UNIVERSIDAD TORCUATO DI TELLA

Escuela de Negocios
Licenciatura en Tecnología Digital

Descifrando el Cánon Arquitectónico del Siglo XX

Construcción de una metodología basada en Procesamiento de Lenguaje Natural

Autores: Chantal Levi Aparaín, Lucas De Fino, Pilar Solari Barrios
Profesora: María de los Angeles Scetta
Proyecto propuesto por: Julián Varas

29 de noviembre de 2024

Índice

Agradecimientos

Gracias a Julián Varas, quien propuso este proyecto, por todo el interés a lo largo del desarrollo. Fue muy valioso para nosotros contar con su opinión en todo momento.

Agradecemos a Angie Scetta por su apoyo a lo largo del cuatrimestre y por su disposición a comprender las complejidades de este proyecto.

Por último, queremos darle las gracias a la Universidad Torcuato Di Tella y a todo su personal.Todos los espacios y herramientas que nos brindaron nos permitieron llegar hasta donde estamos hoy.

"Alea iacta est: la suerte está echada."

Resumen Ejecutivo: Procesamiento de lenguaje natural aplicado a la obtención de un canon arquitectónico

En este trabajo se busca encontrar una metodología que "descifre" el canon arquitectónico del siglo XX, es decir, que encuentre patrones de datos que permitan observar cómo ha cambiado el canon arquitectónico del siglo XX a lo largo del tiempo. El canon reúne obras destacadas, fundamentales en la formación de profesionales, artistas y en el establecimiento de altos estándares de calidad. Esta investigación se enmarca en las Humanidades Digitales, aplicando inteligencia artificial para interpretar grandes cantidades de información de diversas fuentes escritas.

El objetivo principal del proyecto es identificar la mayor cantidad posible de obras arquitectónicas dentro de un conjunto de libros definido, para luego construir una base de datos que responda preguntas clave como: *¿quién diseñó cada obra?*, *¿en qué ciudad y país se encuentra?*, y *¿cuándo comenzó y terminó su construcción?* Además, se realiza un reconocimiento posicional en el texto, diferenciando aquellas obras que aparecen en títulos, párrafos o referencias; buscando responder la pregunta: *¿qué tan influyente fue cada obra según la mirada de cada autor?*

La metodología planteada emplea diversas técnicas de Procesamiento de Lenguaje Natural (Natural Language Processing). Comienza utilizando un modelo de Reconocimiento de Entidades Nombradas (Named Entity Recognition) para las obras en los libros, continúa con un modelo de Segmentación de Texto (Text Segmentation) para lograr el reconocimiento posicional, y finaliza con un modelo de Generación Aumentada por Recuperación (Retrieval-Augmented Generation) para responder las preguntas planteadas y así completar la base de datos.

Estos modelos, los cuales debieron ser afinados (fine-tuned) para nuestras tareas, obtuvieron los siguientes resultados en testeo:

- Named Entity Recognition: 69% de aciertos
- Text Classification: 85% accuracy
- Retrieval-Augmented Generation:
    - Accuracy Autor: 64%
    - Accuracy Ciudad: 67%
    - Accuracy País: 57%
    - Accuracy Inicio: 41%
    - Accuracy Fin: 35%

Habiendo analizado diversos modelos y alternativas en cada paso de la construcción de esta metodología, llegamos a la siguiente conclusión:

(1) Los modelos de Named Entity Recognition y Text Classification, si bien pueden alcanzar buenos resultados, son altamente dependientes de la calidad y cantidad de los datos de entrenamiento.

(2) El modelo de Retrieval-Augmented Generation demostró un buen funcionamiento en la extracción de resultados, aunque resultó ser bastante restrictivo respecto al poder de cómputo por lentitud y costos.

Introducción: Definición del canon arquitectónico

En el ámbito del arte, la arquitectura y otras disciplinas culturales, los conceptos, categorías, ideas y valoraciones se transmiten de generación en generación. Este proceso permite establecer consensos sobre las razones que justifican el interés en determinadas obras. De esta forma, resulta más sencilla la organización de conjuntos considerados relevantes a lo largo del tiempo. Tales conjuntos se consolidan como pilares fundamentales en la educación, la formación y la cultura.

La valoración que los autores otorgan a estas obras desempeña un papel crucial en el presente y futuro del área de interés. En este contexto, la selección de dichas obras cobra una importancia esencial, ya que constituye la muestra representativa a través de la cual se transmiten los valores globales del patrimonio arquitectónico y cultural de una ciudad, país o región.

La búsqueda de un canon arquitectónico surge de una necesidad de encontrar un conjunto de obras que sirvan como referencia y apoyo tanto al desarrollo de nuevas propuestas como a la continuidad cultural en una comunidad. A lo largo del tiempo, este ha influido en el diseño y en la valoración arquitectónica, funcionando como una guía para establecer criterios y el legado cultural en la disciplina. Pero el canon de la arquitectura occidental del siglo XX es una definición variable y de bordes imprecisos: se modifica con el paso del tiempo, y cada autor ofrece una versión diferente. El análisis comparativo de esas miradas permite entender con precisión esas diferencias, referenciándolas a parámetros específicos de los libros en que se manifiestan (tales como, país de origen, año de publicación, y otros datos relevantes sobre el autor, la editorial, y el público para el cual se pensó la obra).

Este proyecto fue propuesto por Julián Varas, profesor de la Escuela de Arquitectura y Estudios Urbanos de esta universidad. En pos de alcanzar mayor objetividad en las bases para futuras producciones, Julián plantea aprovechar el auge de la inteligencia artificial para acelerar el análisis de libros que por el contrario llevaría incontables horas de lectura.

Justificación del tema: ¿Por qué construir un canon?

Cómo definimos a lo largo de la introducción, la constitución del canon arquitectónico es fundamental para comprender diversos aspectos, como las razones detrás de la arquitectura de una ciudad, el valor de una obra arquitectónica en un contexto específico, cuáles de estas obras son representativas en un momento determinado de la historia y por qué lo son, así como el desarrollo e innovación dentro de su ámbito. Pero esta metodología si bien toma la arquitectura en su implementación, puede aplicarse con leves variaciones a otras humanidades como Historia, Ciencias Sociales y Arte, volviéndo una gran herramienta para acceder a un tipo de investigación basada en datos, poco usual en el campo de las humanidades. Por esto es que consideramos que el tema elegido tiene alta relevancia a nivel académico.

A nivel práctico, el proyecto requirió enlazar tres técnicas de Procesamiento de Lenguaje Natural definiendo un camino a seguir para el análisis de bibliografía. La rapidez de esta metodología por sobre el trabajo manual termina de darle importancia como herramienta.

Planteamiento del Problema

Realizar una caracterización precisa de los cambios en la definición del canon arquitectónico, desde un autor a otro y a lo largo de distintos momentos históricos, es una tarea compleja en términos de análisis, que requiere recorrer una gran cantidad de bibliografía.

Si se focaliza exclusivamente en las menciones de obras arquitectónicas y su valoración según el lugar que ocupan en cada libro, una máquina capaz de analizar estas páginas y extraer dicha información podría ofrecer un aporte clave para construir una narrativa sobre la evolución del canon de manera mucho más eficiente. Además, podría ser un proceso que puede adaptarse a nuevos textos y proveer una visión dinámica del presente e ir corrigiendo y ajustando la visión del canon al presente.

El desafío radica en desarrollar un método automatizado que permita distinguir las referencias a obras de arquitectura contenidas en los libros de todo el universo de nombres propios que se mencionan en los mismos.

Hipótesis

Se plantea la hipótesis de que es posible desarrollar un modelo de Reconocimiento de Entidades Nombradas especializado en identificar obras y proyectos arquitectónicos mencionados en libros de historia. Este modelo tendría la capacidad de diferenciar dichas menciones del resto de los nombres propios que suelen aparecer en este tipo de textos, como los de autores, lugares y conceptos relacionados.

Dada la hipótesis, un arquitecto o estudiante en arquitectura podría llevar a cabo un análisis exhaustivo basado en los resultados del proyecto. La información obtenida a partir de estos modelos permitirá establecer comparaciones entre diferentes textos, lo que podría facilitar el reconocimiento de tendencias y preferencias de los autores en aspectos como lo geográfico, lo estilístico y otras valoraciones comúnmente empleadas en la selección de materiales que conforman este tipo de textos. Sin embargo, en esta ocasión no profundizaremos en el análisis del canon arquitectónico obtenido, ya que no forma parte del objetivo de este proyecto.

