Aplicación de Machine Learning para el análisis de normativa en el Boletín Oficial de la Ciudad de Buenos Aires

1. Introducción

Trabajo como bibliotecario en el Consejo Profesional de Arquitectura y Urbanismo y una de mis principales tareas consiste en mantener actualizada la normativa de la Ciudad de Buenos Aires referida al ejercicio profesional de la arquitectura y el urbanismo. Esta abarca, pero no se limita, a construcción, planificación urbana, habilitaciones comerciales, impacto ambiental, seguridad e higiene, planes de evacuación, uso del espacio público, etc.

Para llevar a cabo dicha tarea realizo un relevamiento diario del Boletín Oficial de la Ciudad de Buenos Aires. Cada vez que se detecta una norma relevante, ésta es catalogada y posteriormente se producen piezas informativas para difundir en redes sociales, la web institucional y el newsletter de la biblioteca.

El objetivo general de este trabajo es entrenar un modelo de Machine Learning para que detecte normativa relevante para el ejercicio profesional de la arquitectura y el urbanismo a partir del análisis de los ejemplares del Boletín Oficial de la Ciudad de Buenos Aires.

Los objetivos secundarios son:

1) Que el modelo aprenda el marco normativo que rige en la Ciudad de Buenos Aires a partir del año 2018, cuando se aprobaron los nuevos códigos de edificación y urbanístico que rigen en la actualidad.

2) Que el modelo aprenda a detectar cambios en la normativa vigente (modificaciones, sustituciones, derogaciones, etc).

3) Que el modelo sea capaz de detectar figuras normativas nuevas que no formen parte de sus datos de entrenamiento.

2. Marco conceptual

Este trabajo es la cuarta iteración de una serie de prototipos para automatizar el relevamiento normativo del Boletín Oficial:

- **Versión 1 – Búsqueda por palabras clave:** script en Python que recorre PDFs y exporta coincidencias a Excel.
- **Versión 2 – Palabras clave + verbos de acción normativa:** reducción de falsos positivos incorporando verbos como "aprueba", "deroga", etc.
- **Versión 3 – LLM vía API:** incorpora el uso de un modelo de lenguaje para evaluar la relevancia de los fragmentos candidatos.

Esta experiencia sirvió como base para el desarrollo del pipeline de construcción del dataset.

ChatGPT se utilizó como tutor y co-diseñador del pipeline. Para la realización de la Diplomatura se creó un Proyecto, el cual se alimentó con el plan de estudios, los apuntes provistos por los docentes y un prompt inicial (que hace las veces system prompt) en que le asignó el rol de tutor para realizar el acompañamiento de la cursada. Se creó un chat para cada uno de los módulos y para el Proyecto Final.

Por otro lado, se utilizó NotebookLM como herramienta de estudio y generación de resúmenes teóricos. Se crearon sendos cuadernos con bibliografía sobre Machine Learning y Natural Language Processing. A partir de estos, se generaron informes sobre temas específicos (por ejemplo: TF-IDF, umbrales, métricas de desempeño, modelos BERT, etc.) con el objetivo de generar un marco teórico que acompañe cada paso del proyecto.

3. Metodología

Dentro del Proyecto de ChatGPT creado para la Diplomatura, se destinó un chat exclusivo para la realización del Trabajo Final. Se inicializó con el prompt que se incluye como Anexo del presente informe. Se utilizó GPT 5 (posteriormente GPT 5.1) Thinking en modo pensamiento ampliado.

3.1 Construcción del dataset

El dataset está constituido por un conjunto de datos etiquetados para clasificar fragmentos “pertinentes” (normativa relevante para el ejercicio profesional de la arquitectura) vs. “no pertinentes”, a partir de los ejemplares en formato PDF del Boletín Oficial de la Ciudad de Buenos Aires.

3.1.1 Criterio de partición de dataset

Dado que el objetivo final es entrenar un modelo que aprenda a detectar tanto las modificaciones en la normativa vigente como la aparición de nuevas normas, se decidió realizar un *split* temporal y mantener una flecha del tiempo para medir una generalización real (el modelo se entrena con pasado y se evalúa en futuro).

- **Train (2018 → 1º semestre de 2024 inclusive)**
  Se eligió comenzar en el 2018 porque en ese año se sancionaron los nuevos códigos de Edificación y Urbanístico de la Ciudad. Esto produjo un efecto derrame sobre las normas relacionadas, ya que debieron ser adaptadas al nuevo marco normativo.

- **Validación** (**2º semestre 2024)**
  Utilizado para comparar modelos, seleccionar hiperparámetros y definir umbrales.

- **Test (enero → noviembre 2025)**
  Usado para métrica final.

