizan como entrada para un modelo adicional, llamado meta-modelo o *meta-learner*. Este meta-modelo aprende cómo combinar de manera óptima las predicciones de los modelos base para generar la predicción final. [8, 9]

El principal objetivo del *stacking* es aprovechar las ventajas de distintos algoritmos de aprendizaje automático y reducir sus limitaciones individuales. Al integrar las capacidades predictivas de varios modelos, el *stacking* puede capturar patrones más complejos en los datos y lograr una mayor precisión en las predicciones. No obstante, es una técnica que puede requerir más recursos computacionales y un ajuste cuidadoso para evitar el sobreajuste. [8, 9]

El flujo de trabajo del *stacking* se puede desglosar en los siguientes pasos [9]:

1. **Preparación de los datos:** mediante la identificación de características relevantes, la limpieza de datos y, fundamentalmente, la división en conjuntos de entrenamiento y prueba, se realiza la preparación de los datos para entrenar el modelo. Una partición adecuada es necesaria para prevenir la fuga de datos, esto es el uso de datos de prueba para el aprendizaje, situación que puede provocar una sobreestimación de los resultados finales. [9]
2. **Selección de los modelos:** se eligen cuidadosamente los modelos individuales que formarán parte del ensamble. Es esencial diversificar las características de estos, pues la variedad garantiza que se generen diferentes tipos de errores y que los modelos se complementen mutuamente. [9]
3. **Entrenamiento de los modelos base:** cada modelo base se entrena de forma independiente sobre el conjunto de entrenamiento original. Se pueden emplear distintos algoritmos o configuraciones de hiperparámetros para fomentar aún más la diversidad entre ellos. [9]
4. **Generación de predicciones en el conjunto de prueba:** una vez entrenados, los modelos base se utilizan para generar predicciones sobre el conjunto de prueba. Estas predicciones son cruciales, ya que se convierten en las nuevas características de entrada para el meta-modelo. [9]
5. **Desarrollo del meta-modelo:** se define el meta-modelo, que es de un nivel superior a los modelos base. Su propósito es aprender a combinar las predicciones originales para producir la respuesta final. El meta-modelo puede generarse a partir de cualquier algoritmo, desde una simple regresión lineal hasta redes neuronales. [9]
6. **Entrenamiento del meta-modelo:** el meta-modelo se entrena utilizando como entrada

las predicciones generadas por los modelos base. Librerías como *scikit-learn* facilitan esto, su clase “StackingRegressor”, donde los modelos base se especifican en el parámetro “estimators” y el meta-modelo en “final_estimator”, permite la combinación de modelos base. Para evitar el sobreajuste, se puede entrenar el meta-modelo mediante validación cruzada. [8, 9]

7. **Realizar predicciones en el conjunto de prueba:** el ensamble de *stacking* se emplea para realizar predicciones sobre el conjunto de prueba. Se obtienen las predicciones de los modelos base en dicho conjunto y el meta-modelo la utiliza como entrada para generar la respuesta final. [9]
8. **Evaluación del modelo:** por último, se evalúa el rendimiento del ensamble comparando sus predicciones con los valores reales del conjunto de prueba. [9]

Ventajas del stacking frente a otras técnicas de ensamble

Si bien el *bagging* y el *boosting* son técnicas de ensamble que ofrecen buenos resultados, el *stacking* ofrece beneficios distintivos que lo posicionan como una opción preferente en diversas situaciones, especialmente cuando se busca la máxima precisión predictiva. Las ventajas clave del *stacking* incluyen [9]:

- **Rendimiento predictivo mejorado:** el uso de *stacking* permite aprovechar la diversidad de varios modelos base para generar predicciones más precisas. Al combinar las salidas de estos modelos mediante un meta-modelo, se pueden identificar relaciones en los datos que podrían pasar desapercibidas para un solo modelo. Esta integración de diferentes perspectivas contribuye a una mayor robustez y exactitud en los resultados, ya que el meta-modelo aprende a extraer lo mejor de cada modelo base y a minimizar sus posibles errores. [8, 9]
- **Diversidad de modelos:** el éxito del *stacking* radica en la diversidad de los modelos base ya que fomenta el uso de modelos entrenados con metodología variadas, estos es algoritmos, arquitecturas y configuraciones de hiperparámetros diferentes, por ejemplo, combinando modelos lineales con modelos basados en árboles. Esta heterogeneidad es crucial porque asegura que los modelos base cometan diferentes tipos de errores, lo que a su vez reduce el riesgo de sobreajuste. [8, 9]
- **Flexibilidad y aplicabilidad:** el *stacking* es una estrategia altamente versátil que puede aplicarse a una amplia gama de problemas relacionados con *machine learning*, incluyendo tareas como la clasificación o la regresión. El *stacking* puede integrar diversas arquitecturas de modelos base, como árboles de decisión, máquinas de vectores de soporte (SVM) o redes neuronales. [9]
- **Interpretación potencial:** aunque la interpretación de un ensamble puede ser más compleja que la de un modelo único, el *stacking* puede ofrecer una visión sobre la importancia relativa de cada modelo base en la predicción final. Al estudiar los coeficientes o contribuciones de cada modelo base en el meta-modelo, se puede obtener una mejor comprensión de su relevancia y cómo sus predicciones son ponderadas. [9]

En conclusión, el *stacking* representa una estrategia avanzada dentro de las técnicas de ensamble, capaz de combinar la fortaleza de múltiples modelos base mediante un meta-modelo que aprende a integrar sus predicciones de manera óptima. Su éxito radica en la diversidad y complementariedad de los modelos utilizados, así como en una cuidadosa preparación y

evaluación del proceso de entrenamiento. Aunque puede requerir un mayor esfuerzo computacional y una atención especial para evitar problemas como el sobreajuste o la fuga de datos, el *stacking* suele traducirse en mejoras significativas en la precisión y robustez de las predicciones, superando tanto a los modelos individuales como a otros métodos de ensamble más simples. Por estas razones, el *stacking* se ha consolidado como una herramienta fundamental en la práctica moderna del aprendizaje automático.

2.4. Técnicas de ensamble, bagging

Como se ha abordado previamente, las técnicas de ensamble combinan múltiples modelos de aprendizaje para mejorar la robustez y el rendimiento predictivo. Dentro de estas, el *bagging* (*bootstrap aggregating*) emerge como una de las estrategias fundamentales y más utilizadas.

2.4.1. ¿Qué es el bagging?

El *bagging* es un protocolo de aprendizaje de ensamble que implica el entrenamiento de múltiples modelos de forma independiente y en paralelo [10]. Esta técnica de ensamble es conocida por su robustez frente a los ataques de “envenenamiento de datos”². Su característica distintiva es que cada modelo individual, conocido como “sub-clasificador” o “aprendiz débil”, se entrena en un subconjunto de datos seleccionado con reemplazo (*bootstrap sampling*) del conjunto de entrenamiento original³. Una vez que todos los modelos han sido entrenados, sus predicciones se combinan para determinar la salida final. En problemas de clasificación, esto se logra típicamente mediante el promedio de las probabilidades de clase⁴ o un mecanismo de votación por mayoría. [10, 13, 14]

El ejemplo más popular y ampliamente estudiado de un método de *bagging* es *random forest*, donde los aprendices combinados son árboles de decisión, concretamente, árboles CART. Se tratan de una colección de árboles de decisión donde cada árbol se entrena en un subconjunto aleatorio de muestras y características del conjunto de entrenamiento. La aleatoriedad en los *random forest* no solo se introduce mediante la selección de muestras con reemplazo (*bootstrap sampling*), sino también al considerar, en las divisiones de cada árbol, únicamente un grupo aleatorio de variables del modelo. De este modo, cada árbol toma decisiones basadas en diferentes combinaciones de variables, lo que incrementa la diversidad entre los árboles y refuerza la capacidad del ensamble para poder ampliar al resto de casos. [10, 14].

2.4.2. Ventajas del bagging frente a otras técnicas de ensamble

La popularidad del *bagging*, y en particular de *random forest*, se debe a sus ventajas más relevantes:

- **Reducción del sobreajuste y mejora de la estabilidad:** los métodos de árbol de decisión individuales pueden ser propensos al sobreajuste y ser inestables, lo que significa

²**Envenenamiento de datos** (*Data poisoning*): “es un tipo de ciberataque donde un adversario compromete intencionalmente un conjunto de datos de entrenamiento utilizado por un modelo de IA o *machine learning* para influir o manipular la operativa de dicho modelo.” [11]

³**Técnica del reemplazo:** técnica que se basa en que cada vez que se elige un ejemplo al azar, se devuelve al conjunto original para poder ser elegido de nuevo. Así, algunos ejemplos pueden repetirse y otros no aparecer en el subconjunto.

⁴**Promedio de las probabilidades de clase:** obtiene la media de las probabilidades que cada modelo da para cada clase, y elige la clase con mayor promedio [12].

que cualquier pequeño cambio en el conjunto de datos puede alterar significativamente sus predicciones. Los métodos de ensamble, como el *bagging*, reducen dicho sobreajuste y dan como resultado un modelo más estable al combinar las predicciones de múltiples modelos. [13, 14]

- **Manejo eficaz de conjuntos de datos tabulares:** los *random forests*, donde los clasificadores combinados son árboles de decisión (CART), han demostrado su buen desempeño en el manejo de conjuntos de datos tabulares⁵. Mejoran significativamente el rendimiento de los árboles individuales y mitigan los problemas asociados a ellos. [13]
- **Rendimiento comparable o superior al boosting en clasificación binaria:** generalmente se considera que los modelos de *boosting* tienen un rendimiento superior, al menos a nivel experimental. Sin embargo, los *enhanced random forest* (una extensión de los *random forest*) han demostrado ser comparables, o incluso superiores, en problemas de clasificación binaria, especialmente cuando se ejecutan con hiperparámetros predeterminados. Esto indica la robustez del enfoque de *bagging* mejorado, incluso sin necesitar de una extensa sintonización de hiperparámetros. [13, 14]
- **Robustez certificada contra ataques de envenenamiento de datos:** el *bagging* es un protocolo de aprendizaje de ensamble que destaca por su capacidad para aumentar la robustez frente a ataques de envenenamiento de datos⁶, gracias a su mecanismo de votación mayoritaria. A diferencia de otras defensas que se limitan a algoritmos o propiedades específicas, el *bagging* es versátil y compatible con múltiples arquitecturas. Destaca especialmente la variante *hash bagging*, que incrementa la protección casi sin coste adicional al sustituir el submuestreo aleatorio por uno determinista basado en *hash*. Esta modificación limita la influencia de cada muestra envenenada y mantiene una precisión comparable, además de no imponer restricciones adicionales sobre los hiperparámetros ni sobre el tipo de modelos utilizados, lo que permite su aplicación en una amplia variedad de escenarios. [10]
- **Funcionalidades adicionales para limpieza de datos:** el concepto de importancia de la muestra incorporado en los *enhanced random forests* permite limpiar el conjunto de entrenamiento, descartando las muestras y características menos importantes sin sacrificar el rendimiento. En promedio, se puede reducir el 26.55 % de las características y el 11.01 % de las muestras con una mínima disminución del rendimiento (0.2 %), logrando una reducción en el tiempo de cómputo y en los requisitos de memoria. [13]

En resumen, el *bagging* y sus extensiones, como los *enhanced random forests*, ofrecen una combinación única de robustez y rendimiento, que lo hace preferible en muchos escenarios de aprendizaje automático. Su capacidad para manejar conjuntos de datos tabulares, reducir el sobreajuste y proporcionar una defensa efectiva contra ataques de envenenamiento de datos mediante *hash bagging* lo posiciona como una herramienta muy valiosa en la ciencia del dato y el aprendizaje automático.

⁵**Datos tabulares:** datos organizados en forma de tabla. Este formato es el más común en bases de datos y hojas de cálculo, y es típico en muchos problemas de *machine learning* clásico, como clasificación o regresión.

⁶**Envenenamiento de datos** (*Data poisoning*): “es un tipo de ciberataque donde un adversario compromete intencionalmente un conjunto de datos de entrenamiento utilizado por un modelo de IA o *machine learning* para influir o manipular la operativa de dicho modelo.” [11]

2.5. Mejoras de los LLM

Para mejorar el rendimiento de un LLM, generalmente se considera el escalado del modelo en términos de su tamaño (número de parámetros) y la cantidad de datos de entrenamiento (número de tokens). Cuantos más datos se usen, más grandes sean los modelos y mayor sea la capacidad de cómputo, el rendimiento en los LLM será superior en la mayoría de los casos. Las “leyes de escalamiento” indican que existe una relación predecible entre el número de parámetros de un modelo de lenguaje y su rendimiento. [15, 16, 17]

Pasos para mejorar el rendimiento de un LLM

El proceso para mejorar el rendimiento de un LLM es conceptualmente sencillo, pero complejo en la práctica. Los pasos a seguir son los siguientes:

1. **Escalar el pre-entrenamiento:** se aumenta el número de parámetros del modelo y la cantidad de tokens de entrenamiento. Esto implica alimentar el modelo con cantidades colosales de texto cuidadosamente procesado para que aprenda patrones fundamentales del lenguaje. [15, 16]
2. **Ajuste fino (*fine-tuning*) y alineación:** Después del pre-entrenamiento, el modelo se optimiza usando conjuntos de datos más pequeños, pero de mayor calidad, refinados mediante técnicas como el ajuste fino supervisado. Alternativamente, se puede realizar un aprendizaje por refuerzo con retroalimentación humana para alinear su comportamiento con las preferencias humanas, haciéndolo más útil y seguro. [15, 16]
3. **Realizar el entrenamiento:** se entrena el modelo con los datos de entrenamiento, ajustando sus parámetros para minimizar la pérdida y mejorar su capacidad de generalización. Este proceso requiere una gran cantidad de recursos computacionales y tiempo. [15, 16, 18]

Escalamiento de LLM

Recientemente, debido a los resultados obtenidos por el proyecto “Chinchilla” [17], se ha modificado la forma de entender el escalamiento de los LLM. Anteriormente, se creía que el tamaño de un modelo debía de crecer más rápido que el del conjunto de entrenamiento. Sin embargo, este cambio de paradigma ha llevado a comprender que, para un entrenamiento computacionalmente óptimo, el tamaño del modelo y el número de tokens de entrenamiento deben de escalarse en una proporción lo más cercana posible a 1:1. [15, 17]

La conclusión del estudio de “Chinchilla” fue que la relación óptima es alrededor de 20 tokens por parámetro, lo que permite minimizar las pérdidas en el entrenamiento. Sin embargo, una proporción más común en la práctica es de 150 tokens por parámetro, lo que favorece modelos más pequeños, cuya implementación es más reducida en coste, especialmente cuando se debe hacer en repetidas ocasiones. [15, 17]

Esta relación entre el tamaño del modelo y la cantidad de datos de entrenamiento también influye directamente en los recursos y el tiempo necesarios para completar el entrenamiento. En general, cuanto más grande es el modelo en términos de parámetros, más tiempo tarda en entrenarse en una configuración dada. Sin embargo, esta relación no es siempre lineal: pasar de 1.3 mil millones a 6 mil millones de parámetros (un aumento de 4.6 veces en datos) resultó en un aumento de 20 veces en el tiempo de entrenamiento. De 6.7 mil millones a 13 mil millones

de parámetros (un aumento de 1.9 veces) resultó en un aumento de 3.5 veces en el tiempo de entrenamiento. [19]

Coste del entrenamiento de LLM

El coste del entrenamiento de un LLM es extremadamente elevado, y, aun así, se duplica cada 9 meses aproximadamente. Este coste no solo incluye el entrenamiento en sí, sino también los gastos asociados a hardware, consumo de energía y salarios del personal de I+D. [18]

Ejemplo de LLaMA 3 400B:

- Este modelo, con 405 mil millones ($4,05 \times 10^{11}$) de parámetros y entrenado en 15.6 billones ($1,56 \times 10^{13}$) de tokens, tuvo un coste computacional estimado en aproximadamente $3,8 \times 10^{25}$ FLOPS. [15]
- En cuanto a hardware, hizo falta el funcionamiento de 16.000 GPU H100 durante aproximadamente 70 días, lo que suma entre 26 y 30 millones de horas de uso. Su coste se estima en más de \$50 millones, suponiendo una tasa de \$2 por hora. [15]
- El coste total de entrenamiento para un modelo similar a este, incluyendo costes asociados (como los salarios de los empleados), puede superar los \$75 millones. [15]

En resumen, escalar el número de parámetros y la cantidad de datos sigue siendo una estrategia eficaz para mejorar el rendimiento de los LLM, pero implica costes y tiempos de entrenamiento muy elevados. Por otro lado, la tendencia actual apunta a buscar un equilibrio más eficiente entre el tamaño del modelo y la cantidad de datos de entrenamiento, junto con la optimización de la eficiencia y la consideración de los enormes costes asociados a la creación y el uso de datos.