Objetivos del Proyecto

El objetivo principal del proyecto es consolidar una base de datos por cada libro propuesto y, por ende, reconocer la mayor cantidad posible de obras arquitectónicas junto con la información sobre ellas. Para alcanzar esto definimos los siguientes pasos (o subtareas):

1. Preprocesar los libros para facilitar la extracción de información.
2. Adaptar un modelo NER pre-entrenado para reconocer nuestra entidad de interés: obras de arquitectura. Generar datos de entrenamiento de cada libro para poder capturar información a pesar de los distintos estilos de escritura. Implementar técnicas de procesamiento de input y optimizar el modelo ajustando hiper parámetros.
3. Implementar un modelo de clasificación para luego determinar en qué segmento del texto se encuentra cada obra arquitectónica.
4. Preparar un modelo RAG que genere la información pertinente a cada obra arquitectónica reconocida por el modelo NER, para completar la base de datos.
5. Evaluar los resultados obtenidos de cada modelo, implementando testeos, evaluando métricas e iterando sobre distintas combinaciones de hiper parámetros para mejorar la precisión de la base de datos.

Marco teórico: Introducción a las Tecnologías Utilizadas

Procesamiento de Lenguaje Natural (NLP)

El **procesamiento de lenguaje natural** (NLP, por sus siglas en inglés) es un campo interdisciplinario que combina la lingüística, la inteligencia artificial y la informática para permitir que las máquinas comprendan, interpreten y generen lenguaje humano de forma significativa. Este proceso no se limita a una transformación directa de texto a datos procesables, sino que busca capturar los significados sintácticos y semánticos subyacentes en el lenguaje.

Un paso inicial clave en el procesamiento de lenguaje natural consiste en dividir el texto en unidades más pequeñas llamadas tokens. Este proceso se llama **tokenización** y facilita la lectura para el modelo. Estos tokens pueden ser palabras, subpalabras o incluso caracteres, dependiendo de la granularidad requerida por el modelo. Por ejemplo, en la frase "El día está soleado", la tokenización podría producir los siguientes tokens: ["El", "día", "está", "sole", "ado"].

Una vez tokenizado el texto, cada token puede ser representado mediante vectores, conocidos como **word embeddings**, en un espacio multidimensional. Esta es la forma que tiene el modelo de entender relaciones semánticas y sintácticas. Pero, para lograr comprender correctamente un texto completo, los modelos deben aprovechar el contexto general en vez de procesar palabras de manera aislada. Para esto se usan modelos avanzados como los **Transformers** que emplean mecanismos de **atención** (*attention mechanisms*) y así otorgar importancia a distintas partes del texto dependiendo de la palabra que se esté procesando.

Gracias a estos procesos, los sistemas de NLP pueden llevar a cabo tareas complejas como traducción automática, resumen de documentos y generación de texto, al lograr una comprensión más profunda y contextual del lenguaje humano.

Para implementar tareas de NLP, **Python** y **R** son los lenguajes de programación elegidos frecuentemente debido a su amplia gama de bibliotecas y su facilidad para integrar modelos de inteligencia artificial. En Python, bibliotecas como *NLTK (Natural Language Toolkit)*, *spaCy* y *Transformers (de Hugging Face)* proporcionan potentes funcionalidades para el procesamiento de texto, desde la tokenización hasta la implementación de modelos preentrenados. Además, se acostumbra utilizar el entorno de desarrollo basado en la nube, **Google Colab**, debido a su facilidad de uso, recursos de computación gratuitos (como las GPUs) y su integración con bibliotecas de Python. Colab permite trabajar con grandes conjuntos de datos y entrenar modelos avanzados sin necesidad de una infraestructura local costosa.

Reconocimiento de Entidades Nombradas (NER)

El reconocimiento de entidades nombradas (NER, por sus siglas en inglés) es una de las tareas fundamentales dentro del procesamiento de lenguaje natural. Su objetivo principal es identificar y clasificar entidades específicas dentro de un texto, como nombres de personas, organizaciones, ubicaciones, fechas, y otras categorías previamente definidas. Por ejemplo, en la oración "Apple anunció la apertura de una nueva sede en Nueva York", un modelo de NER podría identificar "Apple" como una organización y "Nueva York" como una ubicación. Esta tarea es esencial para aplicaciones como análisis de noticias, extracción de información, y sistemas de búsqueda inteligentes. Un modelo entrenado para NER no transforma ni genera nuevo texto; su única función es detectar y etiquetar aquellas palabras o frases que corresponden a las entidades deseadas. Por cada entidad reconocida por el modelo NER, este también arroja un porcentaje de certeza que determina la probabilidad de que dicha entidad haya sido identificada correctamente, es decir, que haya sido etiquetada de manera adecuada según su tipo (como persona, lugar, organización, fecha, etc.) dentro del texto procesado.

Para lograr que un modelo esté entrenado en una tarea de NER, es fundamental alimentarlo con porciones de texto previamente etiquetadas. Este etiquetado suele realizarse manualmente, donde personas asignan categorías específicas a palabras o frases dentro de un texto. El etiquetado puede realizarse de distintas formas, dependiendo de las necesidades del proyecto. Uno de los enfoques es el esquema **BIO**, que clasifica las palabras según su posición en la entidad:

- **B (Beginning):** Marca el inicio de una entidad.
- **I (Inside):** Indica que la palabra pertenece al interior de una entidad.
- **O (Outside):** Palabras que no forman parte de ninguna entidad.

Por ejemplo, en el texto:
"La Universidad Torcuato di Tella está en Buenos Aires"

El etiquetado bajo el esquema **BIO** sería:

- "La" → **O** (no forman parte de ninguna entidad).
- “Universidad” **B-ORG** (inicio de una entidad, tipo organización).
- “Torcuato” **I-ORG** (interior de una entidad, tipo organización).
- “di” **I-ORG** (interior de una entidad, tipo organización).
- “Tella” **I-ORG** (interior de una entidad, tipo organización).
- "está" → **O** (no forman parte de ninguna entidad).
- "en" → **O** (no forman parte de ninguna entidad).
- "Buenos" → **B-UBI** (inicio de una entidad, tipo ubicación).
- "Aires" → **I-UBI** (interior de una entidad, tipo ubicación).

Este nivel de detalle permite a los modelos aprender patrones más precisos y manejar mejor la complejidad de las relaciones contextuales entre las palabras. La elección del esquema de etiquetado,

como **BIO**, **BIOES** u otros, dependerá de los objetivos específicos del proyecto y la granularidad deseada para la detección de entidades.

Como en todo modelo se necesitan métricas para evaluar su desempeño, pero antes debemos explicar cuatro conceptos importantes a tener en cuenta: Verdaderos Positivos, Verdaderos Negativos, Falsos Positivos, Falsos Negativos. Con la ayuda de Chat GPT armamos el siguiente ejemplo para explicarlos:

Imaginemos que tenes un perro pastor que ayuda a identificar ovejas sanas y separarlas de las enfermas. En este caso, el perro sería como el "modelo" que toma decisiones, y las decisiones que toma las evaluamos con estas métricas.

1. Verdaderos Positivos (VP): son las veces que el perro identifica correctamente una oveja sana. Por ejemplo, si el perro ve una oveja y ladra para decir: "¡Esta está sana!" y efectivamente lo está, eso es un Verdadero Positivo. El perro hizo bien su trabajo.

2. Falsos Positivos (FP): son los errores donde el perro dice que una oveja está sana, pero en realidad está enferma. En este caso, el perro "se equivocó" y clasificó mal. Es como si hubiera dicho: "¡Esta está bien!" y no lo está.

3. Verdaderos Negativos (VN): este es el caso en el que el perro no ladra porque identifica correctamente que la oveja está enferma. Por ejemplo, si ve una oveja enferma y sabe que no está sana, el perro también está haciendo bien su trabajo.

4. Falsos Negativos (FN): aquí es cuando el perro no ladra porque piensa que una oveja está enferma, pero en realidad está sana. Es un error diferente, donde el perro deja pasar una oveja que no está sana.

Si lo aplicamos a modelos de NER estas métricas sirven para evaluar qué tan bien está haciendo su trabajo el modelo al clasificar correctamente las etiquetas (como el perro que decide si una oveja está sana o no).

En base a estos conceptos podemos hablar ahora sí de las métricas que típicamente se utilizan:

- **Precisión (Precision):** Mide cuántas de las entidades identificadas por el modelo son correctas. Es decir, de todas las entidades etiquetadas por el modelo como relevantes, ¿cuántas realmente lo son?