3.1.2 Extracción de candidatos

Se desarrolló un script en Python que toma cada ejemplar del Boletín Oficial en el que se publicó normativa considerada relevante y realiza las siguientes operaciones:

1) Extrae texto por página, realiza una limpieza mínima y segmenta en oraciones.

2) Realiza la búsqueda de las siguientes palabras clave:

    - **KEYWORDS:** listado de términos que pertenecen al área temática que se desea detectar (por ejemplo: “código urbanístico”, "código de edificación", “autorización de actividad económica", "impacto ambiental", etc.).

    - **PATRONES_NORMAR:** listado de patrones regex de normas, cuyas modificaciones se desean detectar (por ejemplo: r"[Dd]isposici[oó]n(?: [Nn]°?)? ?526(?:[-/]?GCABA)?[-/]?DGFYCO[-/]?24", r"[Dd]ecreto(?: [Nn]°?)? ?51/18", r"[Ll]ey(?: [Nn]°?)? ?6\.099", etc.)

    - **VERBOS_ACCION:** listado de patrones regex con verbos que denotan una acción normativa (r"\b[Mm]odifica\b", r"\b[Mm]odifícase\b", r"\b[Mm]odificar\b", r"\b[Aa]prueba\b", r"\b[Aa]probar\b", r"\b[Aa]pruébese\b",etc.).

3) Cada vez que se produce un match, se extrae el fragmento de contexto (± 2 oraciones ~400–600 palabras).

4) Evalúa el contexto y guarda el fragmento si cumple con alguna de las siguientes condiciones:
    - Contiene KEYWORDS, o
    - Contiene PATRONES_NORMAR, o
    - Contiene (VERBOS_ACCION and PATRONES_NORMAR) or (KEYWORDS and PATRONES_NORMAR).
5) Aplica dedupliacación para remover duplicados exactos y fragmentos donde se solapan oraciones.
6) Guarda los datos en un archivo csv con los siguientes campos:
    - fragmento: texto a etiquetar.
    - label: vacío (se completa manualmente).
    - origen_flags: `VERBO;KEYWORD;NORMAR.
    - Trazabilidad: `fecha`, `anio`, `origen_pdf`
    - split: `train`, `test`, `val`

Notebook → 01_Dataset_Builder_v1_.ipynb

3.1.3 Curación, filtrado y etiquetado

Luego de realizar la extracción de los candidatos, se obtuvo las siguiente cantidad de registros:

- train: ~27.250
- validación: ~2.000
- test: ~3.150

A continuación se llevó a cabo la limpieza de los datasets para reducir su tamaño. En el caso del conjunto *train* se aplicó un filtrado adicional, utilizando marcadores que suelen indicar articulado/decisión:

- “RESUELVE”, “DISPONE”, “DECRETA”, “ARTÍCULO 1°”,

- y fórmulas legislativas (“La Legislatura… sanciona con fuerza de Ley”).

En el caso de los conjuntos *val* y *test* se realizó un muestreo aleatorio simple para representar la prevalencia natural del período.

Para realizar la edición de los fragmentos (limpieza fina) y el etiquetado (1/0) se desarrolló un editor visual con la biblioteca Gradio. Finalmente, los datasets quedaron del siguiente tamaño:

- ***train*:** 2.372 registros, de los cuales 718 son positivos (30,27%)
- ***val*:** 500 registros, de los cuales 80 son positivos (16%)
- ***test*:** 1.200 registros, de los cuales 43 son positivos (3,58%)

Notebooks → **02_Triage_v1_etiquetas.ipynb**
03_Editor_Gradio_BO_CSV.ipynb

3.2 Baseline con Machine Learning

El objetivo de esta etapa es validar un clasificador supervisado de pertinencia normativa usando modelos clásicos de Machine Learning sobre TF-IDF. Se priorizó la sensibilidad (*recall*) por encima de la precisión (*precision*) debido a que, para el objetivo que se pretende alcanzar, es tolerable tener una cantidad razonable de Falsos Positivos, pero sería inaceptable que se produzcan Falsos Negativos.

3.2.1 Preprocesamiento

- Texto crudo (sin stemming/lemmatización).
- TF-IDF word n-grams (1–2), min_df=3, max_df=0.9, sublinear_tf=True, max_features=100k.
- Para Random Forest: reducción de dimensión con SVD (300 comp.).
- class_weight='balanced' en modelos lineales.

3.2.2 Modelado y validación (sin fuga)

- **Pipelines:** TfidfVectorizer → clasificador.
- **Modelos:** Logistic Regression (LR), LinearSVC (SVM), ComplementNB (NB), RandomForest (RF).
- **Selección de HP:** GridSearchCV 5-fold estratificado sobre *train* con scoring AUC-PR (AP) y AUC-ROC; refit="ap".
- Se reentrenó cada mejor pipeline en todo *train*.