2.6. Selección de modelos

Para la aplicación de las técnicas de ensamble, es crucial seleccionar modelos base que sean capaces de abordar las tareas de razonamiento complejo y que presenten un rendimiento destacado en los *benchmarks* relevantes. A continuación, se detallan los modelos seleccionados para este estudio:

- **Claude 3.5 Sonnet:** modelo de lenguaje desarrollado por Anthropic, destaca por su rapidez, capacidad de razonamiento, habilidades de codificación y visión, superando a modelos anteriores y competidores en diversas evaluaciones. Ha sido diseñado con un fuerte compromiso hacia la seguridad y la privacidad, sometiéndose a rigurosas pruebas y no utilizando datos de usuarios para su entrenamiento sin permiso explícito. [20]
- **DeepSeek Chat v2.5:** modelo de código abierto que integra procesamiento de lenguaje natural avanzado y potentes capacidades de generación y análisis de código, ofreciendo una solución unificada para tareas conversacionales y de programación. [21]
- **Gemini 2.0 Flash Experimental:** modelo de Google DeepMind que muestra explícitamente sus razonamientos para resolver problemas complejos y ha mejorado su rendimiento en matemáticas, ciencia y procesamiento multimodal. Ofrece una ventana de contexto de más de 1 millón de tokens, capacidad de razonar (experimental) y conocimiento hasta agosto de 2024. [22, 23]

- **GPT-4o (2024-08-06):** una versión avanzada del modelo GPT de OpenAI, que introduce la función de *JSON Structured Outputs*, permitiendo a los desarrolladores definir el formato de salida mediante un esquema JSON, lo que garantiza datos estructurados y reduce la necesidad de post-procesamiento. Ofrece mejoras significativas en eficiencia de costos, dando lugar a un amplio uso en aplicaciones comerciales. [24]
- **Meta Llama 3.1 70B Instruct:** parte de la colección de Meta Llama 3.1. Es un modelo que destaca por su capacidad avanzada de *tool calling*, permitiendo integrar y coordinar llamadas a herramientas externas (como búsquedas web o cálculos matemáticos) y ejecutar código Python de forma estructurada y controlada dentro de conversaciones multi-turno. [25]
- **Mixtral 8x7B v0.1:** modelo de lenguaje de código abierto basado en una SMoE, optimizado para seguir instrucciones, generar texto creativo y realizar tareas de programación. Destaca por su alto rendimiento, siendo el mejor modelo de código abierto en el *benchmark* MT-Bench. [26]
- **Qwen 1.5 110B:** modelo de lenguaje de gran tamaño desarrollado por Alibaba Cloud, basado en arquitectura transformer y preentrenado con más de 2.2 billones de tokens en múltiples idiomas, textos, código y matemáticas. Destaca por su alto rendimiento en tareas de razonamiento, código y matemáticas, soporte multilingüe, contexto de hasta 32K tokens y bajo coste de despliegue, superando a otros modelos *open source* de tamaño similar en varios *benchmarks*. [27]

Justificación de la selección

La selección de estos modelos base responde a criterios de popularidad, rendimiento demostrado en tareas de procesamiento del lenguaje natural y diversidad arquitectónica. Además, están centrados en tareas generalistas y desempeñan buenos resultados en los *benchmarks* de MMLU-Pro[28], siendo una consecuencia de su capacidad general y no un intento de destacar. Esta variedad permite que cada modelo aporte perspectivas y capacidades complementarias, lo que resulta esencial para el éxito del método de *stacking*. La combinación de modelos de diferentes arquitecturas y enfoques permite capturar una mayor diversidad de patrones y relaciones entre los datos, incrementando la capacidad del sistema para razonar y responder con precisión a preguntas tipo test.

3. Desarrollo y resultados

En este capítulo se describe el desarrollo del sistema de mejora del rendimiento de los LLM mediante técnicas de ensamble. Se detallan las etapas clave del proceso, desde la selección de modelos hasta la implementación del meta-modelo y su evaluación.

3.1. Selección de modelos

La selección de modelos base es un paso crucial en el proceso de ensamble, ya que la diversidad y complementariedad de los modelos elegidos pueden influir significativamente en el rendimiento del sistema final. En este trabajo, se han seleccionado varios LLM populares y bien establecidos, cada uno con características únicas que aportan diferentes fortalezas al conjunto.

3.1.1. Modelos base

Para el desarrollo del sistema se ha empleado una amplia variedad de modelos, abarcando tanto alternativas de código abierto como modelos comerciales. Esta diversidad permite disponer de un amplio espectro de capacidades y enfoques en el procesamiento del lenguaje natural. Los modelos utilizados son los adquiridos del Github de MMLU-Pro[29], concretamente los siguientes:

- **ARX-0314, ARX-3:** de una discusión de reddit [30], se pueden encontrar enlaces a, entre otros, una publicación en X de Applied General Intelligence, que reclama su autoría [31]. Sin embargo, no son fuentes fiables como para afirmarlo con certeza.
- **C4AI Command-R v0.1:** modelo creado por Cohere y Cohere Labs, con 35B de parámetros y una longitud de contexto de 128K [32].
- **Claude 3.5 Haiku (2024-10-22), Claude 3.5 Sonnet (2024-10-22), Claude 3.5 Sonnet, Sonnet, Sonnet 3.5:** modelos de Anthropic, con ventanas de contexto de 200K [33, 34]. Se estima un tamaño de 175B de parámetros para el modelo Claude 3.5 Sonnet [35].
- **DeepSeek LLM, DeepSeek Chat v2.5, DeepSeek Coder V2:** modelos *open source* de DeepSeek-AI, con distintos tamaños y ventanas de contexto. El primero cuenta con 7B y 67B de parámetros, y una ventana de contexto de 4096 tokens [36]; el segundo, con 236B y 128K [37]; y el tercero, con 16B y 236B de parámetros y una ventana de contexto de 128K tokens [38].
- **Flash:** modelo del Grupo Alibaba en conjunto con FSA-Lab@USYD. Cuenta con 66B y 175B de parámetros [39].

- **Gemini 1.5 Flash 002, Gemini 1.5 Pro 002, Gemini 2.0 Flash Experimental:** modelos de Google DeepMind, con ventanas de contexto de 1, 2 y 1 millón de tokens respectivamente [40, 41].
- **Gemma 7B:** modelo de Google con 8.54B de parámetros y una ventana de contexto de 8K tokens [42, 43].
- **GPT-4o Mini, GPT-4o (2024-05-13), GPT-4o (2024-08-06):** modelos de OpenAI, con ventanas de contexto de 128K tokens [44, 45, 46]. En cuanto a tamaño, el GPT-4o Mini tiene un tamaño similar a Llama 3 8b, Claude Haiku and Gemini 1.5 Flash, unos 8B de parámetros [47].
- **iAsk Pro:** desarrollado por iAsk.AI [48].
- **Jamba 1.5 Large:** desarrollado por AI21 Labs. Se trata de un modelo de 398B¹ de parámetros y una ventana de contexto de 256K tokens [49].
- **Llama 2 13B HF, Llama 2 70B HF, Llama 2 7B HF, Meta Llama 3 70B, Meta Llama 3 70B Instruct, Meta Llama 3 8B, Meta Llama 3 8B Instruct, Meta Llama 3.1 70B, Meta Llama 3.1 70B Instruct, Meta Llama 3.1 8B, Meta Llama 3.1 8B Instruct:** modelos de Meta, que tienen entre 7B y 70B de parámetros, definidos en el propio nombre del modelo (13B, 7B...) [50]. En cuanto a la ventana de contexto, Llama 2 cuenta con 4096 tokens [51], frente a los 8192 de Llama 3 [52] y los 128K de Llama 3.1 [53].
- **Mathstral 7B:** modelo de Mistral AI, con 7B de parámetros y una ventana de contexto de 256K tokens [54].
- **Mistral 7B Instruct v0.1, Mistral 7B Instruct v0.2, Mistral 7B v0.1, Mistral 7B v0.2 HF, Mixtral 8x7B Instruct v0.1, Mixtral 8x7B v0.1:** modelos *open source* de Mistral AI, con 7B de parámetros, salvo los Mixtral 8x7B (utilizan MoE, aumentando a 56B, con 12B activos) y una ventana de contexto de 32K [55].
- **Opus:** modelo de Anthropic con una ventana de contexto de 200K tokens y 1M para algunos casos [56].
- **Phi-3 Mini 4K Instruct:** modelo de Microsoft, con 3.8B de parámetros y 4K y 128K tokens de ventana de contexto [57].
- **Qwen 1.5 110B, Qwen 1.5 14B Chat, Qwen 1.5 72B Chat, Qwen 1.5 7B Chat:** modelos *open source* de Alibaba Cloud, con 32K tokens de ventana de contexto y tamaño presente en el nombre de cada uno [58].
- **Yi 34B, Yi 6B, Yi 6B Chat:** modelos *open source* creados por 01 AI, con 6B y 34B de parámetros con una ventana de contexto de 200K tokens [59].

Nota: El tamaño de los modelos se mide en B (billones de escala corta: miles de millones [60]) de parámetros.

3.2. Fase 1

El desarrollo inicial del sistema para mejorar el rendimiento de los LLM mediante técnicas de ensamble se ha abordado de forma programática, siguiendo una metodología similar a la de

¹se ha supuesto un error de escritura de Nvidia, un total de 94B de parámetros y 398B activos”, debería ser al revés [49].

un análisis de datos. Para ello, se han empleado todos los modelos disponibles en el Github de MMLU-Pro[29] y se han realizado las siguientes etapas, en el orden descrito a continuación:

Nota: N se refiere al número de preguntas que se han seleccionado para el análisis. Además, se ha usado el *dataset* de MMLU-Pro[61] de 12102 filas, aunque, debido a la existencia de repeticiones en la tupla pregunta-opciones, se puede acceder a 11716 combinaciones únicas. Se han utilizado todas ellas para obtener los resultados más cercanos a la realidad, sin embargo, en el caso de querer realizar un análisis más rápido, se podría reducir N a, por ejemplo, 1000, obteniendo resultados similares.

1. **Búsqueda de los modelos más precisos:** se ha obtenido el porcentaje de acierto de cada modelo. Para ello, se han elegido aleatoriamente N filas, de donde se extraen la pregunta, sus opciones, la respuesta correcta y la categoría correspondiente. Estas tuplas se han buscado en los ficheros tipo *json* y se han comparado cada una de las respuestas recibidas con su respectiva solución correcta, calculando así el porcentaje de acierto. En la figura 3.1 se muestran los 47 modelos ordenados por el número de preguntas respondidas correctamente (azul), en caso de igualdad, tendría prioridad el que tuviera más respuestas vacías (verde), las incorrectas se muestran en blanco.

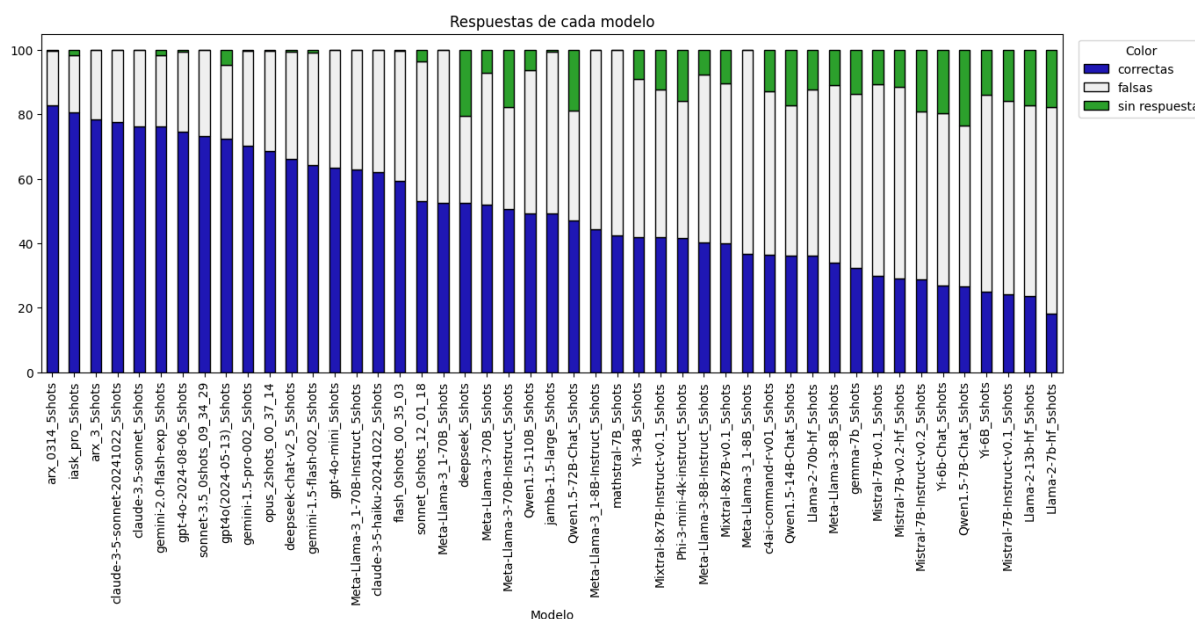


Figura 3.1: Modelos ordenados por porcentaje de acierto.