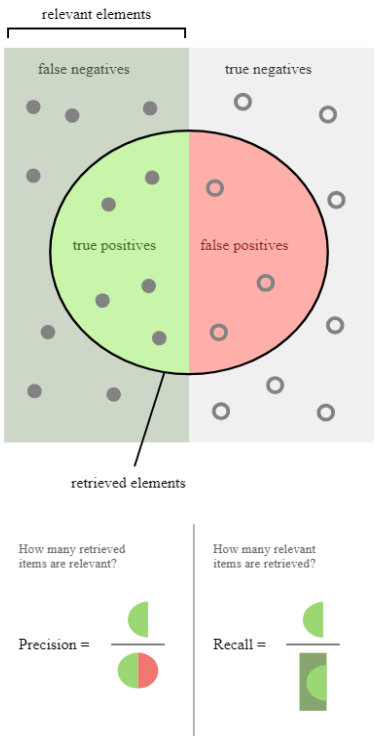
$$Precisión = \frac{Verdaderos\ Positivos\ (VP)}{Verdaderos\ Negativos\ (VN) + Falsos\ Positivos\ (FP)}$$

- **Exhaustividad (Recall):** Evalúa cuántas de las entidades relevantes en el texto fueron identificadas por el modelo. Es decir, de todas las entidades reales en el texto, ¿cuántas logró detectar el modelo?

$$Recall = \frac{Verdaderos\ Positivos\ (VP)}{Verdaderos\ Negativos\ (VN) + Falsos\ Negativos(FP)}$$

- **Puntaje F1 (F1 Score):** Es la media armónica entre la precisión y la exhaustividad. Proporciona un balance entre ambas métricas y es especialmente útil cuando se necesita un equilibrio entre minimizar falsos positivos y falsos negativos.

$$F1\ Score = \frac{2 \cdot Precisión \cdot Recall}{Precisión + Recall}$$



Modelos Generativos

Un modelo generativo es un tipo de inteligencia artificial diseñado para crear nuevos datos similares a los datos en los que fue entrenado. En el ámbito del procesamiento de lenguaje natural, su objetivo principal es generar texto coherente, prediciendo la siguiente palabra en una secuencia dada una serie de palabras previas. Por ejemplo, si se le proporciona la frase "El clima está muy", el modelo podría predecir que la palabra más probable siguiente es "caluroso".

Estos modelos están entrenados en grandes corpus de texto, que pueden incluir fuentes como Wikipedia, libros y páginas web. Durante esta etapa de preentrenamiento, el modelo aprende patrones del lenguaje, como gramática, semántica y relaciones contextuales entre palabras. Posteriormente, pueden ser ajustados (*fine-tuned*) en tareas específicas como respuesta a preguntas, segmentación de texto o generación en un dominio concreto.

La arquitectura de los modelos generativos está basada en **Transformers**, que como ya mencionamos previamente es una de las tecnologías más avanzadas e importantes en el procesamiento de lenguaje natural. Los Transformers emplean **mecanismos de atención**, que les permiten analizar todo el contexto de un texto para identificar cuáles palabras o frases son más relevantes en cada paso de la generación. Esto les otorga la capacidad de manejar dependencias a largo plazo y generar respuestas contextualmente adecuadas y coherentes.

Los modelos generativos suelen evaluarse con métricas como perplejidad (perplexity), que mide qué tan bien un modelo predice una secuencia de texto basada en las probabilidades asignadas a cada posible palabra siguiente. Cuando la tarea del modelo es completar valores en una base de datos, la evaluación es distinta y puede realizarse de manera similar a la de un clasificador, analizando qué proporción de los valores generados coinciden con los valores reales esperados. Esta métrica se conoce como exactitud (accuracy). Por lo general, y sobre todo en el caso de accuracy, los modelos de clasificación se suelen comparar contra modelos aleatorios (que clasifican totalmente al azar, sin entrenamiento) como vara mínima a superar. Por ejemplo, si estamos clasificando entre “gato” y “perro” se esperaría que nuestra accuracy fuera mayor al 50% dado que por debajo de esto sería lo mismo que tirar una moneda y asignar “gato” cuando sale cara y “perro” cuando sale seca. Sin embargo, en el caso de un modelo generativo donde el conjunto de posibles respuestas ya no es binaria sino que abarca todo el vocabulario (incontable cantidad de etiquetas), la evaluación se torna más compleja. Comparar su desempeño con un modelo aleatorio evidencia la dificultad de la tarea: la probabilidad de acertar aleatoriamente es inversamente proporcional al tamaño del vocabulario. Por ejemplo, si hay 50.000 posibles palabras, la probabilidad de acertar al azar sería solo 1/50.000. Esto implica que la vara mínima a superar es casi inexistente y que por ende cualquier modelo generativo funcional debería superarla ampliamente. Por esto cuando hablamos de la accuracy de un modelo generativo pierde sentido comparar contra un modelo aleatorio perdiendo referencia de un mínimo porcentaje a alcanzar.

Una de las razones por las cuales estos modelos pueden tener una baja perplexity o accuracy se debe a que pueden generar información que podría no ser verídica. A esto se lo conoce como alucinaciones (hallucinations) del modelo y puede abarcar desde cosas muy notorias como “Las vacas son de color rosa”, hasta detalles pequeños difíciles de notar como “La penicilina fue descubierta en 1918”. Estas alucinaciones pueden reducirse con algunas estrategias a la hora de confeccionar la solicitud (prompt) que el usuario envía al modelo:

- **Few-shot learning:** Se le proporcionan pocos ejemplos en el input para guiar su desempeño.
- **Prompts diseñados cuidadosamente:** Se construyen preámbulos o instrucciones claras que ayudan al modelo a entender el contexto de la tarea.
- **RAG (Retrieval-Augmented Generation):** Se proveen contextos para que el modelo enfoque qué información utilizar para generar la respuesta.

Retrieval-Augmented Generation (RAG)

La técnica de RAG tiene como objetivo principal **acotar el rango de búsqueda** del modelo generativo. Como ya mencionamos, estos modelos suelen estar entrenados con grandes corpus de datos (Wikipedia, internet, libros, etc.),y enfrentan un desafío importante: al manejar tanta información, las respuestas generadas pueden ser contradictorias o imprecisas, ya que diferentes partes de los datos pueden ofrecer perspectivas diferentes sobre un mismo tema.

Para abordar este problema, RAG permite restringir las fuentes de información que el modelo generativo puede usar. En este enfoque se proporciona al modelo un conjunto limitado de contextos (porciones de texto que pueden llegar a proveer información para completar la solicitud) relevantes junto con el prompt, reduciendo así la probabilidad de errores, contradicciones o información irrelevante.

El proceso de RAG consta de dos etapas principales:

1. **Obtención de contextos (Retrieval):** Un modelo de inteligencia artificial recibe una solicitud del usuario y consulta una base de datos con posibles contextos (por ejemplo: artículos, párrafos o frases) que fue armada a partir de la información que queremos explotar. Este modelo devuelve un conjunto de contextos clasificados según su relevancia para responder la solicitud.
2. **Generación de respuesta:** Los contextos seleccionados son pasados al modelo generativo, que los utiliza para crear una respuesta específica y coherente.

Es importante que los contextos seleccionados sean unos pocos y altamente relevantes, ya que pasar demasiada información puede confundir al modelo generativo y reducir la calidad de las respuestas. Por esto, la etapa de obtención de contextos juega un papel clave en el éxito de la técnica.

Segmentación de Texto

Otra tarea clásica en el procesamiento de lenguaje natural es la **segmentación de texto**. Esta tarea consiste en asignar una o varias etiquetas (categorías) a un texto según su contenido. Por ejemplo, en una tarea de clasificación de correos electrónicos, un modelo podría clasificar un mensaje como "spam" o "no spam". De manera similar, un modelo podría clasificar las reseñas de un restaurante en categorías como "positivo", "negativo" o "neutral".

La clasificación de texto es ampliamente utilizada en diversas aplicaciones, como la organización de contenidos, la moderación de comentarios, la detección de temas y la interpretación de opiniones. Este tipo de tarea requiere que el modelo comprenda tanto la estructura gramatical como el contexto semántico de un texto, lo que hace que los modelos generativos basados en arquitecturas como **Transformers** sean particularmente eficaces en este tipo de tareas.

Los modelos de clasificación de texto pueden entrenarse utilizando enfoques supervisados, donde se les proporciona un conjunto de datos etiquetado (es decir, textos con las categorías ya definidas), o enfoques no supervisados, donde el objetivo es agrupar textos similares sin etiquetas previas dejando que el modelo identifique patrones en común. Tanto en el enfoque supervisado como en el no supervisado, el modelo aprende a reconocer patrones y características en los textos que lo ayudan a predecir las categorías correctas para nuevos textos no vistos.

La evaluación de los modelos de clasificación de texto se realiza comúnmente utilizando la métrica **accuracy**, que mide la proporción de predicciones correctas sobre el total de predicciones realizadas. Esta métrica es especialmente útil cuando las clases están equilibradas (hay una misma cantidad de datos de entrenamiento para cada etiqueta) y el objetivo es maximizar la cantidad de etiquetas correctamente predichas. Además de accuracy, en ocasiones se utilizan otras métricas como **precision** y **recall**, especialmente en casos de clases desbalanceadas, para obtener una evaluación más detallada del rendimiento del modelo.

Scraping a Wikipedia

Una de las técnicas más comunes para obtener datos de internet es a través del scraping de sitios web. El scraping es un proceso mediante el cual se extrae toda la información contenida en el HTML de una página web. Para realizar esta tarea, normalmente se utiliza una URL, que debe estar completa para así redirigirse a la página que se desea scrapear. Existen diversas bibliotecas en distintos lenguajes de programación que permiten llevar a cabo este proceso.

La dificultad de esta tarea radica en dos aspectos principales: primero, en identificar cuáles son las URL que se desean scrapear, y segundo, en que el proveedor de la página web permita realizar el scraping de manera legítima. Muchos proveedores de servicios, como Google, no permiten el scraping directo de sus páginas ya que detectan que demasiadas solicitudes (requests) pueden ser originadas por bots maliciosos cuyo objetivo no es realizar un scraping legítimo, sino interrumpir el servicio para otros usuarios o atacar al proveedor.

Uno de los proveedores que sí permite el scraping de manera libre es Wikipedia. No solo tiene una URL fácilmente accesible para realizar scraping (todas las páginas de Wikipedia comienzan con *https://es.wikipedia.org/wiki/*), sino que también permite un número ilimitado de solicitudes por parte de los usuarios para obtener información. Esto facilita muchísimo la obtención de información de manera automatizada.

Marco metodológico: Desarrollo del Proyecto

Herramientas de gestión del proyecto

Este proyecto se llevó a cabo durante 8 sprints de dos semanas cada uno como la organización de la materia propuso. A lo largo de estos sprints hubo dos herramientas fundamentales para gestionar el desarrollo, ClickUp y Google Drive.

ClickUp es un software específico para la gestión de proyectos que nos permitió organizar los objetivos y los tiempos de cada etapa. Sacamos mucho provecho a su posibilidad de crear listas de tareas con campos como “estado de la tarea”, “persona asignada”, “fecha de entrega” y “documento asociado”. En cada sprint fuimos armando la lista de objetivos para las dos semanas que abarca y así tuvimos claridad en el rol que cada integrante del equipo debía ocupar. Además, hicimos uso del template de reuniones semanales que provee y con esto pudimos dejar para futura referencia un punteo de cada reunión semanal de forma ordenada.

Google Drive es el servicio de nube de Google que permite almacenar archivos y compartirlos sin ocupar espacio local. Este servicio fue elegido por dos razones. Por un lado, al contar con cerca de 30 libros y varios archivos de código se hacía indispensable tener un sistema de almacenamiento en nube para evitar problemas al compartirlos. Por otro lado, al ser un proyecto que involucra Procesamiento del Lenguaje Natural era indispensable hacer uso de Google Colab; herramienta que previamente destacamos por sus recursos de cómputo gratuitos, pero que además se integra perfectamente con Drive al permitir leer y cargar archivos automáticamente.

Desarrollo del proyecto

En esta sección detallaremos cronológicamente el desarrollo del proyecto para que también sirva de guía a quienes quieran replicar esta metodología con otros textos. Omitiremos todo aquello que no dio resultado para facilitar la lectura y lo estaremos discutiendo en la sección discusión para quienes estén interesados en entender con más detalle las alternativas evaluadas.

Fase 1: Preprocesamiento de los libros

Contamos inicialmente con 23 libros (Apéndice A) los cuales debimos inspeccionar y preparar para su futuro uso. En estos libros encontramos dos en español que debieron ser descartados ya que los modelos más accesibles no admiten más de un idioma a la vez y además porque el idioma elegido para este proyecto por quien lo propuso es el inglés. También encontramos que más de la mitad tenían un trabajo de OCR (reconocimiento óptico de caracteres) de baja calidad obstruyendo la extracción de información de estos PDFs.

Por esto decidimos procesar los 21 libros restantes utilizando una librería de Python llamada Tesseract-OCR al ser el mejor servicio gratuito disponible en el momento. Tesseract-OCR utiliza machine learning para reconocer las líneas de texto en una imagen y extraer los caracteres que hay en ella. Para utilizar esta herramienta definimos el script de Python procesamiento_de_textos.py que dado un PDF aplica el OCR página por página. Los resultados obtenidos de este script son guardados de dos formas distintas pero ambas en formato TXT el cuál es mucho más liviano de almacenar y mucho más fácil de leer. La primera forma en que se almacena el resultado es en un archivo donde una página del PDF original equivale a un renglón de nuestro TXT; estos archivos quedan nombrados con el prefijo “pages”. La segunda forma de almacenamiento consta de un archivo donde un renglón del PDF original equivale a un renglón de nuestro TXT; estos archivos reciben el prefijo “lines”. Esta forma particular de almacenar el resultado del OCR en dos archivos distintos cobrará sentido en futuras fases del proyecto.

Fase 2: Reconocimiento de las obras arquitectónicas en los textos

Para poder identificar las obras mencionadas en los libros se nos propuso hacer NER. Como explicamos anteriormente, NER es una tarea de Procesamiento de Lenguaje Natural que dado un texto reconoce ciertas entidades. En nuestro caso, los modelos de NER preexistentes no contemplaban la tarea de reconocer obras arquitectónicas, es decir, no había un modelo definido que ya tuviera una etiqueta específica para estas. Por esto tomamos la decisión de elegir un modelo preentranado que mejor se ajuste a la cuestión y luego agregarle nuestra propia etiqueta **ARCH** (obra arquitectónica).

Elección del modelo

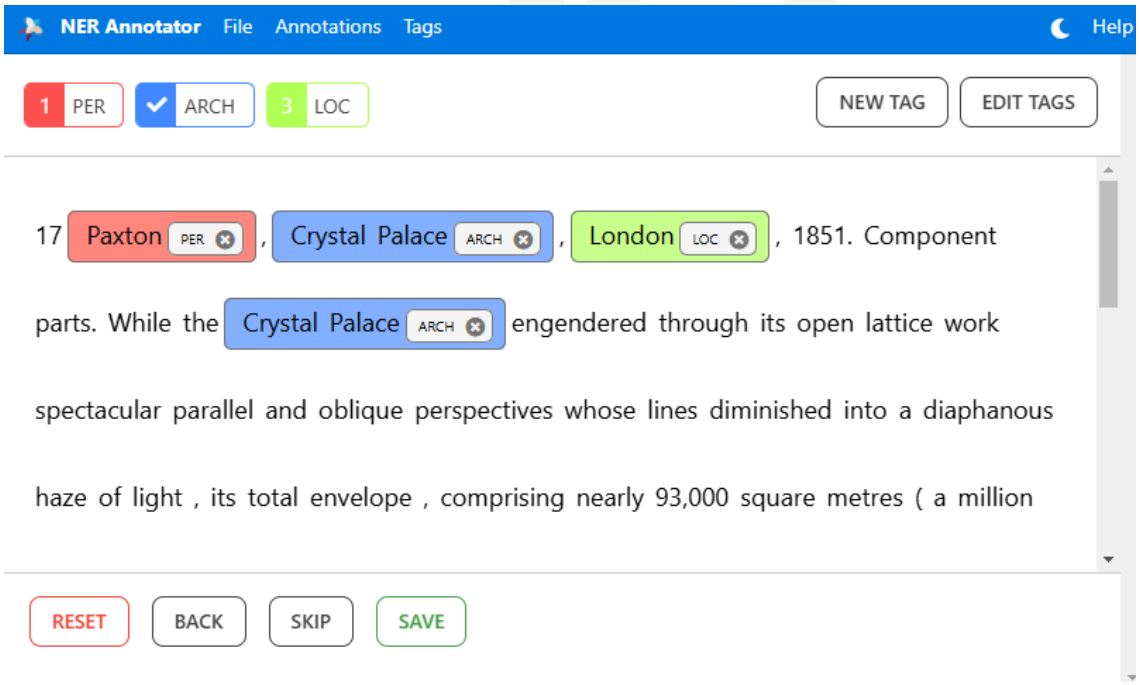
Decidimos utilizar el modelo “Distilbert-NER” por varias razones. Primero que nada, Distilbert es una versión destilada del modelo más popular de transformers llamado BERT. Este, a diferencia de BERT, resulta más fácil de entrenar debido a la menor cantidad de parámetros necesarios para el funcionamiento del modelo lo que lo hace más liviano y por ende rápido. Pero no nos quedamos con la base de Distilbert, sino que en particular investigamos hasta encontrar un modelo específico que ya esté entrenado con la tarea NER, de ahí surge su nombre Distilbert-NER. Por más que no estuviera preparado para reconocer nuestra etiqueta **ARCH**, este ya allanaba el camino resultando en un entrenamiento más rápido al conocer la tarea en sí.

Para poder utilizar y entrenar este modelo decidimos importarlo a nuestro código a través de las funciones que brinda la librería **Transformers de Hugging Face**. Esta librería nos permite cargar el tokenizador, cargar el modelo para ser entrenado, provee la estructura de entrenamiento y la función necesaria para aplicar el modelo ya entrenado sobre nuevos datos. Todo esto la convierte en una pieza clave de la tarea por lo que nos pareció importante mencionarla.

Adaptación de Distilbert-NER

Para enseñarle al modelo a reconocer nuestra etiqueta **ARCH** fue necesario preparar un conjunto de datos de entrenamiento. Estos datos debían ser lo más completos posibles, esto quiere decir, que debían abarcar distintas formas de mencionar las obras arquitectónicas y los distintos estilos de escritura en nuestros textos.