2. **Obtención de la respuesta más típica:** para cada una de las preguntas, se ha calculado la respuesta más frecuente entre todos los modelos. Para ello, se ha utilizado el método *mode()* de los *DataFrame* de la librería *pandas*, que devuelve el valor más común en una serie de datos. Esta informática se extiende en el paso 4.
3. **División según la categoría:** dado que el *dataset* de MMLU-Pro[61] incluye 14 categorías distintas (*biology, business, chemistry, computer science, economics, engineering, health, history, law, math, philosophy, physics, psychology* y *other*²), es esperable que los modelos presenten rendimientos desiguales en cada una ellas. Por lo tanto, se ha calculado el porcentaje de acierto de cada modelo en cada categoría, cuyos resultados se muestran en la figura 3.3. Dicho mapa de calor es una “tabla” ordenada por orden alfabético de los distintos modelos y categorías que muestra el porcentaje de acierto de

²En español son: biología, empresariales, química, informática, economía, ingeniería, medicina, historia, derecho, matemáticas, filosofía, física, psicología y otras, en ese orden

cada par, por ejemplo, Meta Llama 3 8B en economía (*economics*) tiene un 46.21 % de precisión en sus respuestas.

4. **Obtención de la respuesta más típica para los n mejores modelos:** para extender la información del paso 2, donde se ha obtenido la respuesta más típica, se ha hecho para todos los posibles conjuntos de n mejores modelos. Se ha comenzado seleccionando los 3 mejores y obteniendo la respuesta más frecuente entre ellos. Posteriormente, se ha repetido el proceso incrementando progresivamente el número de modelos considerados (4, 5, 6, etc.) hasta incluir el conjunto completo. Los resultados se presentan en la figura 3.2, la cual sigue la misma estructura que la 3.1, añadiendo los nuevos modelos generados, los cuales aparecen en negrita y con el nombre “mas_tipica_topX”, donde X representa el número de modelos utilizados para cada caso.

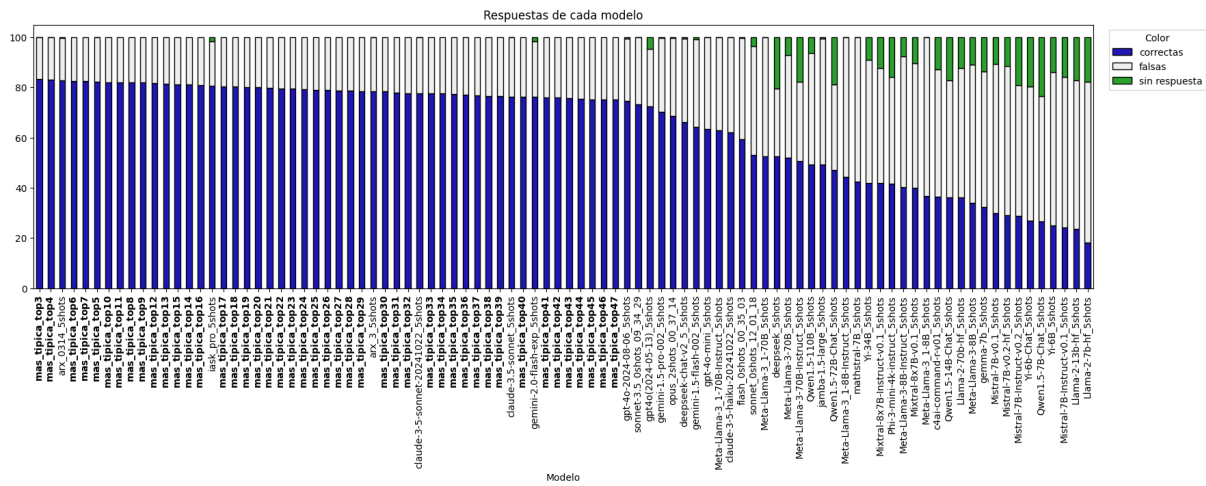


Figura 3.2: Respuesta más típica para los n mejores modelos, para 11716 filas.

5. **Obtención de la respuesta más típica y posteriormente división según la categoría:** se han juntado los pasos 4 y 3, en ese orden. Se han seleccionado los grupos de los n mejores modelos a nivel global. A partir de esta estructura, se ha creado una serie de subconjuntos de datos, uno para cada categoría, conteniendo las preguntas y respuestas correspondientes y las contestaciones de los distintos grupos. Finalmente, se ha calculado el porcentaje de acierto obtenido por cada conjunto en cada categoría.
6. **División según la categoría y posteriormente obtención de la respuesta más típica:** se ha realizado el proceso en el orden inverso al paso 5. Se han creado subconjuntos de datos, dividiendo en función de las categorías. Para cada uno de ellos se han organizado los modelos según su rendimiento y se han obtenido los grupos de los n mejores, que no tienen por qué coincidir con el orden global. Finalmente, se ha calculado el porcentaje de acierto obtenido por cada conjunto en cada categoría.
7. **Comparación de los pasos 5 y 6:** con el objetivo de analizar las diferencias que puede implicar cada enfoque, se ha llevado a cabo una comparación entre los pasos 5 y 6. En este análisis, se ha recogido únicamente el porcentaje de acierto más alto obtenido por cada método, así como el modelo, o grupo de ellos, que lo alcanza en cada categoría. En la tabla 3.1 se muestran dichos resultados, donde “T→C” se refiere al paso 5 y “C→T” al 6. Algunas celdas referentes a los modelos contienen varios elementos, esto indica que distintas combinaciones han alcanzado el mismo porcentaje de acierto.

Categoría	Mod. T→C	% T→C	Mod. C→T	% C→T
biology	mas_tipica_top7	91.35	mas_tipica_top7	91.35
business	mas_tipica_top3, arx_0314_5shots	86.82	arx_0314_5shots, mas_tipica_top3	86.82
chemistry	arx_0314_5shots	87.28	arx_0314_5shots	87.28
computer science	arx_0314_5shots, mas_tipica_top3, mas_tipica_top7, mas_tipica_top4	86.10	mas_tipica_top7	86.34
economics	mas_tipica_top4	86.49	mas_tipica_top3	87.09
engineering	mas_tipica_top3	72.55	mas_tipica_top5	72.76
health	mas_tipica_top7, mas_tipica_top10	77.46	mas_tipica_top8	77.94
history	mas_tipica_top6	76.90	arx_3_5shots, mas_tipica_top3	76.64
law	arx_0314_5shots	70.71	mas_tipica_top3	70.91
math	mas_tipica_top3	91.49	mas_tipica_top3	91.86
philosophy	mas_tipica_top3	81.96	mas_tipica_top3	81.96
physics	mas_tipica_top3	87.07	mas_tipica_top3	87.76
psychology	mas_tipica_top4	84.96	mas_tipica_top3	85.59
other	mas_tipica_top11	82.58	mas_tipica_top11	82.58

Tabla 3.1: Modelos usados según la categoría para obtener los mejores resultados

8. **Obtención del modelo más preciso usando todos los pasos anteriores:** el paso 7 indica qué modelos conviene emplear en cada categoría. No obstante, en caso de empate, se ha optado por seleccionar el conjunto más pequeño de los posibles. Por ejemplo, en la categoría *business* se ha decantado por el uso de “ARX-0314”, en vez del grupos de los 3 modelos más precisos, que en este caso, coinciden para ambos enfoques.

Mediante este procedimiento, eligiendo el método más eficaz para cada categoría, se alcanza un **83.79 %** de acierto. Este valor supera tanto a cualquiera de los modelos originales por sí solos, como a los conjuntos de n más precisos. La selección aplicada a cada categoría puede consultarse en la tabla 3.2, la cual muestra los grupos de modelos que se han de utilizar para obtener los resultados óptimos.

Categoría	Modelos
biology	arx_0314_5shots, gpt-4o-2024-08-06_5shots arx_3_5shots, claude-3.5-sonnet_5shots claude-3-5-sonnet-20241022_5shots gemini-2.0-flash-exp_5shots, iask_pro_5shots
business	arx_0314_5shots
chemistry	arx_0314_5shots
computer science	arx_0314_5shots
economics	claude-3-5-sonnet-20241022_5shots iask_pro_5shots, arx_0314_5shots
engineering	arx_0314_5shots, iask_pro_5shots, arx_3_5shots claude-3.5-sonnet_5shots
health	arx_0314_5shots, arx_3_5shots claude-3-5-sonnet-20241022_5shots gpt-4o-2024-08-06_5shots, iask_pro_5shots claude-3.5-sonnet_5shots, gemini-1.5-pro-002_5shots gemini-2.0-flash-exp_5shots
history	arx_3_5shots, claude-3.5-sonnet_5shots arx_0314_5shots, claude-3-5-sonnet-20241022_5shots gpt-4o-2024-08-06_5shots, iask_pro_5shots
law	arx_0314_5shots, iask_pro_5shots claude-3-5-sonnet-20241022_5shots
math	arx_0314_5shots, iask_pro_5shots, gemini-2.0-flash-exp_5shots
philosophy	arx_0314_5shots, iask_pro_5shots, arx_3_5shots
physics	arx_0314_5shots, iask_pro_5shots, gemini-2.0-flash-exp_5shots
psychology	claude-3-5-sonnet-20241022_5shots arx_0314_5shots, gemini-1.5-pro-002_5shots
other	arx_0314_5shots, arx_3_5shots claude-3-5-sonnet-20241022_5shots iask_pro_5shots, gpt-4o-2024-08-06_5shots claude-3.5-sonnet_5shots, gpt4o(2024-05-13)_5shots sonnet-3.5_0shots_09_34_29, gemini-2.0-flash-exp_5shots gemini-1.5-pro-002_5shots, opus_2shots_00_37_14

Tabla 3.2: Modelos usados según la categoría para obtener los mejores resultados. “T→C” y “C→T” especifican el orden de las ejecuciones: respuesta más típica (T) y categoría (C)

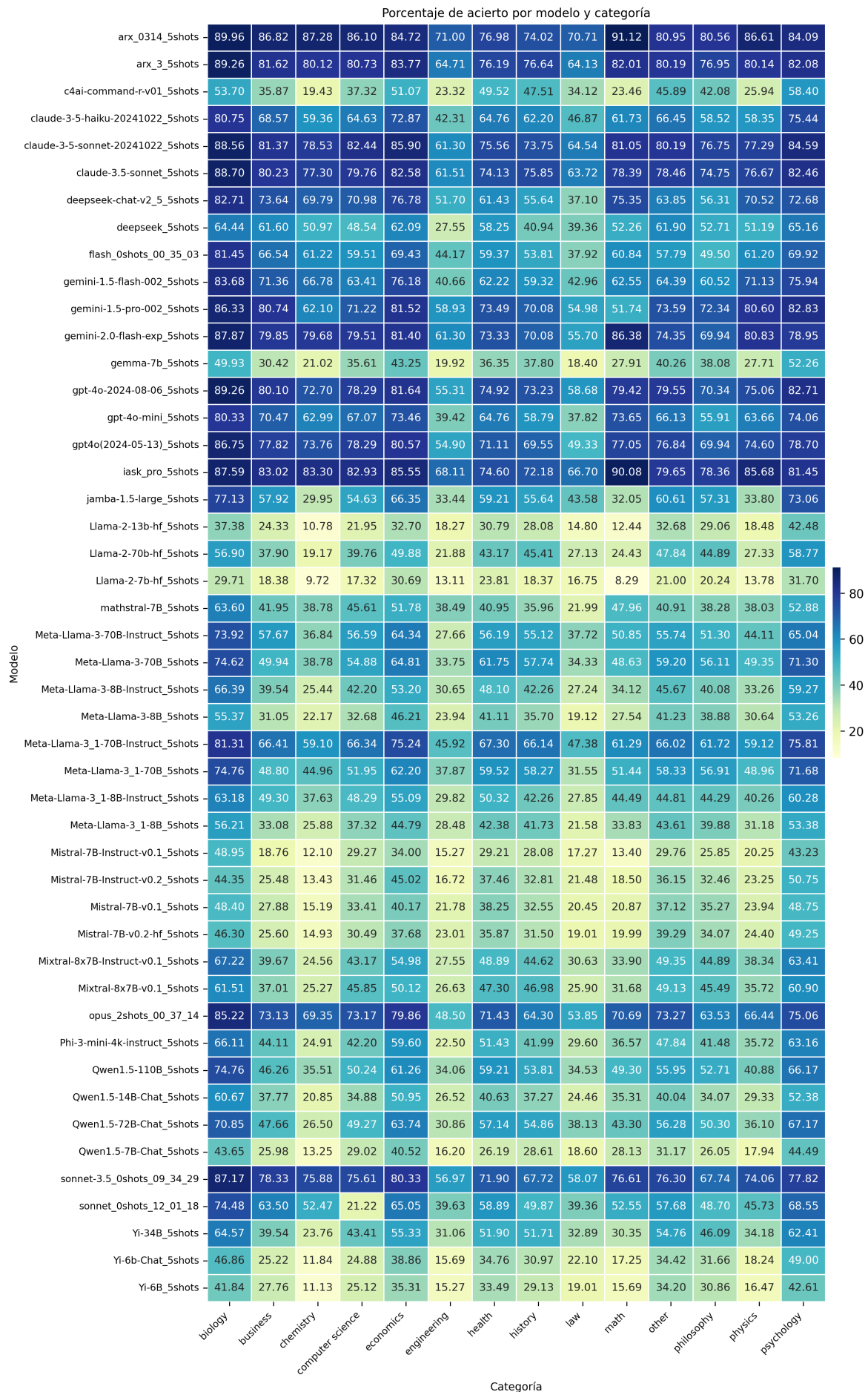


Figura 3.3: Porcentaje de acierto de los modelos según la categoría

3.2.1. Modelos seleccionados

El resultado obtenido en la Fase 1 es, con un 83.79 % de acierto, superior a todos los modelos base. Sin embargo, se debe interpretar con cautela, pues algunos contienen configuraciones muy específicas, ya que están centrados en mejorar el rendimiento en MMLU-Pro y son diferentes de los que usa el público general habitualmente. Por ello, se ha seleccionado un subconjunto más conocido y utilizado por el gran público.

De todos los modelos empleados inicialmente, se han seleccionado únicamente 7 para el desarrollo final del sistema. Estos han sido elegidos debido a su diversidad arquitectónica, su rendimiento destacado en *benchmarks* recientes y la disponibilidad en aplicaciones de chat de IA. Los modelos utilizados son: Claude 3.5 Sonnet, DeepSeek Chat v2.5, Gemini 2.0 Flash Experimental, GPT-4o (2024-08-06), Meta Llama 3.1 70B Instruct, Mixtral 8x7B v0.1 y Qwen 1.5 110B.

Se ha seguido el mismo proceso que en la Fase 1, limitado a los 7 elementos nombrados. Sin embargo, el rendimiento logrado es inferior, en parte por la inutilización de los 3 modelos más precisos a nivel global.

- En la Fase 1 se han usado los pasos 1, 2 y 4 para obtener los n modelos más precisos. En esta ocasión, se ha repetido el proceso usando los 7 modelos seleccionados, obteniendo los resultados mostrados en la figura 3.4, que muestra el porcentaje de acierto de cada uno de los modelos, tanto originales, como grupos generados. Cabe destacar que el porcentaje de acierto de los nuevos conjuntos es superior a cualquiera de los elementos definidos de base.

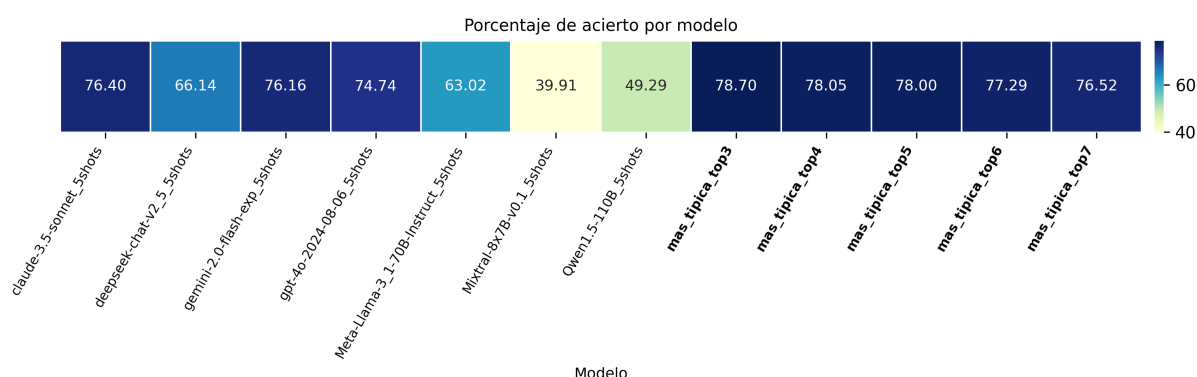


Figura 3.4: Porcentaje de acierto

- El paso 3 describe cómo se ha realizado la división de los modelos según la categoría. Nuevamente, “ $T \rightarrow C$ ” y “ $C \rightarrow T$ ” especifican el orden de las ejecuciones: respuesta más típica (T) y categoría (C); en el paso siguiente se ha ampliado esta información. Los resultados obtenidos están presentes en la figura 3.5, que muestra el porcentaje de acierto de cada uno de los modelos, considerando como tales también a los conjuntos con su respectivo método de optimización.

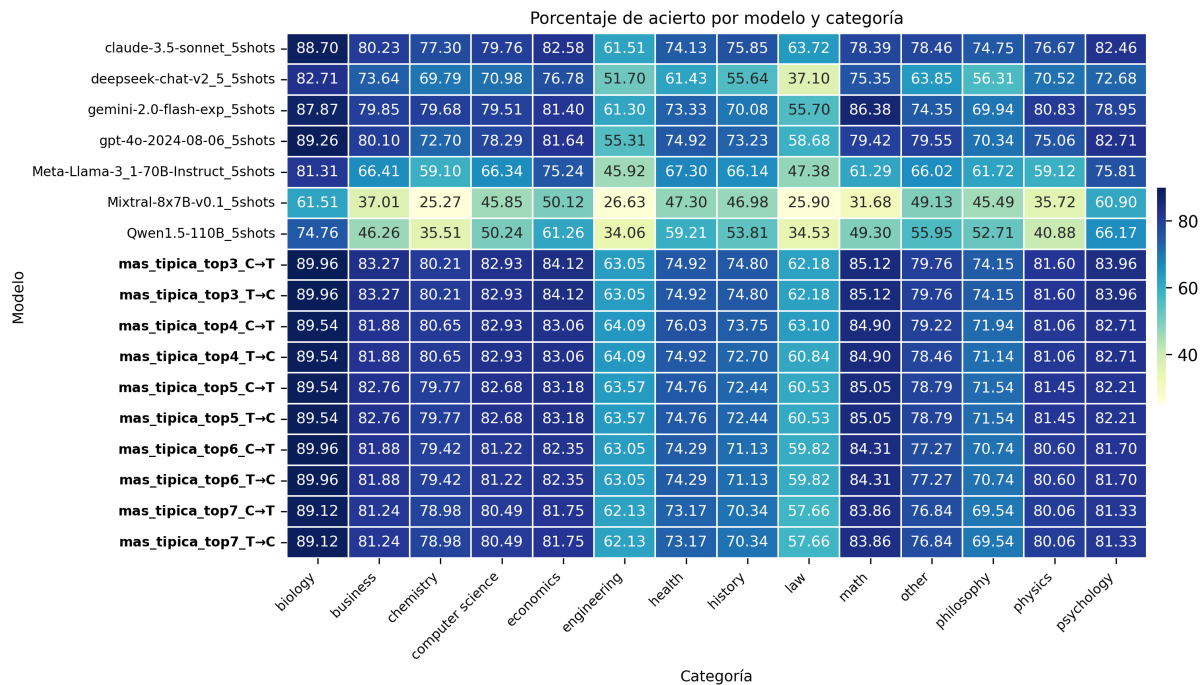


Figura 3.5: Porcentaje de acierto según el modelo/grupo y la categoría

- Por último, con la información obtenida en el punto anterior y siguiendo el método del paso 7 de la Fase 1, se desarrolla la tabla 3.3, la cual muestra el porcentaje de acierto de los mejores modelos y grupos para cada categoría y metodología según la categoría. En ella se denomina “T→C” al orden del paso 5 (respuestas más típicas primero y ordenar por categoría después) y “C→T” al del paso 6 (primero categoría y luego más típicas), referentes a la Fase 1 (sección 3.2). A partir de la tabla anterior, se puede realizar la 3.4, que discrimina la mejor alternativa, tanto en porcentaje de acierto como en número de modelos utilizados (cuantos menos mejor, pues se reduce el coste computacional). Este proceso logra un **79.22 %** de acierto, que, de nuevo, es superior a todos los modelos usados como base, aunque inferior al resultado obtenido de la Fase 1.

Categoría	Mod. T→C	% T→C	Mod. C→T	% C→T
biology	most_typical_top3, most_typical_top6	89.96	most_typical_top3, most_typical_top6	89.96
business	most_typical_top3	83.27	most_typical_top3	83.27
chemistry	most_typical_top4	80.65	most_typical_top4	80.65
computer science	most_typical_top3, most_typical_top4	82.93	most_typical_top3, most_typical_top4	82.93
economics	most_typical_top3	84.12	most_typical_top3	84.12
engineering	most_typical_top4	64.09	most_typical_top4	64.09
health	gpt-4o-2024-08-06, most_typical_top3, most_typical_top4	74.92	most_typical_top4	76.03
history	claude-3.5-sonnet	75.85	claude-3.5-sonnet	75.85
law	claude-3.5-sonnet	63.72	claude-3.5-sonnet	63.72
math	gemini-2.0-flash-exp	86.38	gemini-2.0-flash-exp	86.38
philosophy	claude-3.5-sonnet	74.75	claude-3.5-sonnet	74.75
physics	most_typical_top3	81.60	most_typical_top3	81.60
psychology	most_typical_top3	83.96	most_typical_top3	83.96
other	most_typical_top3	79.76	most_typical_top3	79.76

Tabla 3.3: Modelos usados según la categoría para obtener los mejores resultados. “T→C” y “C→T” especifican el orden de las ejecuciones: respuesta más típica (T) y categoría (C)

Categoría	Modelos
biology	claude-3.5-sonnet, gemini-2.0-flash-exp, gpt-4o-2024-08-06
business	claude-3.5-sonnet, gemini-2.0-flash-exp, gpt-4o-2024-08-06
chemistry	claude-3.5-sonnet, gemini-2.0-flash-exp, gpt-4o-2024-08-06, deepseek-chat-v2_5
computer science	claude-3.5-sonnet, gemini-2.0-flash-exp, gpt-4o-2024-08-06
economics	claude-3.5-sonnet, gemini-2.0-flash-exp, gpt-4o-2024-08-06
engineering	claude-3.5-sonnet, gemini-2.0-flash-exp, gpt-4o-2024-08-06, deepseek-chat-v2_5
health	gpt-4o-2024-08-06, claude-3.5-sonnet, gemini-2.0-flash-exp, Meta-Llama-3_1-70B-Instruct
history	claude-3.5-sonnet
law	claude-3.5-sonnet
math	gemini-2.0-flash-exp
other	claude-3.5-sonnet, gemini-2.0-flash-exp, gpt-4o-2024-08-06
philosophy	claude-3.5-sonnet
physics	claude-3.5-sonnet, gemini-2.0-flash-exp, gpt-4o-2024-08-06
psychology	claude-3.5-sonnet, gemini-2.0-flash-exp, gpt-4o-2024-08-06

Tabla 3.4: Modelos usados según la categoría para obtener los mejores resultados

Nota: se ha quitado el texto “_5shots” de los nombres de los modelos en las tablas 3.3 y 3.4, ya que no aporta información relevante y dificultaba la representación de las tablas.

3.3. Fase 2

Con el afán de mejorar el 79.22 % de precisión de la Fase 1 para los modelos seleccionados, se ha implementado una segunda fase de desarrollo. En esta ocasión, en vez de utilizar una metodología centrada en una selección manual, se ha optado por una estrategia automatizada que permite explorar un espacio de hiperparámetros³ más amplio. El proceso se ha llevado a cabo mediante la técnica de *One Hot Encoder* junto a 4 métodos de clasificación diferentes.

3.3.1. Random Forest Classifier

Metodología de desarrollo

El desarrollo de esta búsqueda se compone de varios pasos:

1. Dividir la información que se ha de usar para el entrenamiento en 2 *DataFrame*: las respuestas de todos los modelos por un lado (*X*), y las correctas por otro (*Y*).
2. Aplicar *One Hot Encoder*, concretamente *fit_transform*, al *DataFrame X* para convertirlo en el formato adecuado. Este método transforma variables categóricas (como letras) en columnas binarias (unos y ceros), de modo que cada valor posible se representa por una columna y solo la correspondiente al valor presente toma el valor uno, facilitando así que los modelos de aprendizaje automático puedan procesar estos datos. En el anexo A, en la sección One Hot Encoder, se describe su funcionamiento mediante un ejemplo.
3. Ejecutar el método *train_test_split*, que divide los datos (*X* e *Y*) en dos conjuntos, uno para entrenar el modelo y otro para evaluarlo, permitiendo así medir su rendimiento en datos no vistos. En el anexo A, en la sección Train Test Split, se describe su funcionamiento mediante un ejemplo.
4. Crear un modelo de clasificación utilizando *Random Forest Classifier*, entrenarlo con los subconjuntos de datos obtenidos en el paso anterior y evaluar su precisión. Este modelo se basa en la técnica de ensamblaje de árboles de decisión, que ayuda a mejorar la precisión y reducir sobreajustes.
5. Optimizar los hiperparámetros del modelo mediante técnicas de búsqueda, ajustando configuraciones clave para mejorar el rendimiento. Es decir, obtener la precisión máxima posible mediante la búsqueda de la combinación óptima.

Optimización de hiperparámetros

Con el fin de obtener el mejor resultado posible, se han optimizado los hiperparámetros mediante un proceso no secuencial, es decir que se deben ejecutar en conjunto. Sin embargo, los modelos que se han usado para este proceso sí que se han analizado de antemano. En la figura 3.6 se muestran los resultados de aplicar un *Random Forest Classifier* a todos los grupos, desde los 2 mejores hasta la utilización de todos los involucrados, donde se observa que el mejor resultado se obtiene con todos los modelos.

³Hiperparámetro: parámetro definido antes de comenzar el entrenamiento en el ámbito de *machine learning*. Estos valores pueden modificarse y tienen impacto directo en el rendimiento del modelo durante el aprendizaje[62].

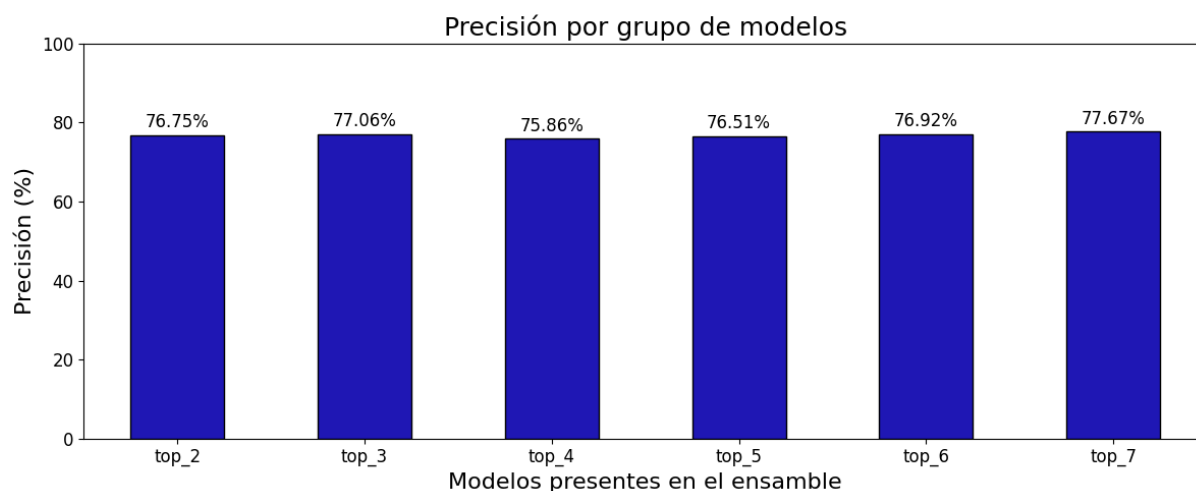


Figura 3.6: Precisión de los principales grupos de modelos

A continuación, se ha realizado una exhaustiva búsqueda de los hiperparámetros óptimos. Se ha logrado una mejora del porcentaje de acierto mediante la modificación de los siguientes:

- Para $n_estimators$, que es el número de árboles que se generan en el bosque aleatorio, se han comprobado valores que oscilan entre 0 y 700 con saltos de 50. Este hiperparámetro es crucial, ya que un número mayor de árboles puede mejorar la precisión del modelo, pero también aumenta el tiempo de entrenamiento y la complejidad del modelo.
- El hiperparámetro max_depth , que define la profundidad máxima de los árboles, es decir, el número máximo de nodos que se pueden crear en un árbol de decisión, se ha analizado en el rango de 1 a 20.

A pesar de que existen otros 16 hiperparámetros que podrían mejorar los resultados, la falta de capacidad de cómputo supone una barrera para su aplicación en este trabajo. La tabla 3.5 muestra los valores seleccionados y la precisión final lograda, que asciende a un 79.48 % de acierto.

Hiperparámetro	Valor óptimo
Modelos	Todos
$n_estimators$	650
max_depth	7
Precisión final (%)	79.48

Tabla 3.5: Mejor combinación de hiperparámetros y la precisión lograda.