Para generar estos datos de entrenamiento seleccionamos un conjunto de páginas de cada libro. En cada una de estas páginas marcamos las obras arquitectónicas con la etiqueta **ARCH**. Como complemento marcamos a las ubicaciones con la etiqueta **LOC** y a las personas con la etiqueta **PER**. La razón por la cual incluimos las dos etiquetas extras fue para disminuir el riesgo de que el modelo se confunda con otros nombres propios. Este proceso fue realizado manualmente por nosotros a lo largo del trabajo mediante la web gratuita NER Text Annotator, la cual provee una interfaz muy amigable para el etiquetado y luego permite descargar los datos en un formato que facilita su posterior lectura. Para hacer uso de esta web se carga un TXT con el conjunto de páginas a etiquetar, se crean las etiquetas de interés y luego solo basta con elegir la etiqueta y resaltar las palabras que se desean etiquetar. Una vez finalizado el etiquetado se exportan las anotaciones y estas se descargan automáticamente en un formato JSON. En la web misma se puede ver más en detalle el paso a paso del etiquetado en caso de querer replicarlo.



Procesamiento de los datos de entrenamiento

Las anotaciones que recibimos del paso anterior debieron ser procesadas para pasarlas por el modelo en un formato que este las pueda trabajar. Primero que nada, definimos que el esquema interno de etiquetado a utilizar es **BIO**, el cual explicamos brevemente en el marco teórico. Esta elección se basa en que el modelo que utilizamos como base (Distilbert-NER) ya utiliza este esquema desde un principio; utilizar otro esquema implicaría una curva de aprendizaje que puede ser evitada. Con el esquema interno ya definido, el siguiente paso fue procesar las anotaciones mismas. Recordemos que estas anotaciones estaban hechas sobre páginas de los libros y este procesamiento implicó separarlas en oraciones, además de una limpieza de caracteres especiales (como algunos signos de puntuación: **- _ + / “”** …) que podrían dificultar la identificación de las entidades. Una vez procesadas se continúa tokenizándolas para finalmente presentarlas al modelo en un lenguaje que comprende en su entrenamiento.

Como en todo modelo de inteligencia artificial dividimos nuestras anotaciones en dos conjuntos, uno de entrenamiento y uno de validación. El conjunto de entrenamiento representa todos los ejemplos del cual el modelo va a aprender, mientras que el conjunto de validación sirve para mostrar que tan bueno es el desempeño del modelo en un conjunto de datos con los que no entreno. La cantidad de datos por etiqueta que tenemos en nuestro conjunto de entrenamiento y de validación es la siguiente:

Train labels:

O	B-PER	I-PER	B-ARCH	I-ARCH	B-LOC	I-LOC
68902	2776	5169	892	3201	1042	1109

Validation labels:

O	B-PER	I-PER	B-ARCH	I-ARCH	B-LOC	I-LOC
16931	706	1227	195	695	231	241

Entrenamiento

A la hora de realizar nuestro entrenamiento, la elección fue priorizar la exhaustividad (recall) ante todo debido a dos motivos. Primero, como la cantidad de datos por etiqueta presenta tanto desbalance (es bastante desigual), métricas como accuracy no reflejan bien el desempeño del modelo sobre los datos. Segundo, ya que queremos que el modelo reconozca cuantas más arquitecturas posibles, tomamos la exhaustividad como métrica de entrenamiento, a pesar de que esta pueda admitir falsos positivos. Sin embargo, los falsos positivos no presentan un problema para nosotros dado que en un mecanismo posterior estaremos filtrándolos.

El recall obtenido en validación fue de 76%. Cabe destacar que durante la validación en el entrenamiento, nuestra métrica de recall no tiene en cuenta el etiquetado BIO, es decir, que si la etiqueta correcta era B-ARCH y la predicha fue de I-ARCH, esto se traduce en un acierto.

Otro detalle del entrenamiento es que como en todo modelo de inteligencia artificial existen ciertos hiper parámetros que deben ser cuidadosamente elegidos, más allá de los parámetros que los modelos mismos aprenden por sí solos. Para automatizar este proceso y lograr resultados óptimos, utilizamos la librería Optuna que provee las funciones necesarias para encontrar la mejor combinación de hiper parámetros para nuestro modelo. Su implementación fue realizada en un archivo aparte llamado Optuna_NER.ipynb que duplica el código original y le aplica las funciones necesarias.

Ejemplo de las entidades encontradas junto a su porcentaje de certeza:

entity	label	certainty_score
Casa Vicens	ARCH	0.999578
Gaudí	PER	0.9998374
London	LOC	0.9998294
Crystal Palace	ARCH	0.9990508
Paris	LOC	0.9994604
Eiffel Tower	ARCH	0.9995205

Fase 3: Identificación del segmento de texto en que se encuentran las obras

Queremos encontrar en qué partes del texto se ubican las obras para que quienes posteriormente realicen análisis sobre la base de datos puedan medir la importancia de estas. No es lo mismo una obra que aparece en un párrafo, que una que tiene una imágen o hasta su propio título. Para poder encarar este problema, decidimos utilizar un modelo de inteligencia artificial que toma los libros que se encuentran separados por línea (archivos con prefijo *lines*) y clasifica por "Title" (Título), "Paragraph" (Párrafo) y "Caption" (Pie de foto + Notas). Esta última categoría resulta ser la más ambigua de las tres y esto se debe a que los libros no son tan uniformes a la hora de escribir los pie de fotos y notas, haciendo que los formatos sean bastante similares y hasta indistinguibles para el modelo dado que recibe texto plano y no una “foto” de la página.

Elección del modelo

El modelo utilizado fue distilbert-base-uncased al ser más maleable y trabajable para una tarea no tan pesada como esta. Al igual que con la tarea de NER, elegimos el modelo distilbert con la diferencia de que tomamos su versión “base” al no necesitar que esté adaptado a una tarea particular; la habilidad de realizar clasificaciones viene por defecto en el modelo. Por último, la elección de tomar el modelo “uncased” (versión que desprecia mayúsculas) se debe a que no resulta primordial

en esta tarea diferenciar entre minúsculas y mayúsculas, convirtiendo al modelo en algo aún más liviano.

Una vez más al igual que con NER, la librería que utilizamos para poder importar el modelo, el tokenizador y la función de entrenamiento, fue **Transformers de Hugging Face**.

Datos de entrenamiento

Para este modelo fue necesario anotar datos nuevamente de forma manual y que comprenda los distintos estilos de escritura de los libros. Lo que hicimos fue, tomando los archivos de prefijo "lines", elegimos 10 ejemplos de "title", 10 ejemplos de "paragraph" y 10 ejemplos de "caption" por cada libro. En total conseguimos 630 datos de entrenamiento. En el caso de este modelo no fue necesario utilizar una web para anotarlos dada la simpleza de esta tarea. Solo tuvimos que armar un Excel donde en la primera columna está la línea de texto a etiquetar y en la segunda columna la etiqueta correspondiente, y finalmente exportarlo como un CSV.

Entrenamiento

En el entrenamiento de este modelo se utilizó accuracy como métrica al tener las clases completamente balanceadas y ser una tarea sencilla. Los hiper parámetros no pasaron por un proceso de optimización dado que los resultados fueron muy buenos desde el principio y porque decidimos ahorrar el cómputo para otras cosas (recordemos que los recursos de Google Colab son bastante limitados).

Extracto de un texto segmentado:

text	label
Modern Architecture: A Very Short Introduction	Title
Chapter 1	Title
Somewhere between 1910 and 1970, architecture changed. Now that	Paragraph
modern architecture is familiar—and we've seen how it became both	Paragraph
1. Iron Bridge, Ironbridge, Shropshire, UK, completed 1779. A	Caption

Fase 4: Retrieval-Augmented Generation

Con los resultados obtenidos del modelo de NER queremos completar una base de datos (producto final de este proyecto). Para lograr esto empleamos una estrategia basada en Retrieval-Augmented Generation (RAG), donde será necesario identificar los contextos (ventana de palabras anteriores y/o posteriores al objeto de búsqueda) más relevantes para responder las preguntas que completan cada columna de la base de datos.

Cada obra arquitectónica identificada tiene dos posibles tipos contextos. Por un lado están los contextos generados a partir de los libros de la bibliografía. Estos se generan a través de guardar cada página completa por cada aparición de una obra arquitectónica específica. Por el otro, están los contextos obtenidos a partir de scrapeo de wikipedia. Si se encuentra una entrada en Wikipedia relacionada con la obra arquitectónica identificada, se realizará *scraping* sobre dicha página para extraer contextos a partir de sus párrafos. Es importante aclarar que, en caso de no encontrar una entrada específica de la obra en Wikipedia, se buscará la página correspondiente a su autor y se analizará para ver si contiene algún párrafo relevante sobre la obra identificada, ya así utilizarlo como contexto.

Durante la generación de respuestas, se da prioridad a las entradas de Wikipedia al completar campos que requieren datos precisos. Esto se debe a que, aunque los libros pueden contener información valiosa, a veces pueden no ser completamente exactos o estar desactualizados. En contraste, las entradas de Wikipedia suelen estar sometidas a un mayor nivel de revisión y actualización constante.