3.3.2. K Neighbors Classifier

La búsqueda de los hiperparámetros óptimos para *K Neighbors Classifier* se ha llevado a cabo mediante una metodología similar a la empleada con *Random Forest Classifier*, aunque con algunas diferencias clave. Este clasificador se fundamenta en el algoritmo de los vecinos más cercanos, que predice la etiqueta de una muestra en función de las clases más frecuentes entre sus k vecinos más próximos en el conjunto de entrenamiento.

Se ha entrenado el sistema, habiendo creado previamente un modelo de clasificación utilizando *K Neighbors Classifier*, y se ha evaluado su precisión. El proceso de optimización de

hiperparámetros para mejorar el rendimiento del modelo comienza con la selección de los mejores grupos para el entrenamiento. En la figura 3.7 se muestran los resultados de aplicar un *K Neighbors Classifier* a los 6 grupos, donde el mejor porcentaje se obtiene con todos los modelos.

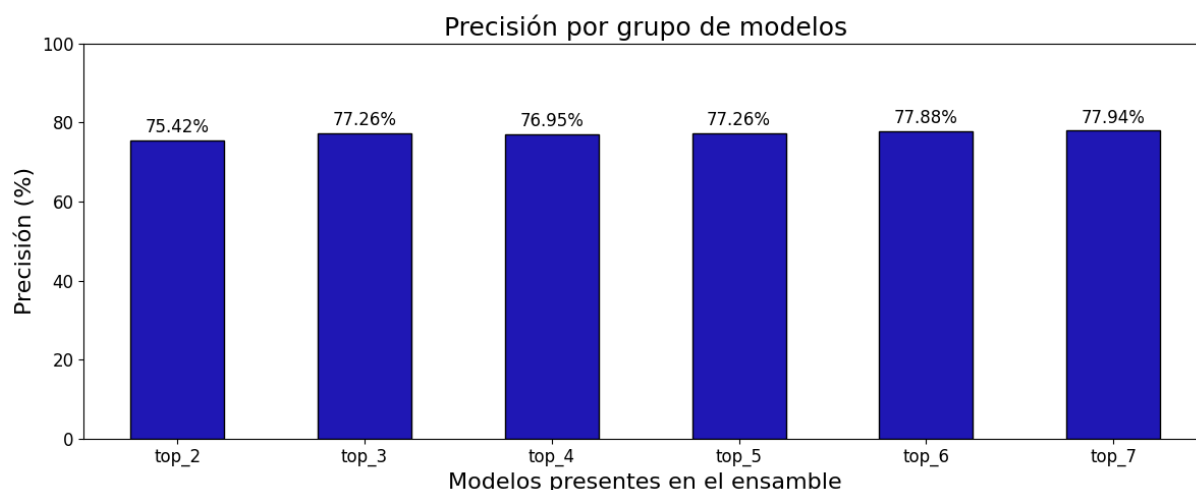


Figura 3.7: Precisión de los principales grupos de modelos

A continuación, se ha realizado una optimización de hiperparámetros. Se ha pretendido una mejora del porcentaje de acierto mediante la modificación de los siguientes:

- El hiperparámetro *n_neighbors*, que determina cuántos vecinos se tienen en cuenta al clasificar una muestra, se ha evaluado en el rango entre 10 y 20.
- Para *p*, que se define como la potencia de la métrica de Minkowski⁴, se ha buscado entre 1 y 4 con incrementos de 0.5.

A pesar de haber otros 6 hiperparámetros optimizables, la baja capacidad de cómputo accesible lo convierte en una tarea inabarcable para este trabajo. La tabla 3.6 muestra los valores seleccionados y la precisión final lograda, que asciende a un 78.46 %.

Hiperparámetro	Valor óptimo
Modelos	Todos
<i>n_neighbors</i>	12
<i>p</i>	2
Precisión final (%)	78.46

Tabla 3.6: Mejor combinación de hiperparámetros y la precisión lograda.

3.3.3. Support Vector Classifier

En esta ocasión se ha realizado un proceso similar, mas utilizando SVC (Support Vector Classifier), que se basa en el algoritmo de máquinas de soporte vectorial. Este modelo busca un hiperplano que separe las diferentes clases de datos de la mejor manera posible, maximizando el margen entre las clases.

⁴En informática podemos definir la distancia de Minkowski como una métrica que permite calcular la distancia entre puntos en espacios de cualquier dimensión, y engloba diferentes formas de medir distancias utilizadas tanto en matemáticas como en *machine learning*[63]

Al igual que en los apartados anteriores, se ha entrenado el sistema creando un modelo de clasificación con SVC y se ha evaluado su precisión. El proceso de optimización de hiperparámetros para mejorar el rendimiento del modelo empieza por seleccionar los mejores grupos para el entrenamiento. En la figura 3.7 se muestran los resultados de la aplicación de un SVC a todos los grupos utilizados, donde el mejor valor proviene de usar los 6 mejores modelos.

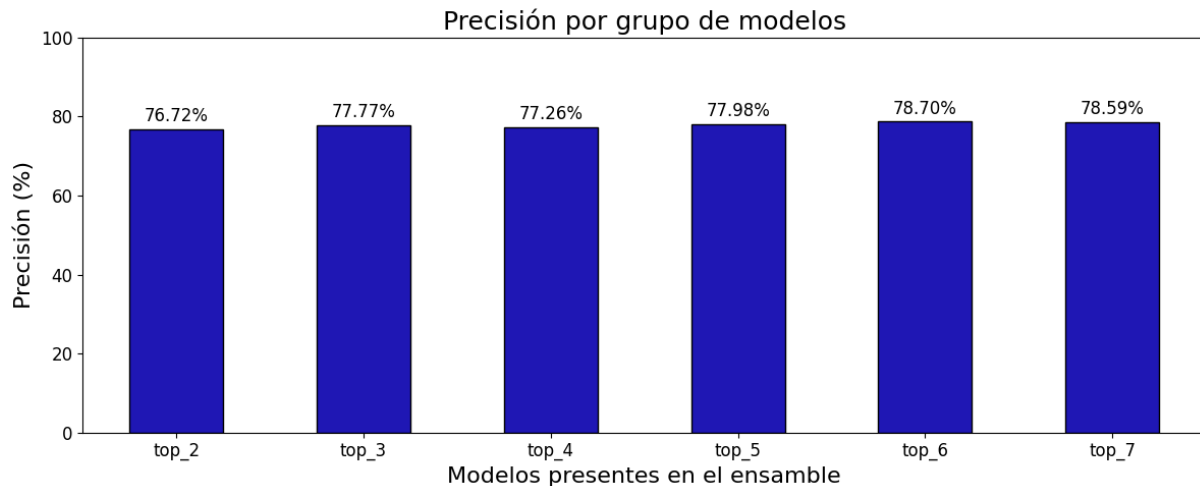


Figura 3.8: Precisión de los principales grupos de modelos

A continuación, se han optimizado los hiperparámetros. Se han mejorado resultados previos mediante la modificación de los siguientes:

- Los valores para comprobar C , que controla la compensación entre maximizar el margen y minimizar el error de clasificación, son los presentes en el rango de 0.1 a 1, con incrementos de 0.1
- El hiperparámetro γ , que define el coeficiente del kernel, tiene como opciones los valores *scale*, *auto* y números decimales. Se ha buscado entre los valores 0.1 y 1, con incrementos de 0.1, a parte de los 2 elementos textuales.
- En cuanto a *max_iter* se refiere, que define el número máximo de iteraciones para el algoritmo de optimización, se ha buscado entre 290 y 310.

El mejor resultado obtenido es un 79.38 % de precisión, utilizando un valor de 0.1 para C y γ , 300 iteraciones y utilizando los 6 mejores modelos. Habiendo fijado estos hiperparámetros a sus valores óptimos, se han analizado ciertos booleanos adicionales: *shrinking*, *probability*, *verbose* y *break_ties*, que logran cierta mejoría en el resultado, como se muestra en la figura 3.9. Esta imagen arroja el porcentaje de acierto de cada grupo de estos 4 elementos, cuyas etiquetas aparecen en el orden que muestra la imagen, y tienen los valores “verdadero” (T) y “falso” (F). Sin embargo, el valor final solo es dependiente de *break_ties*, que logra un 79.41 % de precisión cuando se activa.

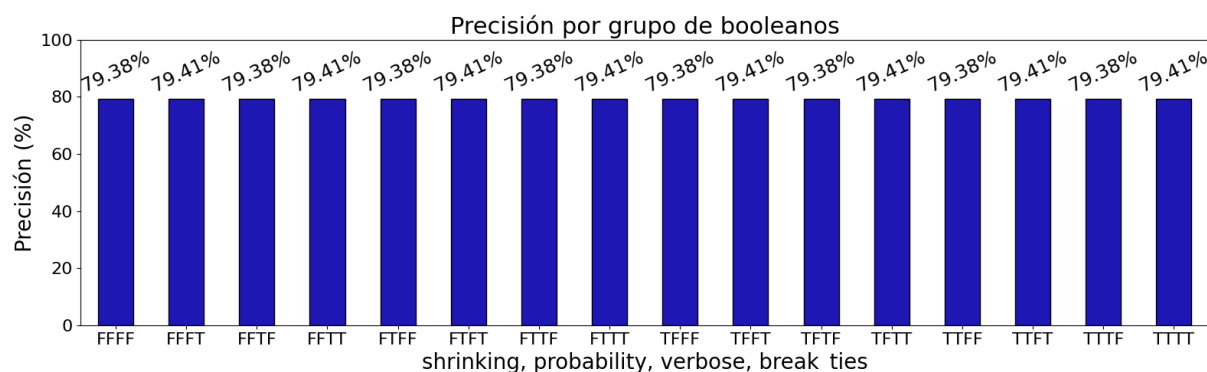


Figura 3.9: Precisión de los modelos con diferentes valores de los parámetros booleanos

La tabla 3.7 muestra los valores seleccionados y la precisión final lograda, que asciende a un 79.41 %. Sin embargo, la existencia de otros hiperparámetros que no se han podido explorar por falta de capacidad de cómputo, como *kernel* o *degree*, podrían superarlo.

Hiperparámetro	Valor óptimo
Modelos	Top 6
C	0.1
gamma	0.1
max_iter	300
break_ties	True
Precisión final (%)	79.41

Tabla 3.7: Mejor combinación de hiperparámetros y la precisión lograda.

3.3.4. Linear Regression

El último modelo analizado es *Linear Regression*, basado en el algoritmo de regresión lineal. Este método busca establecer una relación proporcional entre las variables de entrada y salida, ajustando una función matemática sencilla a los datos. Aunque suele emplearse principalmente en tareas de regresión, en este caso se ha adaptado para abordar un problema de clasificación.

Al igual que en los modelos anteriores, se ha entrenado el sistema creando un modelo de clasificación con *Linear Regression* y se ha evaluado su precisión. Para optimizar los hiperparámetros y mejorar el rendimiento del modelo, se comienza por obtener el mejor grupo de modelos. La figura 3.7 muestra los resultados de aplicar un *Linear Regression* a los 6 conjuntos presentes, donde el mejor valor proviene de unir los 6 mejores.

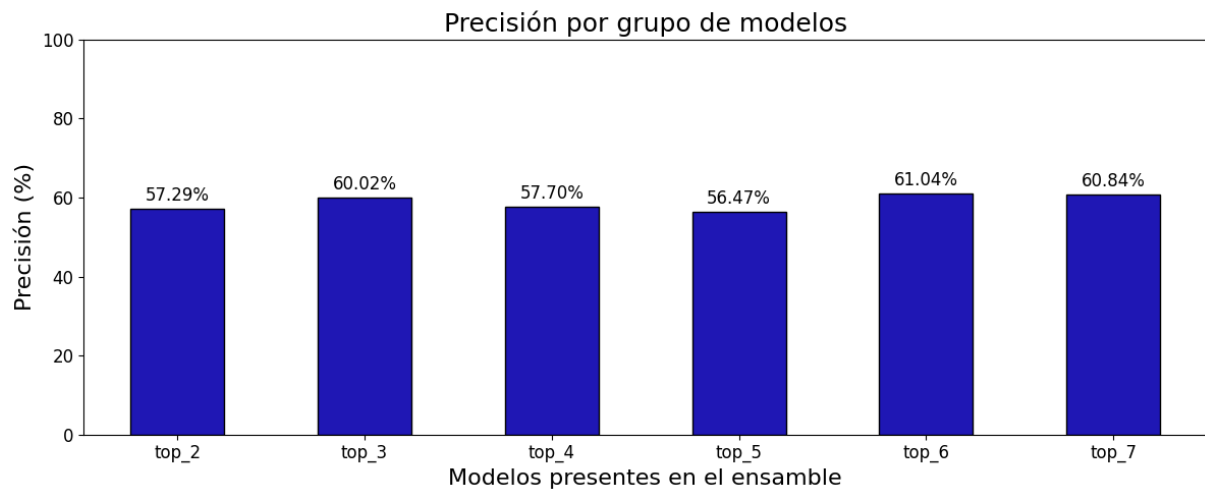


Figura 3.10: Precisión de los principales grupos de modelos

A continuación se realiza la optimización de hiperparámetros, que tiene únicamente 3 booleanos por sacar su mejor valor: *fit_intercept*, *copy_X* y *positive*. Estos hiperparámetros se han analizado en la figura 3.11, donde activar *positive* y desactivar *fit_intercept* a la vez produce el mejor porcentaje de acierto. La tabla 3.8 muestra los valores seleccionados y la precisión final lograda, que asciende a un 79.41 %.

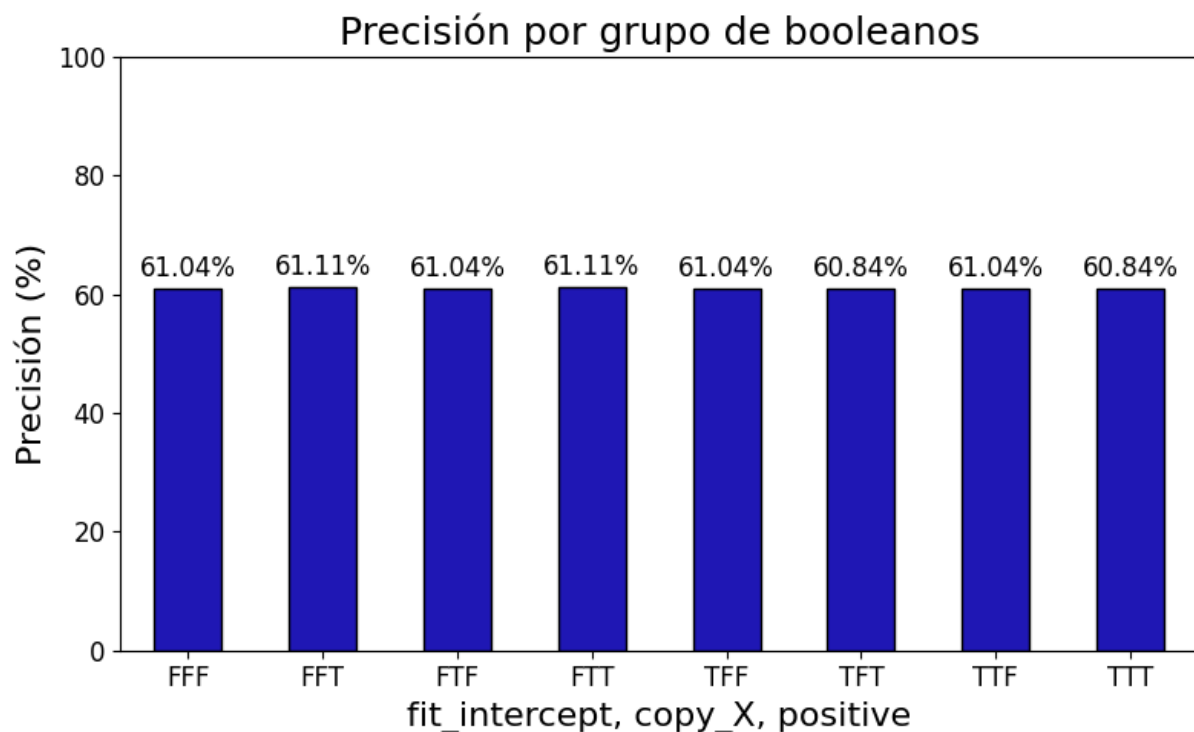


Figura 3.11: Precisión de los modelos con diferentes valores de los parámetros booleanos

Hiperparámetro	Valor óptimo
Modelos	Top 6
positive	True
fit_intercept	False
Precisión final (%)	61.11

Tabla 3.8: Mejor combinación de hiperparámetros y la precisión lograda.

Linear Regression ha demostrado ser inefectivo en este contexto, con una precisión significativamente inferior a los otros modelos analizados. Además, la inexistencia de otros hiperparámetros limita la capacidad de mejora del modelo.

3.3.5. Conclusiones de la Fase 2

Esta fase tenía como objetivo mejorar la precisión del sistema de clasificación mediante el uso de técnicas automatizadas de clasificación y optimización de hiperparámetros. Se han explorado diferentes modelos, incluyendo *Random Forest*, *K Neighbors*, *Support Vector Classifier* y *Linear Regression*. Cada uno de estos modelos ha sido evaluado en términos de precisión y se han ajustado los hiperparámetros para maximizar su rendimiento. Sin embargo, la precisión obtenida es, seguramente, inferior a la que se podría lograr mediante mayor capacidad de cómputo y tiempo de entrenamiento.

Los resultados obtenidos en esta fase se muestran en la tabla 3.9. La mejora ha sido modesta, logrando superar a aquellos de la Fase 1 en un 0.26 % mediante el uso de *Random Forest Classifier*. *K Neighbors Classifier* también han mostrado resultados competitivos, aunque inferiores, y *Support Vector Classifier* incluso supera a la Fase 1. Por otro lado, *Linear Regression* ha demostrado ser inefectivo en este contexto, con una precisión significativamente inferior.

Modelo	Precisión (%)
Metodología Programática (Fase 1)	79.22
Random Forest Classifier	79.48
K Neighbors Classifier	78.46
Support Vector Classifier	79.41
Linear Regression	61.11

Tabla 3.9: Resultados de la Fase 2

En resumen, la Fase 2 ha permitido explorar un espacio de hiperparámetros más amplio y ha proporcionado una visión más completa del rendimiento de los modelos seleccionados. Además de superar la precisión obtenida en la Fase 1, se ha logrado una comprensión más profunda de las capacidades y limitaciones de cada modelo en el contexto del sistema de clasificación propuesto.

3.4. Fase 3

Tras los resultados obtenidos en la Fase 2, donde la mejora de precisión fue modesta, se ha planteado una tercera fase orientada a explotar las capacidades complementarias de los distintos modelos. En este caso, en lugar del uso de todos los modelos de manera individual, se ha planteado aprovechar sus predicciones en conjuntos. Para ello se ha creado una red neuronal multicapa que actúa como meta-modelo, prediciendo, para cada ejemplo, cuál de los modelos base es más probable que acierte la respuesta correcta.

3.4.1. Metodología

Se ha implementado un sistema que utiliza como entrada las predicciones de los modelos seleccionados, genera nuevas variables basadas en el consenso de modelos y devuelve una respuesta fundamentada en su aprendizaje. Para lograrlo, se ha seguido el siguiente proceso:

1. Se han generado columnas de consenso para todas las combinaciones posibles de 2 o más modelos. En caso de un consenso del 100 % entre todos los modelos, se añade un 1 en dicha columna, si no un 0. Posteriormente, se ha creado un *DataFrame* con estas columnas.
2. Se han generado columnas de consenso entre las respuesta de los modelos y la respuesta real, indicando si el modelo ha acertado o no. Estas columnas se han añadido a un *array*.
3. Se han dividido los conjuntos de datos mediante *train_test_split* (mirar anexo A.2 para más detalles) y se han generado 2 nuevos grupos en cada caso: uno para entrenamiento y otro para prueba. Esto permite asegurar que las muestras no se solapen, evitando resultados anormalmente elevados.
4. Se ha creado un *StandardScaler* que convierte los *DataFrame* de las variables de entrada en *array* de datos normalizados, con media 0 y varianza 1. Esta medida ayuda al entrenamiento de la red neuronal, reduciendo el tiempo de convergencia y mejorando la precisión del modelo.
5. Se ha definido una red neuronal multicapa con la API *Sequential* de Keras, formada por varias capas densas con activación *ReLU*⁵ y *Dropout*. La salida es otra capa densa con activación *sigmoid*, que permite una probabilidad independiente para cada modelo base.
6. Se ha aplicado a la red neuronal anterior un optimizador *Adam* con una tasa de aprendizaje de 0.001 y función de pérdida *binary_crossentropy*.
7. Se ha entrenado la red neuronal con el conjunto de entrenamiento, utilizando 30 épocas, *batch_size* de 16 y un 20 % de los datos de entrenamiento como validación.
8. Se ha evaluado el modelo con el conjunto de prueba, obteniendo la precisión de la red neuronal.

Como resultado de este proceso, se ha obtenido un sistema capaz de analizar las predicciones individuales y los consensos entre los distintos modelos base para determinar, en cada caso, cuál de ellos tiene mayor probabilidad de acertar la respuesta correcta. Este enfoque permite aprovechar tanto la diversidad como las fortalezas de cada modelo, integrando la información

⁵La activación ReLU (Rectified Linear Unit) es una función utilizada en redes neuronales que transforma la entrada (x) en $\max(0, x)$. Es decir, devuelve 0 si el valor es negativo y el propio valor si es positivo.

de manera más eficiente que cualquier modelo por separado. Tras la evaluación con el conjunto de prueba, el sistema alcanzó una precisión del 78.87 %.

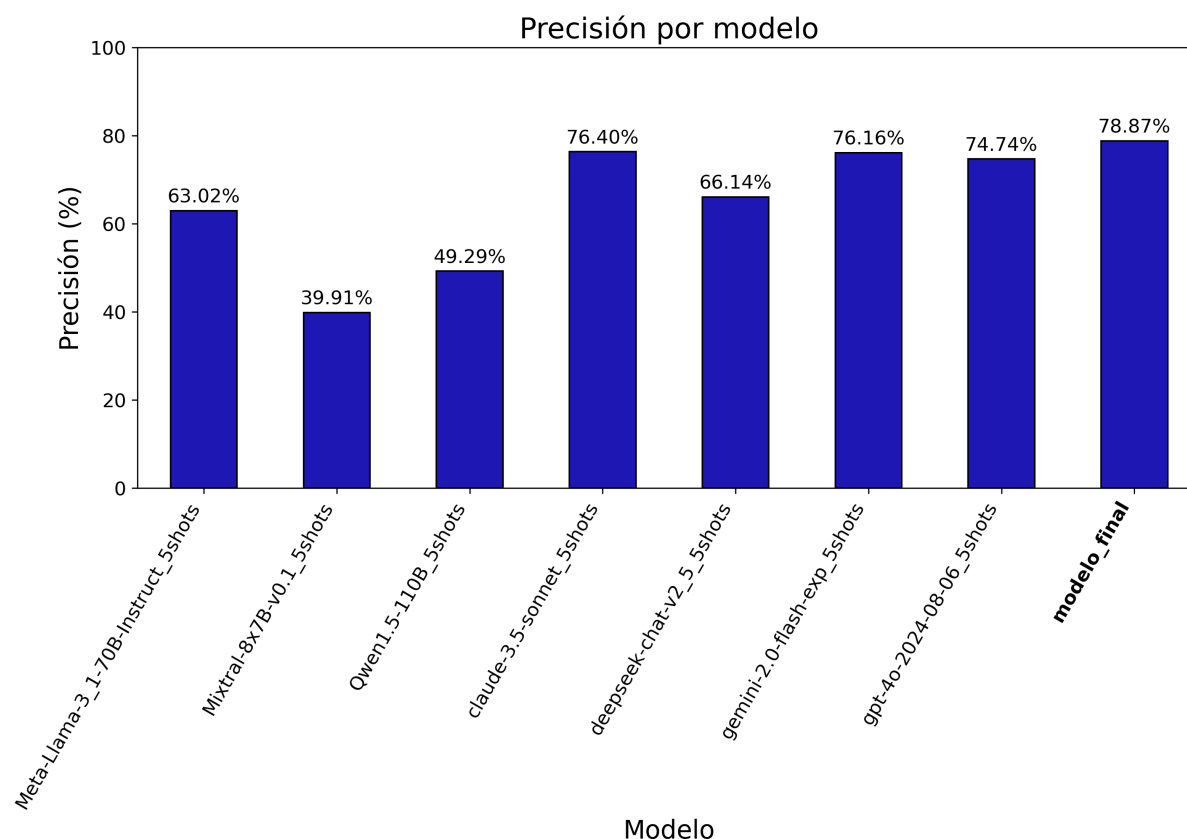


Figura 3.12: Precisión de la red neuronal en comparación con los modelos base

En la figura 3.12 se observa cómo el modelo propuesto supera en precisión a cualquiera de los utilizados como base. Esto pone de manifiesto la capacidad del sistema para combinar y aprovechar la información de las distintas predicciones, logrando un rendimiento superior al de los modelos individuales.

3.4.2. Optimización de hiperparámetros

La red neuronal multicapa ha tenido un buen desempeño teniendo en cuenta la falta de optimización de sus hiperparámetros. Por lo tanto, se ha decidido realizar una búsqueda intensiva para lograr una mejora en la precisión del modelo. Para ello, se ha buscado el valor óptimo de varios de sus hiperparámetros, concretamente los siguientes:

- **Activación:** se trata de la función de activación que se aplica a la salida de cada neurona en la capa en cuestión. Como valores puede tomar “relu”, “tanh” o “elu”. En la tabla 3.10 se muestran los resultados de cada tipo de activación.

Función de activación	Precisión (%)
relu	78.87
tanh	78.56
elu	78.93

Tabla 3.10: Precisión obtenida según la función de activación utilizada.

- **Dropout (tasa de abandono):** es la fracción de neuronas que se desactivan aleatoriamente en cada iteración del entrenamiento. Este valor puede variar entre 0 y 1. En la tabla 3.11 aparecen 5 posibles tasas de abandono con sus respectivas precisiones.

Tasa de abandono	Precisión (%)
0.0	78.49
0.1	78.53
0.2	79.11
0.3	78.87
0.5	78.76

Tabla 3.11: Precisión obtenida según la tasa de abandono utilizada.

- **Capas ocultas:** capas intermedias de una red neuronal, situadas entre la capa de entrada y la capa de salida. Cada capa oculta está formada por un conjunto de neuronas que reciben información de la capa anterior, la procesan mediante funciones de activación y la transmiten a la siguiente capa. Aunque es común utilizar potencias de 2 y reducir el número de neuronas en capas sucesivas, no es obligatorio seguir este patrón. En la tabla 3.12 se muestran los resultados de 5 posibles grupos de capas para este caso.

Capas ocultas	Precisión (%)
[512, 256, 128, 64, 32]	78.80
[256, 128, 64, 32, 16]	78.87
[128, 64, 32, 16, 8]	78.49
[64, 32, 16, 8, 4]	76.44
[32, 16, 8, 4, 2]	76.44
[16, 8, 4, 2, 1]	76.44

Tabla 3.12: Precisión obtenida según la configuración de capas ocultas.

- **Batch size (tamaño de lote):** indica el número de muestras del conjunto de entrenamiento que se procesan juntas antes de actualizar los pesos de la red neuronal. No es necesario que sea potencia de 2, aunque es habitual por cuestiones de eficiencia computacional. En la tabla 3.13 se muestran los resultados obtenidos con 5 tamaños de lote diferentes.

Tamaño de lote	Precisión (%)
4	78.73
8	78.87
16	78.87
32	78.73
64	79.11

Tabla 3.13: Precisión obtenida según el tamaño de lote utilizado.

- **Ratio de aprendizaje:** controla cuánto se ajustan los pesos de la red neuronal en cada paso del entrenamiento. Este valor suele encontrarse en 0.00001 y 0.1, pero no tiene restricciones. En la tabla 3.14 aparecen 4 posibles ratios de aprendizaje con sus porcentajes de acierto asociados.

Ratio de aprendizaje	Precisión (%)
0.01	76.20
0.001	78.87
0.0005	78.87
0.0001	79.07

Tabla 3.14: Precisión obtenida según el ratio de aprendizaje.

- **Épocas:** es el número de veces que el modelo recorre completamente todo el conjunto de datos de entrenamiento durante el proceso de aprendizaje. Un mayor número de épocas permite que el modelo aprenda más, pero también aumenta el riesgo de sobreajuste si se entrena demasiado. Su valor suele oscilar entre 10 y 100. En la tabla 3.15 se muestran los resultados obtenidos al variar el número de épocas.

Épocas	Precisión (%)
10	78.80
20	78.80
30	78.87
50	78.83
75	78.90
100	78.59

Tabla 3.15: Precisión obtenida según el número de épocas.

Tras evaluar el impacto de cada hiperparámetro de forma individual, se ha probado una configuración que combina los mejores valores obtenidos para cada uno de ellos. Esta combinación permite comprobar el rendimiento que puede alcanzar el modelo con una buena selección de hiperparámetros. La tabla 3.16 muestra los valores seleccionados y la precisión final lograda, que asciende a un 79.17 %.

Hiperparámetro	Valor óptimo
Función de activación	elu
Tasa de abandono	0.2
Capas ocultas	[256, 128, 64, 32, 16]
Tamaño de lote	64
Ratio de aprendizaje	0.0001
Épocas	75
Precisión final (%)	79.17

Tabla 3.16: Mejor combinación de hiperparámetros y la precisión lograda.

3.4.3. Conclusiones de la Fase 3

En esta tercera fase se ha explorado el potencial de una red neuronal multicapa como meta-modelo para combinar las predicciones de los modelos base. El objetivo era aprovechar la complementariedad y mejorar la precisión global del sistema, superando el enfoque de utilizar cada modelo de forma individual.