Dado que en algunos casos las múltiples apariciones de una misma entidad podrían generar demasiados contextos para una sola respuesta, se aplicarán búsquedas semánticas para priorizar y rankear los contextos más informativos. Los rankings de contextos se hacen por separado cuando los contextos son de Wikipedia o de la bibliografía. El desafío principal radica en lograr una alta precisión en la identificación de los contextos adecuados y en desarrollar un modelo que pueda responder correctamente a las preguntas planteadas.

Elección del modelo

El modelo seleccionado para la generación de respuestas fue Flan-T5 debido a su amplia gama de modelos con distintos pesos y su facilidad de implementación. En particular, de todos los modelos de distintos pesos que ofrece Flan-T5, decidimos utilizar su versión “large” (flan-t5-large), esto debido a que esta versión tiene el mejor balance peso-calidad de respuesta. Modelos con mayor cantidad de parámetros también lograban producir respuestas de calidad pero debido a limitaciones de la GPU RAM de Google Colab no se podían realizar muchas solicitudes, mientras que modelos de muchos menos parámetros no producían respuestas de calidad.

Librerías utilizadas

En la obtención de contextos tuvimos que utilizar dos importantes librerías. Para transformar estos contextos en representaciones que puedan ser analizadas con el fin de determinar su relevancia en base a una pregunta dada, elegimos utilizar un **Sentence Transformer**, específicamente el modelo all-mpnet-base-v2 debido a la alta calidad que posee para generar representaciones (embeddings). Para realizar búsquedas semánticas en base a las representaciones generadas por el sentence transformer, nuestra elección fue utilizar faiss, la librería de vanguardia para resolver este tipo de problemas.

Entrenamiento

Decidimos no entrenar por nuestra cuenta ninguno de estos modelos debido a que la capacidad base de estos fue suficiente para cumplir con la tarea del proyecto.

Armado del producto final

Gracias a RAG pudimos responder aquellas preguntas que se planteaban al principio del proyecto: *¿quién diseñó cada obra?*, *¿en qué ciudad y país se encuentra?*, y *¿cuándo comenzó y terminó su construcción?* Todas estas respuestas las almacenamos en una base de datos (una por cada libro) que consta de las siguientes columnas:

- Obra arquitectónica (architecture): indica la entidad identificada en el libro.
- ¿Es una obra? (is_architecture): responde por **sí o no** acerca de que si la entidad identificada es una obra arquitectónica efectivamente.
- Ciudad (city): indica la ciudad en donde se sitúa la obra.
- País (country): indica el país en donde se sitúa la obra.
- Autor (author): indica quién o quiénes construyeron la obra.
- Año de inicio (start_year): indica el año de comienzo de la construcción de la obra.
- Año de finalización (end_year): indica el año de finalización de la construcción de la obra.
- Cantidad en título (qty_tittle): indica la cantidad de veces que la obra aparece en un título.
- Cantidad en párrafo (qty_paragraph): indica la cantidad de veces que la obra aparece en un párrafo.
- Cantidad en referencias y fotos (qty_caption): indica la cantidad de veces que la obra aparece en una referencia o foto. Estas posiciones fueron unificadas en “Caption” dada una alta similaridad en su formato.
- Cantidad no asignada (qty_unassigned):indica la cantidad de apariciones de la obra identificada que no se pudieron determinar en qué parte del texto aparecen.
- Cantidad total (qty): indica la cantidad total de apariciones de la obra identificada.
- Porcentaje de certeza (certainty_score): porcentaje de certeza promedio con el que el modelo identificado todas las apariciones de la obra identificada.
- Tipo funcional de la obra inferido (fuctional_type_infered): indica el tipo funcional que el modelo infirió sobre la obra arquitectónica.
- Tipo funcional de la obra clasificado (fuctional_type_classified): indica el tipo funcional que el modelo eligió para la obra dada una lista cerrada de tipos funcionales.
- Uso de wikipedia (uses_wikipedia): responde por **sí o no** acerca de que si podría pasar que el modelo utilice la página de wikipedia de la obra identificada para completar los valores de la base de datos.

Algunas de las columnas mencionadas conllevan una explicación particular:

- La columna “is_architecture” surge de una necesidad de agregar un filtro extra para los casos en donde el modelo NER etiquetó erróneamente algún dato como ARCH cuando no debería. Para que el modelo generativo pueda determinar si la obra identificada es válida o no, se emplea la técnica de **few-shot**. Esta consiste en plantear una pregunta directa sobre si la supuesta obra

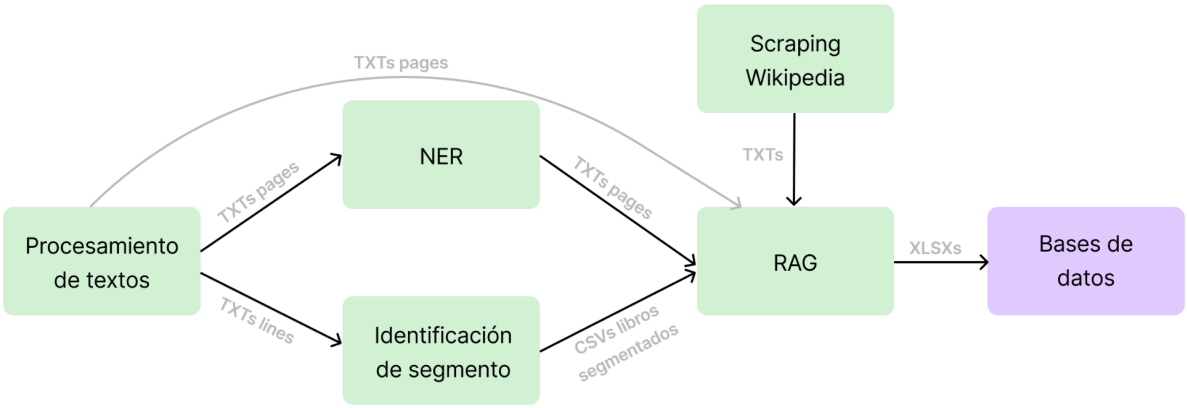
identificada es o no una obra arquitectónica, acompañada de ejemplos que ilustran casos de obras arquitectónicas válidas e inválidas. En el caso de que el modelo responda negativamente en esta columna no se completarán los demás campos relacionados, ahorrando de esta manera grandes cantidades de cómputo (el cual debemos recordar que es escaso).

- El conjunto de columnas con el prefijo "qty" (abreviación de *quantity*, cantidad) son calculadas en base a los resultados obtenidos por el modelo de text classification. Para calcular estos valores, se desarrolló un pequeño código en Python que recorre los libros cuyos renglones ya fueron clasificados (en título, párrafo o caption) buscando y luego contando cada aparición de cada obra según el segmento correspondiente.
- La columna "certainty_score" surge directamente del modelo NER dado que en cada identificación que realiza determina un grado de certeza o probabilidad de que la etiqueta que predijo sobre esa palabra sea efectivamente del tipo correcto.
- La columna "functional_type_inferred" nace de la idea de intentar que el modelo haga una inferencia sobre el tipo funcional que podría asignársele a una obra arquitectónica. Esta inferencia se hace en base a la siguiente definición de tipo funcional provista por Julián Varas: *"A building's functional type is a designation that classifies buildings by their originally intended use or purpose at the time of construction (e.g., house, sanatorium, office). While the functional type is sometimes part of a building's name (e.g., "Farnsworth House" implies "house"), it may also be absent or misleading (e.g., "Seagram Building" does not directly indicate a function, and "Les Arcades du Lac" is residential, not commercial). Additionally, buildings may have overlapping designations (e.g., "house", "residence", or "villa")."*
- La columna "functional_type_classified" es otra forma de abarcar el mismo problema de asignar un tipo funcional a la obra de arquitectura identificada, con la diferencia de que se le pide al modelo que clasifique la obra como alguno de los 134 tipos funcionales propuestos por Julián Varas (Apéndice B).

Ejemplo de base de datos resultante:

architecture	is_architecture	author/s	city	country	start_year	end_year	qty_title	qty_paragraph	qty_caption	qty_unassigned	qty	certainty_score	fuctional_type_infered	fuctional_type_classified	uses_wikipedia
Altes Museum	Yes	Karl Friedrich Schinkel	Berlin	Germany	1845	1845	0	1	4	1	6	0.9836796423	museum	Museum	TRUE
project c	No														
Rue de Rivoli	No														
Palais Mazarin	Yes	Labrouste	Paris	French	1785	1785	0	2	0	0	2	0.9980427027	library	Hospital / Clinic / Dispensary	FALSE

Diagrama de la metodología resultante



Resultados: Métricas

Los resultados finales de cada modelo fueron calculados utilizando la base de datos que recibimos por parte de Julián. Esta base fue realizada por él y sus colaboradores sobre los primeros nueve capítulos del libro: *Kenneth Frampton - Modern Architecture*.