A través de un proceso sistemático de optimización de hiperparámetros, se ha ajustado la arquitectura y los parámetros de la red neuronal, logrando cierta mejora con respecto a la configuración inicial. El análisis detallado de cada hiperparámetro ha permitido identificar combinaciones que maximizan la precisión del modelo. La mejor configuración encontrada para la red neuronal ha alcanzado una precisión del **79.17 %**, lo que supone una mejora respecto a su resultado inicial (78.87 %) y la sitúa en un nivel competitivo frente a los modelos de las fases anteriores, cuyos resultados se resumen en la tabla 3.9.

En conclusión, la Fase 3 confirma que el uso de un meta-modelo basado en redes neuronales es una estrategia válida para combinar predicciones y mejorar la robustez del sistema. La optimización de hiperparámetros es clave para alcanzar un rendimiento competitivo, y el enfoque propuesto permite aprovechar las fortalezas de los modelos individuales.

4. Discusión

Para este TFG se propuso como objetivo principal desarrollar y evaluar un modelo de inteligencia artificial que superara en rendimiento a los LLM actuales en la resolución de preguntas tipo test, empleando estrategias de ensamble. Este enfoque buscaba abordar las limitaciones de los LLM individuales en tareas que requieren un razonamiento complejo y una alta precisión.

A lo largo del desarrollo, se ha confirmado la eficacia de las técnicas de ensamble como herramientas resolutivas en la mejora de la precisión y la robustez de los sistemas de aprendizaje automático. Se ha confirmado el principio de que el conocimiento colectivo de múltiples modelos base supera la capacidad de un único experto.

Los principales hallazgos y resultados se resumen a continuación por fases:

■ Fase 1: Metodología Programática y Selección Manual

Esta fase inicial, que utilizó un amplio conjunto de modelos disponibles en MMLU-Pro Github, logró un 83.79 % de acierto. Con este resultado se ha superado la precisión alcanzada por cualquiera de los modelos originales individualmente. Esto demuestra que una buena selección de grupos de modelos según la categoría es una estrategia efectiva para maximizar la precisión global.

En la primera parte de esta fase se destaca la presencia de modelos con configuraciones muy específicas para MMLU-Pro. Por ello, se particularizó el enfoque a un subconjunto de 7 modelos populares (se pueden observar en la sección 2.6). Siguiendo la misma estrategia definida en esta fase, se logró alcanzar un 79.22 % de acierto, que de nuevo superó a los modelos individuales.

■ Fase 2: Clasificación Automatizada y Optimización de Hiperparámetros

En esta fase se exploró una estrategia automatizada, utilizando *One Hot Encoder* y 4 clasificadores diferentes: *Random Forest Classifier*, *K Neighbors Classifier*, *Support Vector Classifier (SVC)* y *Linear Regression*. Mediante esta metodología se logró superar el 79.22 % de acierto de la Fase 1 para el subconjunto de 7 modelos.

Los resultados concretos definieron que tanto *Random Forest Classifier*, con un 79.48 % de acierto, como *Support Vector Classifier (SVC)*, con un 79.41 %, superaron el resultado de la Fase 1. Sin embargo, tanto *K Neighbors Classifier* con un competitivo 78.46 %, como *Linear Regression*, que demostró ser ineficaz en este contexto, con un 61.11 % de precisión, tuvieron unos resultados inferiores.

■ Fase 3: Ensamble con Red Neuronal

Esta fase buscó explotar las capacidades complementarias de los modelos base utilizando una red neuronal multicapa como meta-modelo. La red neuronal fue diseñada para

predecir, para cada ejemplo, cuál de los modelos base era más probable que acertara la respuesta correcta.

En su configuración inicial, el sistema obtuvo una precisión del 78.87 %, superando individualmente a cualquiera de los modelos base utilizados. Sin embargo, tras realizarse una optimización de hiperparámetros, se mejoró el rendimiento. La mejor combinación descubierta llevó a la red neuronal a una precisión final del 79.17 %. Este resultado superó en precisión a cualquiera de los modelos base individuales y se situó al nivel de las fases anteriores en cuanto a resultados.

Toda la información expuesta anteriormente se muestra en la tabla 4.1. En ella se puede observar tanto el resultado de aplicar cada uno de los métodos, como el mejor resultado de cada fase.

Modelo	Precisión (%)
Claude 3.5 Sonnet ^a	76.40 %
Respuesta más Típica Top 3 ^a	78.70 %
Metodología Programática	79.22
Fase 1	79.22
Random Forest Classifier	79.48
K Neighbors Classifier	78.46
Support Vector Classifier	79.41
Linear Regression	61.11
Fase 2	79.48
Red Neuronal	78.87
Red Neuronal Optimizada	79.17
Fase 3	79.17

^a Estos modelos corresponden al mejor modelo base individual y al mejor resultado obtenido mediante la generación de la respuesta más frecuente de los n modelos más precisos. Esta información se puede observar en la figura 3.4

Tabla 4.1: Resultados de todas las metodologías empleadas.

En conclusión, este TFG demuestra que, aunque las mejoras sean moderadas, la aplicación de técnicas de ensamble es una vía prometedora para superar las limitaciones de los LLM individuales en tareas de razonamiento complejo, como la resolución de preguntas tipo test. El éxito radica en la diversidad de los modelos base, lo que asegura que estos cometan diferentes tipos de errores y se complementen mutuamente. Si bien se lograron mejoras significativas, el proyecto también puso de manifiesto que las limitaciones computacionales pueden ser un desafío para la optimización completa de estas arquitecturas complejas. Este trabajo aporta una contribución significativa al campo de la inteligencia artificial, sentando las bases para el desarrollo de sistemas más precisos y robustos.

5. Líneas futuras

Una vez sentadas las bases, es importante considerar las posibles líneas futuras que pueden surgir a partir de este trabajo. Estas líneas no solo permitirán obtener resultados más precisos, sino que también contribuirán al avance del conocimiento en el campo de la inteligencia artificial y el procesamiento del lenguaje natural. Existen áreas de mejora y expansión que pueden ser exploradas para enriquecer el trabajo realizado. A continuación, se presentan algunas de las líneas futuras más relevantes:

- **Mejora de la selección de modelos:** investigar y aplicar técnicas más avanzadas para la selección de modelos base, considerando no solo su rendimiento individual, sino también su complementariedad y diversidad. Los 7 modelos seleccionados en la fase 1 demostraron ser efectivos, pero una exploración más amplia podría revelar combinaciones aún más potentes, incluso con modelos más pequeños o menos precisos por sí solos.
- **Probar con distintos grupos de modelos y modos de desempate en la Fase 1:** en este trabajo se han utilizado grupos de los n mejores modelos, pero se podrían explorar otras combinaciones, como se ha hecho en la Fase 3. Adicionalmente, en caso de empate con el método *mode()* (moda), se eligió el primero de los resultados que apareciese (*mode()[0]*), por lo que, en esos casos, se podrían añadir o quitar modelos para tomar la decisión de desempate.
- **Optimización de hiperparámetros:** la falta de capacidad computacional limitó la optimización de hiperparámetros en este trabajo. En futuras investigaciones, se podría dedicar más tiempo y recursos a la optimización de los hiperparámetros de los modelos base y del meta-modelo. Esto podría incluir el uso de múltiples GPU.
- **Ampliación del conjunto de datos:** el conjunto de datos MMLU-Pro utilizado en este trabajo es relativamente pequeño. En futuras investigaciones, se podría considerar la ampliación del conjunto de datos con más preguntas o la inclusión de preguntas de diferentes dominios para evaluar la adaptabilidad del sistema a otros contextos.
- **Validación con datos reales:** aunque se ha utilizado un conjunto de datos estandarizado, sería beneficioso validar el sistema con datos reales o en situaciones más cercanas a la realidad. Esto permitiría evaluar la robustez y aplicabilidad del enfoque en escenarios prácticos.
- **Exploración de nuevas arquitecturas de meta-modelo:** investigar y probar diferentes arquitecturas de meta-modelo, como redes neuronales más complejas o técnicas de ensamble no utilizadas en este trabajo.

Estas líneas futuras permitirán continuar el trabajo iniciado, aportando mejoras y nuevas perspectivas al campo de estudio abordado.

Bibliografía

- [1] «LLM Evaluation Essentials: From LLM-as-a-Judge to Perplexity (Part 1)», [en línea]. Available: <https://medium.com/data-analytics-at-nesta/llm-evaluation-essentials-from-llm-as-a-judge-to-perplexity-part-1-04294bfff304>.
- [2] «ChatGPT Replicates Gender Bias in Recommendation Letters», [en línea]. Available: <https://www.scientificamerican.com/article/chatgpt-replicates-gender-bias-in-recommendation-letters/>.
- [3] «A Survey on Evaluation of Large Language Models», [en línea]. Available: <https://arxiv.org/pdf/2307.03109>.
- [4] «From Generation to Judgment: Opportunities and Challenges of LLM-as-a-judge», [en línea]. Available: <https://arxiv.org/pdf/2411.16594>.
- [5] «Ensemble deep learning: A review», [en línea]. Available: https://www.researchgate.net/publication/362378768_Ensemble_deep_learning_A_review.
- [6] «Descubre los increíbles métodos ensemble en Ciencia de Datos», [en línea]. Available: <https://edumaster.ar/descubre-los-increibles-metodos-ensemble-en-ciencia-de-datos/>.
- [7] «Guía definitiva a las técnicas de Ensemble», [en línea]. Available: <https://medium.com/@nicolasarrioja/gu%C3%ADa-definitiva-a-las-t%C3%A9cnicas-de-ensemble-7a4bb1203bcb>.
- [8] «Stacking ensemble de modelos de forecasting», [en línea]. Available: <https://cienciadedatos.net/documentos/py52-stacking-ensemble-modelos-forecasting>.
- [9] «Stacking to Improve Model Performance: A Comprehensive Guide on Ensemble Learning in Python», [en línea]. Available: https://medium.com/@brijesh_soni/stacking-to-improve-model-performance-a-comprehensive-guide-on-ensemble-learning-in-python-9.
- [10] «On Collective Robustness of Bagging Against Data Poisoning», [en línea]. Available: <https://proceedings.mlr.press/v162/chen22k/chen22k.pdf>.
- [11] «Data Poisoning: The Exploitation of Generative AI», [en línea]. Available: <https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/data-poisoning/>.
- [12] «Combining Predictions Under Uncertainty: The Case of Random Decision Trees», [en línea]. Available: https://www.researchgate.net/publication/355174141_Combining_Predictions_Under_Uncertainty_The_Case_of_Random_Decision_Trees.
- [13] «Binary Classification: Is Boosting stronger than Bagging?», [en línea]. Available: <https://arxiv.org/html/2410.19200v1>.
- [14] «Random Forests», [en línea]. Available: <https://link.springer.com/article/10.1023/A:1010933404324>.

- [15] «Unraveling the Mystery of Large Language Models: A Peek Behind the Curtain», [en línea]. Available: <https://medium.com/%40bgsumanth8/unraveling-the-mystery-of-large-language-models-a-peek-behind-the-curtain-80fff23bb68c>.
- [16] «A Review of Current Trends, Techniques, and Challenges in Large Language Models (LLMs)», [en línea]. Available: <https://www.mdpi.com/2076-3417/14/5/2074>.
- [17] «Chinchilla: Training Compute-Optimal Large Language Models», [en línea]. Available: <https://arxiv.org/pdf/2203.15556>.
- [18] «Position: The Most Expensive Part of an LLM should be its Training Data», [en línea]. Available: <https://arxiv.org/pdf/2504.12427>.
- [19] «Counting The Cost Of Training Large Language Models», [en línea]. Available: <https://www.nextplatform.com/2022/12/01/counting-the-cost-of-training-large-language-models/>.
- [20] «Introducing Claude 3.5 Sonnet Anthropic», [en línea]. Available: <https://www.anthropic.com/news/claude-3-5-sonnet>.
- [21] «DeepSeek R1: DeepSeek v2.5», [en línea]. Available: <https://deepseeksr1.com/v2-5/>.
- [22] «9TO5 Google - Updated Gemini 2.0 Flash Thinking Experimental», [en línea]. Available: <https://9to5google.com/2025/01/21/gemini-2-0-flash-thinking-experimental-jan-2025/>.
- [23] «Google AI Dev - Gemini 2.0 Flash - Docs», [en línea]. Available: <https://ai.google.dev/gemini-api/docs/models#gemini-2.0-flash>.
- [24] «Introducing GPT-4o-2024-08-06 API with Structured Outputs on Azure — Microsoft Community Hub», [en línea]. Available: <https://techcommunity.microsoft.com/blog/azure-ai-services-blog/introducing-gpt-4o-2024-08-06-api-with-structured-outputs-on-azure/4232684>.
- [25] «Llama 3.1 — Model Cards and Prompt formats», [en línea]. Available: https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_1/.
- [26] «mixtral-8x7b-instruct-v0.1 Model by Mistral AI — NVIDIA NIM», [en línea]. Available: <https://build.nvidia.com/mistralai/mixtral-8x7b-instruct/modelcard>.
- [27] «qwen:110b», [en línea]. Available: <https://ollama.com/library/qwen:110b>.
- [28] «MMLU-Pro Benchmarks», [en línea]. Available: <https://huggingface.co/spaces/TIGER-Lab/MMLU-Pro>.
- [29] «MMLU-Pro Github», [en línea]. Available: <https://github.com/TIGER-AI-Lab/MMLU-Pro>.
- [30] «Discusión sobre los resultados de ARX en Reddit», [en línea]. Available: https://www.reddit.com/r/LocalLLaMA/comments/1f5ii16/where_did_arx03_come_from_and_who_makes_it/.
- [31] «Publicación en x.com de AGI», [en línea]. Available: <https://x.com/appliedgeneral/status/1884738566645018932?s=46>.
- [32] «CohereLabs/c4ai-command-r-v01 · Hugging Face», [en línea]. Available: <https://huggingface.co/CohereLabs/c4ai-command-r-v01>.

- [33] «Claude 3.5 Haiku: Pricing, Context Window, Benchmarks, and More», [en línea]. Available: <https://llm-stats.com/models/claude-3-5-haiku-20241022/>.
- [34] «Claude 3.5 Sonnet — Generative AI on Vertex AI — Google Cloud», [en línea]. Available: <https://cloud.google.com/vertex-ai/generative-ai/docs/partner-models/claude/sonnet-3-5>.
- [35] «The Ultimate Review of Claude 3.5 Sonnet AI—Comparison to ChatGPT & All You Need to Know!», [en línea]. Available: <https://felloai.com/2024/08/claude-ai-everything-you-need-to-know/>.
- [36] «DeepSeek LLM Scaling Open-Source Language Models with Longtermism», [en línea]. Available: <https://arxiv.org/pdf/2401.02954>.
- [37] «DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model», [en línea]. Available: <https://arxiv.org/pdf/2405.04434>.
- [38] «deepseek-ai/DeepSeek-Coder-V2: DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence», [en línea]. Available: <https://github.com/deepseek-ai/DeepSeek-Coder-V2>.
- [39] «AlibabaResearch/flash-llm», [en línea]. Available: <https://github.com/AlibabaResearch/flash-llm?tab=readme-ov-file>.
- [40] «Google’s Gemini 1.5 Flash (002) - AI Model Details», [en línea]. Available: <https://docsbot.ai/models/gemini-1-5-flash-002>.
- [41] «Google Gemini 2.0 llega para todos, un gran paso para enfrentarse a DeepSeek — Lifestyle — SmartLife — Cinco Días», [en línea]. Available: <https://cincodias.elpais.com/smartlife/lifestyle/2025-02-06/google-gemini-20-para-todos.html>.
- [42] «google/gemma-7b · Hugging Face», [en línea]. Available: <https://huggingface.co/google/gemma-7b>.
- [43] «Google Gemma Model Stats. Google’s newly released Gemma model (7B... — by Michael Humor — Medium», [en línea]. Available: <https://medium.com/%40michaelhumor/gemma-model-stats-3d354bde46c9>.
- [44] «GPT-4o mini: advancing cost-efficient intelligence — OpenAI», [en línea]. Available: <https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/>.
- [45] «OpenAI’s GPT-4o 2024-05-13 - AI Model Details», [en línea]. Available: <https://docsbot.ai/models/gpt-4o-2024-05-13>.
- [46] «Azure OpenAI in Azure AI Foundry Models - Azure OpenAI — Microsoft Learn», [en línea]. Available: <https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/models?tabs=global-standard%2Cstandard-chat-completions>.
- [47] «OpenAI unveils GPT-4o mini, a smaller and cheaper AI model — TechCrunch», [en línea]. Available: https://techcrunch.com/2024/07/18/openai-unveils-gpt-4o-mini-a-small-ai-model-powering-chatgpt/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cucmVkJmNvbS8&guce_referrer_sig=AQAAALbJ9K1G268VbjOwQc75mmZ8nDQVVLNjM-dbUYNcEQyEw7A0Uo-IN6Ov6bh8oJRkXPISrwsK9A3-BLfF341EjblspIp2chobXoPt97Yq.
- [48] «iAsk Pro · Ask AI Questions», [en línea]. Available: <https://iask.ai/pricing>.

- [49] «jamba-1.5-large-instruct Model by AI21 Labs — NVIDIA NIM», [en línea]. Available: <https://build.nvidia.com/ai21labs/jamba-1.5-large-instruct/modelcard>.
- [50] «meta-llama/Llama-2-7b-chat-hf · Hugging Face», [en línea]. Available: <https://huggingface.co/meta-llama/Llama-2-7b-chat-hf>.
- [51] «What You Need to Know About Meta's Llama 2 Model — Deepgram», [en línea]. Available: <https://deepgram.com/learn/llama-2-llm-explained>.
- [52] «Llama 3 70B vs GPT-4: Comparison Analysis», [en línea]. Available: <https://www.vellum.ai/blog/llama-3-70b-vs-gpt-4-comparison-analysis>.
- [53] «Introducing Llama 3.1: Our most capable models to date», [en línea]. Available: <https://ai.meta.com/blog/meta-llama-3-1/>.
- [54] «Mistral releases three new LLMs for math, code and general tasks», [en línea]. Available: <https://the-decoder.com/mistral-releases-three-new-llms-for-math-code-and-general-tasks/>.
- [55] «Mixtral of experts — Mistral AI», [en línea]. Available: <https://mistral.ai/news/mixtral-of-experts/>.
- [56] «Introducing the next generation of Claude Anthropic», [en línea]. Available: <https://www.anthropic.com/news/claude-3-family>.
- [57] «microsoft/Phi-3-mini-4k-instruct · Hugging Face», [en línea]. Available: <https://huggingface.co/microsoft/Phi-3-mini-4k-instruct>.
- [58] «qwen:110b-text-v1.5-q4_1», [en línea]. Available: https://ollama.com/library/qwen%3A110b-text-v1.5-q4_1.
- [59] «Yi: Open Foundation Models by 01.AI», [en línea]. Available: <https://arxiv.org/pdf/2403.04652>.
- [60] «Billion vs. billón», [en línea]. Available: <https://www.bbc.com/mundo/noticias-58598282>.
- [61] «MMLU-Pro Datasets», [en línea]. Available: <https://huggingface.co/datasets/TIGER-Lab/MMLU-Pro>.
- [62] «Hyperparameter Definition — DeepAI», [en línea]. Available: <https://deepai.org/machine-learning-glossary-and-terms/hyperparameter>.
- [63] «Minkowski Distance - GeeksforGeeks», [en línea]. Available: <https://www.geeksforgeeks.org/minskowski-distance/>.
- [64] «Salario de un ingeniero de telecomunicaciones en España», [en línea]. Available: https://www.glassdoor.es/Sueldos/ingeniero-de-telecomunicaciones-sueldo-SRCH_KO0%2C31.htm.
- [65] «¿Cuántas horas trabaja un ingeniero de telecomunicaciones?», [en línea]. Available: <https://payfit.com/es/contenido-practico/convenio-ingenierias-y-oficinas-tecnicas/>.

A. Anexo A: Descripción del funcionamiento del *One Hot Encoder*

A.1. One Hot Encoder

A simple vista, un *OneHotEncoder* puede parecer un modelo sencillo y liso, pues recibe una serie de letras y devuelve una cantidad muy superior de dígitos, concretamente unos y ceros. Sin embargo, es muy útil para transformar datos categóricos en un formato que los modelos de aprendizaje automático pueden entender y procesar. Se muestra un ejemplo para la comprensión de su funcionamiento:

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Se crea un DataFrame de ejemplo
data = pd.DataFrame({
    'mod1': ['A', 'B', 'C', 'A'],
    'mod2': ['A', 'A', 'B', 'C'],
    'mod3': ['C', 'B', 'A', 'C'],
    'correcta': ['A', 'B', 'C', 'A']
})
# Se separan las respuestas de los modelos (X)
# y las respuestas correctas (Y)
X = data.drop('correcta', axis=1)
Y = data['correcta']
# Se crea el OneHotEncoder y se aplica al
# DataFrame de los modelos
encoder = OneHotEncoder(sparse=False)
X_encoded = encoder.fit_transform(X)
```

El resultado de aplicar el *One Hot Encoder* al *DataFrame* de ejemplo es el siguiente:

```
X_encoded
array([
    [1., 0., 0., 1., 0., 0., 0., 0., 1.],
    [0., 1., 0., 1., 0., 0., 0., 1., 0.],
    [0., 0., 1., 0., 1., 0., 1., 0., 0.],
    [1., 0., 0., 0., 0., 1., 0., 0., 1.]
])
```

Se recibe como entrada un *DataFrame* con 4 columnas y 3 filas ('mod1', 'mod2' y 'mod3') de

letras, y se obtiene una matriz de 4 filas y 9 columnas de dígitos binarios. Se deben de tener en cuenta los siguientes puntos:

- Hay presentes 3 letras diferentes, las primeras del abecedario: 'A', 'B' y 'C'.
- El *DataFrame* de entrada se transpone (pero no solo eso) para dar el de salida. En el ejemplo hay 4 columnas de entrada, que se convierten en 4 filas en la salida.
- Con la información anterior, se obtienen las 9 columnas que hay en la salida: la transposición debería dar 3 columnas, pero hay un factor de 3 que coincide con el número de letras diferentes.

Aquí se muestran juntas la entrada (X) transpuesta y la salida:

A A C	1 0 0 1 0 0 0 0 1
B A B	0 1 0 1 0 0 0 1 0
C B A	0 0 1 0 1 0 1 0 0
A C C	1 0 0 0 0 1 0 0 1

Agrupando de 3 en 3 (letras diferentes) las columnas de la salida, se puede observar que hay un 1 y dos 0 en cada grupo. Obtenemos esta división:

A A C	1 0 0	1 0 0	0 0 1
B A B	0 1 0	1 0 0	0 1 0
C B A	0 0 1	0 1 0	1 0 0
A C C	1 0 0	0 0 1	0 0 1

A es '1 0 0', B es '0 1 0' y C es '0 0 1'. Se llega a la conclusión de que cada columna corresponde a una letra, donde el uno nos indica que se trata de esa letra y el cero que no lo es. Por ejemplo, '1 0 0' indica que es la letra A y que no es ni B ni C.

A.2. Train Test Split

La función *train_test_split*, como su nombre indica, se utiliza para dividir un conjunto de datos en dos partes: una para entrenar el modelo y otra para probar su rendimiento. Esta división es crucial para evaluar la capacidad del modelo, cuando se generalice a datos no vistos.

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(
    X_encoded, Y, test_size=0.2, random_state=42
)
```

Programación A.1: Ejemplo de código de un *train_test_split*.

De la sección anterior, (One Hot Encoder), se usan *X_encoded* e *Y*, que se pueden ver en la Programación A.2.

- Estos 2 conjuntos de datos, junto a 2 variables adicionales (*test_size* y *random_state*), se pasan como parámetros.
 - *test_size* indica la proporción del conjunto de datos que se utiliza para la prueba. Su valor se establece en, por ejemplo, 0.2, esto significa que el 20 % de los datos se usan para la prueba y el 80 % restante para el entrenamiento. En este caso serían 1 y 3 filas, respectivamente.

- *random_state* se utiliza para asegurar la reproducibilidad en la partición de los datos, permitiendo obtener los mismos resultados en distintas ejecuciones. Por ejemplo, al asignarle el valor 42, la separación de los conjuntos será idéntica cada vez que se ejecute el programa. Si no se especifica, la partición cambiará en cada ejecución, lo que puede dificultar la comparación de resultados.
- Se obtienen como resultado 4 conjuntos de datos: *X_train*, *X_test*, *Y_train* e *Y_test*, los cuales se utilizan para entrenar y evaluar el modelo. En la Programación A.3 se muestran las salidas.
 - *X_train* e *Y_train* se utilizan para entrenar el modelo, mientras que *X_test* e *Y_test* se encargan de evaluar su rendimiento.
 - El número de filas y columnas son dependientes de la utilidad (entrenamiento o prueba) y de si es parte de la entrada (*X*) o de la salida (*Y*). Los *DataFrame* de entrenamiento tienen 3 filas y los de prueba 1. En cuanto a las columnas, los de entrada tienen 9 y los de salida 1.

```
data = pd.DataFrame({
    ...
    'correcta': ['A', 'B', 'C', 'A']
})
Y = data['correcta']
X_encoded = array([
    [1., 0., 0., 1., 0., 0., 0., 0., 1.],
    [0., 1., 0., 1., 0., 0., 0., 1., 0.],
    [0., 0., 1., 0., 1., 0., 1., 0., 0.],
    [1., 0., 0., 0., 0., 1., 0., 0., 1.]
])
```

Programación A.2: Conjuntos de datos utilizados para el *train_test_split*.

La salida de la ejecución de *train_test_split*, como se muestra en la Programación A.1, es la mostrada en la Programación A.3.

```
X_train
array([
    [1., 0., 0., 0., 0., 1., 0., 0., 1.],
    [1., 0., 0., 1., 0., 0., 0., 0., 1.],
    [0., 0., 1., 0., 1., 0., 1., 0., 0.]
])
Y_train
array(['A', 'A', 'C'])
X_test
array([[0., 1., 0., 1., 0., 0., 0., 1., 0.]])
Y_test
array(['B'])
```

Programación A.3: Resultado del *train_test_split*.

B. Anexo 1: Aspectos éticos, económicos, sociales y ambientales

El apartado “Requisitos de las acreditaciones internacionales EUR-ACE y ABET” de la Normativa de TFT de la ETSIT-UPM establece que *“La memoria del TFT del GITST, GIB y MUIT, y en general la de aquellas titulaciones que hayan obtenido o para las que se desee solicitar una acreditación internacional EUR-ACE o ABET, debe mostrar conciencia de la responsabilidad de la aplicación práctica de la ingeniería, el impacto social y ambiental, el compromiso con la ética profesional, la responsabilidad y las normas de la aplicación práctica de la ingeniería, así como sobre las prácticas de gestión de proyectos, gestión, y control de riesgos, entendiendo sus limitaciones”*.

Este anexo obligatorio del TFT tendrá un carácter sintético con los siguientes apartados:

A1. INTRODUCCIÓN

Breve descripción del contexto del proyecto, objetivos, necesidades que pretende cubrir o problemas que pretende resolver, centrándose en su relación con los temas sociales, económicos, éticos, legales y/o ambientales que se hayan identificado.

A2. DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO

Síntesis del trabajo realizado en la fase 3, de selección y descripción de impactos. Presentar y justificar las conclusiones a las que se haya llegado sobre cuáles son los asuntos más relevantes relacionados con la sostenibilidad social, económica o ambiental, así como los principales grupos de interés identificados y que se han considerado en los análisis posteriores.

A3. ANÁLISIS DETALLADO DE ALGUNO DE LOS IMPACTOS

Síntesis del trabajo de análisis realizado.

A4. CONCLUSIONES

Valorar el proyecto desde un punto de vista ético, social, económico y medioambiental y justificar si el uso de criterios de sostenibilidad ha aportado o puede aportar valor añadido al proyecto.

C. Anexo 2: Presupuesto económico

El apartado “Requisitos de las acreditaciones internacionales EUR-ACE y ABET” de la Normativa de TFT de la ETSIT-UPM establece que “*La memoria del TFT del GITST, GIB y MUIT, y en general la de aquellas titulaciones que hayan obtenido o para las que se desee solicitar una acreditación internacional EUR-ACE o ABET, ... debe incluir un presupuesto económico*”. A modo de ejemplo, la tabla del presupuesto de un proyecto podría ser la siguiente:

COSTE DE MANO DE OBRA (coste directo)		Horas	Precio/hora	Total	
		300	16.18 € ^a	4 854 €	
COSTE DE RECURSOS MATERIALES (coste directo)		Precio de compra	Uso en meses	Amortización (en años)	Total
Ordenador personal (Software incluido) ^b		1 000.00 €	6	5	100.00 €
Ventilador para portátil		50.00 €	6	5	5.00 €
Otro equipamiento					
COSTE TOTAL DE RECURSOS MATERIALES					105.00 €
GASTOS GENERALES (costes indirectos)		15 %	sobre CD		743.85 €
BENEFICIO INDUSTRIAL		6 %	sobre CD + CI		342.17 €
SUBTOTAL PRESUPUESTO					6 045.02 €
IVA APLICABLE			21 %		1 269.45 €
TOTAL PRESUPUESTO					7 314.48 €

^a Se ha estimado en 16.18 € el coste por hora, calculado a partir de un salario medio de 29 000 € brutos anuales para un ingeniero de telecomunicación en España [64]. Para el cálculo se ha considerado el máximo de 1 792 horas de trabajo anuales establecido por el convenio de ingenierías y oficinas técnicas [65].

^b El software ha tenido un coste de 0 €, ya que se ha utilizado software libre y gratuito.

Tabla C.1: Presupuesto económico