NER

Para evaluar el rendimiento del modelo NER comenzamos identificamos en la base dada aquellas arquitecturas que podrían haber sido encontradas (las cuales presentan al menos una mención directa, no del estilo “In such a climate it fell to Wagner to close the Secession as it had begun, with the vigour of his extremely austere yet elegantly proportioned second villa, built in Hutteldorf in 1912" donde el nombre de la obra, "Villa Wagner", debe ser deducido por uno). Con esto calculamos los porcentajes de acierto (porcentaje de obras reconocidas correctamente de la base dada) según el umbral de certeza de NER elegido. El porcentaje de acierto aumenta a medida que disminuimos el umbral de certeza:

- 90% de certeza: 66% de aciertos
- 70% de certeza: 69% de aciertos
- 50% de aciertos: 69% de aciertos
- 30% de aciertos: 69% de aciertos
- 10% de aciertos: 69% de aciertos

Gracias a esto pudimos ver que más de la mitad de las obras son reconocidas con al menos un 90% de certeza de que efectivamente son una obra arquitectónica.

El mejor modelo NER obtenido se encuentra en el siguiente link.

Text Segmentation

La evaluación del modelo de segmentación se realizó a través de la creación de un archivo de texto que encapsula porciones de todos los libros. Este archivo posee títulos, párrafos y captions (Pie de fotos + Notas), de cada uno de los libros propuestos para realizar este proyecto. Los resultados obtenidos para este modelo fueron de:

- Accuracy: 85%

Esto deja ver que el modelo de clasificación de texto es muy eficaz para distinguir los títulos, párrafos y captions independientemente del tipo de libro que se esté clasificando.

El mejor modelo de segmentación obtenido se encuentra en el siguiente link.

RAG

En la evaluación de este modelo se tomaron las obras mencionadas en la base dada y se realizaron predicciones utilizando los contextos encontrados para estas. Luego, comparamos los valores predichos por el modelo con los valores reales. Los resultados para cada categoría fueron:

- Accuracy Autor: 64%
- Accuracy Ciudad: 67%
- Accuracy País: 57%
- Accuracy Inicio: 41%
- Accuracy Fin: 35%

Si bien los porcentajes de accuracy no son excesivamente altos en todas las categorías, muestran igualmente un buen funcionamiento de las tecnologías generativas en la extracción de resultados.

Cabe destacar que lograr que el modelo pueda generar la información correcta sobre cada campo respecto a una obra arquitectónica depende totalmente de si los datos están o no presentes en los contextos del mismo. Esto quiere decir que es prácticamente imposible que el modelo generativo complete correctamente el valor de la columna en los casos donde no se encuentra una entrada de Wikipedia válida y los contextos obtenidos no presentan información relevante.

Guía para ejecutar la metodología en otros libros de arquitectura

Aquí dejamos una breve explicación de cómo ejecutar la metodología para crear la base de datos de un nuevo libro de arquitectura. Tomamos como ejemplo el ficticio libro *arch_book.pdf*.

1. Subimos a Google Drive, en el espacio "Mi Unidad", la carpeta **ejecucion_metodologia**.
2. Dentro de **ejecucion_metodologia** existe una carpeta **libro** donde vamos a cargar arch_book.pdf (el pdf del libro del que queramos obtener una base de datos).

   Es importante tener en cuenta que, debido a las restricciones de tiempo en Google Colab, los PDFs de libros muy extensos no pueden ser procesados en una sola ejecución. En estos casos, se recomienda dividir el PDF en varias partes y analizarlas por separado.

   El número máximo de páginas que puede procesarse en una sola ejecución varía, pero generalmente está entre 140 y 300 páginas.
3. Regresamos a la carpeta **ejecucion_metodologia** y abrimos el archivo ejecucion_metodologia.ipynb.
4. Dentro del notebook debemos asegurarnos de que estemos utilizando la GPU T4 que Google Colab nos ofrece. Para esto vamos a **Entorno de ejecución** > **Cambiar tipo de entorno de ejecución**. En la ventana emergente, seleccionamos **Acelerador de hardware**: **T4 GPU** y hacemos click en Guardar.
5. Una vez que nos aseguramos de estar corriendo con GPU, en la parte superior de la pantalla, hacemos click en la pestaña que dice **Entorno de ejecución**.
6. Dentro del menú desplegable, haz clic en la opción **Ejecutar todo**.
7. Esto ejecutará la metodología completa y enviará la base de datos creada a la carpeta **resultado** bajo el nombre DB_arch_book.csv. Esto podría tardar entre 2 y 4 horas dependiendo del peso del PDF inicial.
8. Para finalizar vaciamos las carpetas **libro**, **lines**, **pages** y **csvs_text_segmentation**, para evitar problemas en un futuro uso de la metodología.

Discusión: Tecnologías no Utilizadas

En la siguiente sección discutiremos acerca de otras tecnologías que durante el proyecto se probaron pero que no resultaron ser las mejores opciones.

Comencemos por mencionar las alternativas que evaluamos para la tarea de NER. Por un lado se encuentra la librería Spacy, la cual a principios del proyecto pareció una opción muy interesante, pero resultó no ser tan amigable a la hora de personalizar el proceso de entrenamiento y de selección de un modelo base. Por esto terminamos utilizando Transformers de Hugging Face como librería para realizar este proyecto dada su maleabilidad. Por otro lado, el modelo Distilbert-NER no fue la única opción que probamos, nuestra intención en un primer momento fue entrenar un modelo de Bert base que directamente aprenda la tarea junto con nuestras etiquetas personalizadas, pero esto resultó extremadamente difícil de realizar debido a nuestra limitada cantidad de datos de entrenamiento.

Luego, para segmentación de textos encontramos que existen tecnologías de procesamiento por imágenes que permiten analizar una página completa de un PDF en vez del método por renglón que utilizamos. Específicamente probamos el modelo DocLayout-YOLO que prometía reconocer de mejor manera títulos, párrafos, epígrafes y otros segmentos de texto. Sin embargo, al probarlo nos encontramos con un peor desempeño que nuestro clasificador (Apéndice C) y que sin un entrenamiento personalizado este tipo de modelos no podían realizar la tarea de la manera que nosotros necesitábamos.

Un ejemplo de esto se puede ver en la siguiente imagen, donde se puede ver como los últimos tres párrafos son en verdad referencias a arquitecturas las cuales no deberían ser consideradas “plain text” (párrafo). Estas referencias nuestro modelo las identifica correctamente.

Además, el entrenamiento de un modelo del estilo es en un trabajo mucho más complicado al requerir anotar imágenes para su entrenamiento implicando gastar mucho más tiempo del que disponíamos para esta tarea.

Por último, es importante hablar de RAG y por qué Flan-T5 fue nuestra decisión final acerca del modelo generativo a utilizar. A pesar de que existen varias opciones open source como BART, Mistral y LLAMA, decidimos utilizar Flan-T5 por su gran usabilidad y variedad de versiones. El resto de modelos open source implicaban complejos procesos de implementación además de requerir mayor poder de cómputo, lo cual repetimos incontables veces que era un recurso totalmente escaso. Otra de las opciones que tuvimos en cuenta fue la API de OpenAI, pero al no ser gratuita y debido a nuestra intención de siempre utilizar tecnologías de código abierto, decidimos descartar esta opción.

Conclusión

Podemos afirmar que tanto la metodología empleada como el desarrollo del modelo han demostrado ser suficientemente eficaces para la tarea de reconocimiento de obras arquitectónicas. El modelo de *Named Entity Recognition* ha logrado obtener resultados satisfactorios en tiempos significativamente cortos. Sin embargo, su desempeño está altamente condicionado por la calidad y cantidad de los datos de entrenamiento, lo que subraya la importancia de contar con conjuntos de datos amplios y bien estructurados.

En términos de escalabilidad, esta metodología tiene potencial para ser aplicada a infinita cantidad de textos de distintas áreas siempre y cuando se cuente con el tiempo suficiente para preparar datos de entrenamiento y así adaptar un modelo NER. Esto lo convierte en una herramienta dentro de todo flexible, que puede ser aplicada en diversos contextos académicos y de investigación. En cuanto a la escalabilidad en nuestro caso de uso específico, podría ampliarse mediante la incorporación de un mayor número de libros de historia de la arquitectura en los datos de entrenamiento, lo que permitiría mejorar su precisión y adaptabilidad a nuevos textos.

Es importante reconocer que, si bien los resultados obtenidos con el conjunto de datos actual no alcanzan la precisión de un análisis realizado por un experto en el área, la relación tiempo-calidad que ofrece la metodología resulta sumamente ventajosa. Procesar libros de manera automatizada es una solución práctica, considerablemente más rápida y eficiente en comparación con un análisis manual.

Por último, consideramos que esta metodología puede tener un alto valor en aplicaciones institucionales. Ofrece la posibilidad de obtener de manera eficaz bases de datos generadas automáticamente, eliminando la necesidad de invertir grandes cantidades de capital o tiempo en estas tareas. El acceso a estas bases de datos permitiría a los miembros de dichas instituciones realizar análisis complejos de bibliografía específica facilitando la investigación.

Recomendaciones

En el caso de que se quiera extender este proyecto, utilizar esta metodología en otra área o se busquen lograr aún mejores resultados, proponemos estos posibles siguientes pasos.

- El gran problema de los modelos entrenados en tareas de NER es que no tienen la habilidad de reconocer menciones indirectas de entidades, es decir, aquellas referencias que no mencionan explícitamente el nombre de una obra arquitectónica, sino que la describen o la aluden de manera contextual. Por ejemplo, la oración "*In such a climate it fell to Wagner to close the Secession as it had begun, with the vigour of his extremely austere yet elegantly proportioned second villa, built in Hutteldorf in 1912*" menciona indirectamente "Villa Wagner" pero el modelo no es capaz de deducirlo.
  Una solución que podría llegar a ser factible sería realizar la identificación de las obras de arquitectura de manera generativa, abandonando totalmente al modelo NER. Esto permitiría que se analice todo el contexto y que se genere el nombre de la obra a pesar de que esta no fuese nombrada textualmente. Tomar este camino requeriría mayor trabajo sobre el modelo generativo debiendo entrenarlo en esta tarea específica.

- Otro aspecto que se podría tener en cuenta en el caso de querer mejorar los resultados de la identificación y recuento de entidades es lograr utilizar modelos generativos que puedan corregir los errores que vienen desde la fase inicial de procesamiento de los textos por OCR. Muchas veces estos lectores fallan en reconocer caracteres especiales o tienen errores notorios debido a la baja calidad de los PDFs (por ejemplo, al ser libros escaneados y no digitales).

- En nuestra discusión sobre las distintas tecnologías que probamos mencionamos que el proceso de segmentación de textos podría haberse hecho mediante análisis de imágenes. Justamente esto es lo que venimos a proponer. En el caso que se esté interesando en preparar datos de entrenamiento para un modelo de detección de imágenes y segmentación de texto, invitamos a investigar sobre tecnologías como DocLayout-YOLO que permiten la posibilidad de realizar ese tipo de trabajo.

- La clasificación en tipos funcionales fue otro de los problemas que enfrentamos al desarrollar la base de datos final. Nuestra intención no se limitaba a realizar una simple inferencia y clasificación, sino también a lograr que el modelo pudiera justificar sus decisiones contando su cadena de razonamiento. Este objetivo no representa una gran dificultad técnica pero si lleva tiempo. Consideramos que el modelo Flan-T5 tiene el potencial de cumplirlo siempre y cuando se someta a un entrenamiento más exhaustivo, lo que requiere armar una gran cantidad de ejemplos para alimentar al modelo.

- Retomando un poco lo expuesto al final de nuestra hipótesis, la adjetivación que da el autor con respecto a una obra es un aspecto que no se está teniendo en cuenta pero que podría serlo. Este proyecto se focalizó en los aspectos sustantivos del discurso únicamente pero podría haberse ampliado a lo adjetivo a través de técnicas de NLP como el análisis de sentimiento.

- Como toda esta metodología no está prevista para ser completamente utilizada por alguien que no tiene conocimientos de NLP e IA, proponemos realizar una interfaz de usuario amigable que permita cargar pdfs y recibir bases de datos sin mayor dificultad.

Referencias bibliográficas

- H. (2023, Sep 6). *Building your own custom Named Entity Recognition (NER) model with SpaCy v3: A step-by-step guide*. Medium. Retrieved from https://mjghadge9007.medium.com/building-your-own-custom-named-entity-recognition-ner-model-with-spacy-v3-a-step-by-step-guide-15c7dcb1c416

- Data Knows All. (2024, March 5). *What is NER and how does it work?*. Recuperado de https://dataknowsall.com/blog/ner.html

- Arunmozhi. (n.d.). *NER annotator: How to annotate data for NER*. Recuperado de https://arunmozhi.in/ner-annotator/

- Hugging Face. (n.d.). *Text classification*, Recuperado de https://huggingface.co/docs/transformers/en/tasks/sequence_classification

- GeeksforGeeks. (2023, Jan 06). *Web scraping from Wikipedia using Python – A complete guide*. Recuperado de https://www.geeksforgeeks.org/web-scraping-from-wikipedia-using-python-a-complete-guide

- Hugging Face. (n.d.). *Flan-T5*. Recuperado de https://huggingface.co/google/flan-t5-large

Apéndices

Apéndice A: Lista de los libros propuestos para hacer este proyecto

1. Adam Sharr - Modern Architecture: A Very Short Introduction.
2. Adolf Behne - The Modern Functional Building.
3. Alan Colquhoun - Modern Architecture.
4. Bruno Zevi - The Modern Language of Architecture.
5. Francis D. K. Ching, Mark Jarzombek, Vikramaditya Prakash - A Global History of Architecture.
6. Harry Francis Mallgrave - Twentieth Century Architecture.
7. Henry-Russell Hitchcock and Philip Johnson - The International Style: Architecture Since 1922.
8. Kenneth Frampton - Modern Architecture: A Critical History.
9. Martin Filler - Makers of Modern Architecture, Volume I.
10. Martin Filler - Makers of Modern Architecture, Volume II.
11. Martin Filler - Makers of Modern Architecture, Volume III.
12. Nikolaus Pevsner - Pioneers of Modern Design: From William Morris to Walter Gropius.
13. Nikolaus Pevsner - The Sources of Modern Architecture and Design.
14. R. Stephen Sennott - Encyclopedia of 20th-Century Architecture, Volume I.
15. R. Stephen Sennott - Encyclopedia of 20th-Century Architecture, Volume II.
16. R. Stephen Sennott - Encyclopedia of 20th-Century Architecture, Volume III.
17. Reyner Banham - Theory and Design in the First Machine Age.
18. Sigfried Giedion - Building in France, Building in Iron, Building in Ferroconcrete.
19. Sigfried Giedion - Space, Time and Architecture: The Growth of a New Tradition.
20. Vittorio Magnago Lampugnani - Encyclopedia of 20th Century Architecture.
21. Walter Curt Behrendt - Modern Building: Its Nature, Problems, and Forms.

Libros en español no tenidos en cuenta:

22. Leonardo Benevolo - Historia de la Arquitectura Moderna Volumen I.
23. Leonardo Benevolo - Historia de la Arquitectura Moderna Volumen II.

Apéndice B: Lista de los 136 tipos funcionales

- Abbey
- Customs Office
- Altar - Asylum / Orphanage
- Studio - Auditorium
- University Auditorium
- Lecture Hall
- Town Hall or Municipality
- Bank (branch or headquarters)
- Public Baths
- Library
- Winery
- Stock Exchange
- Bowling Alley
- Cabins
- Bathhouse
- Retreat House
- Rest Home
- Casino / Gambling House
- Cemetery
- Server Center
- Telephone Center

- Civic Center
- Cultural or Exhibition Center
- Convention Center
- Exhibition Center
- Innovation Center
- Research Center
- Logistics Center
- Congress or Convention Palace
- Sports Center
- Cinema
- Club
- Clubhouse
- Professional Public School
- Holiday Camp
- Summer Camp
- Congress, Parliament, or Assembly
- Municipal Council
- Clinic
- Convent
- Post Office
- Court of Justice
- Crematorium
- General Warehouse
- Train Depot
- Specialized Education
- Embassy or Consulate
- Facility
- Primary School
- Secondary School
- Fire Station
- Subway Station
- Police Station
- Service Station
- Railway Station
- Indoor Sports Stadium
- Baseball Stadium
- Football Stadium
- Olympic Stadium
- Radio Broadcasting Studios
- Factory
- Commercial Arcade
- Gym
- Farm / Stable
- Hangar
- Hybrid
- Hippodrome
- Hospital / Clinic / Dispensary
- Hostel
- Hotel
- Vertical Garden
- Church
- Printing House
- Greenhouse / Botanical Garden
- Courts
- Kindergarten
- Laboratory
- Master Plan
- Slaughterhouse
- Media Library
- Memorial
- Wholesale / Central Market
- Retail Market
- Mosque
- Ministry or Government Department
- Observation Tower
- Monastery
- Motel
- Museum
- Observatory
- Government Administrative Offices
- General Offices
- Exhibition Pavilion
- Meeting Pavilion
- Government Palace
- Pantheon
- Beach Resort
- Parking Garage
- Parliament or Assembly
- Park
- Water Park
- Penitentiary
- Pool / Swimming Facility
- Power Plant
- Recycling Plant
- Bridge
- Recycling Facility
- Shelter
- Renovation
- Government Residence and Palace
- Restaurant / Bar / Coffee Shop
- University Services (for students, etc.)
- Shopping Mall
- Silo
- Synagogue
- Supermarket
- Theater
- Airport Terminal
- Bus Terminal
- Passenger Port Terminal
- Retail Store / Showroom
- Department Store

- Single-Brand Store
- University or College
- Velodrome
- Collective Housing
- Single-Family Housing
- Housing for Pensioners
- University Housing
- Zoo

Apéndice C: Prueba con DocLayout-YOLO

En esta imagen extraída del libro de Kenneth Frampton, se puede observar como el modelo DocLayout-YOLO no era funcional a nuestro problema dado que tanto párrafos como pies de foto los clasifica como “plain text”.