3.2.3 Calibración de umbral en *val*

- Dos puntos operativos por modelo:
  - t_F2: maximiza F-beta (β=2) → prioriza recall.
  - t_R@X: primer umbral con Recall ≥ objetivo (configurable).

Notebook → 04_Baseline_BO_CABA_TFIDF_4modelos.ipynb

3.3 Fine-tuning con modelo de lenguaje

El objetivo de esta etapa es evaluar si un modelo de lenguaje preentrenado puede superar al mejor modelo clásico (SVM + TF-IDF) en la tarea de clasificación binaria de fragmentos del Boletín Oficial. El foco se mantuvo en priorizar el *recall*, es decir, minimizar la cantidad de Falsos Negativos y tolerar cierto nivel de Falsos Positivos.

Se seleccionó RoBERTalex[1], un modelo de lenguaje basado en Transformer para el idioma español y especializado en textos legales de España y la Unión Europea. Se basa en el modelo base RoBERTa y ha sido pre-entrenado con el Spanish Legal Domain Corpora.

3.3.1 Arquitectura y entrenamiento

- Tokenización con el tokenizer propio de RoBERTalex.
- Longitud máxima de secuencia: hasta 512 tokens.
- Entrenamiento con Trainer de Hugging Face sobre GPU en Google Colab.

[1] https://huggingface.co/PlanTL-GOB-ES/RoBERTalex

- Optimización con entropía cruzada, usando una configuración estándar de fine-tuning para clasificación de texto.
- El modelo se entrenó durante un número reducido de épocas, suficiente para que converjan las curvas de entrenamiento/validación sin sobreajuste evidente.

3.3.2 Selección de umbrales en val

El modelo produce una probabilidad (p(\text{pertinente})) para cada fragmento. Sobre el conjunto de validación se calcularon dos umbrales operativos:

- **Umbral t_F2:** prioriza recall, pero con algo de control de precisión.
- **Umbral t_R@X:** fuerza un recall muy alto en *val* (≈0,95), aceptando más falsos positivos.

Ambos se aplicaron posteriormente al conjunto de test para evaluar el rendimiento real.

Notebook → 05_FineTuning_v1_ES_Legal.ipynb

4. Resultados

4.1 Métricas evaluadas

- Precision
- Recall
- F1 Score
- Matriz de confusión
- AUC-ROC
- AUC-PR (se incluyó debido a que en test la clase positiva es el 3,58% del total)

4.2 Resultados del baseline

Modelo	Threshold	Precision	Recall	F1 Score	Matriz de confusión				AUC-ROC	AUC-PR
					TP	FN	FP	TN		
LR	t_F2	0.267	0.954	0.417	42	2	115	1001	0.983	0.863
	t_R@X	0.181	0.977	0.306	43	1	194	922	0.983	0.863

NB	t_F2	0.484	0.727	0.581	32	12	34	1082	0.968	0.752
	t_R@X	0.205	0.931	0.336	41	3	159	957	0.968	0.752
RF	t_F2	0.295	0.886	0.443	39	5	93	1023	0.968	0.740
	t_R@X	0.098	1.0	0.179	44	0	403	713	0.968	0.740
SVM	t_F2	0.666	0.863	0.752	38	6	19	1097	0.985	0.880
	t_R@X	0.180	0.977	0.304	43	1	195	921	0.985	0.880

SVM (t_F2) es el modelo que mostró el mejor equilibrio con una buena cobertura (Precision: 0,666 y Recall: 0,863), una baja carga de revisión (19 FP) y solo 6 falsos negativos. Asimismo, tuvo el mejor desempeño en la curva Precision-Recall (0,880), en tanto que la curva AUC-ROC mostró una excelente capacidad de discriminación (0,986).

Si bien LR (t_R@X), RF (t_R@X) y SVM (t_R@X) tuvieron una mejor cobertura (casi el 100% de TP y casi ningún FN), esta se produjo a costa de una cantidad elevada de FP.

Los resultados obtenidos con el pipeline TF-IDF + SVM (t_F2) demostraron que el problema puede ser resuelto a través de Machine Learning clásico y se pueden utilizar como valores de referencia para realizar una comparación con los resultados del fine-tuning de un modelo de lenguaje.

4.3 Resultados del fine-tuning con modelo de lenguaje

Modelo	**Threshold**	**Precision**	**Recall**	**F1 Score**	**Matriz de confusión**				**AUC-ROC**	**AUC-PR**
					TP	**FN**	**FP**	**TN**		
RoBERTalex	t_F2	0.35	0.795	0.486	35	9	65	1091	0.976	0.738
	t_R@X	0.382	0.772	0.511	34	10	55	1101		

4.4 Comparación con el baseline SVM

- La precisión de SVM es sustancialmente mayor (0,666 vs 0,35 - 0,382)
- El recall también es algo superior (0,863 vs 0,772 - 0,795)
- El F1 Score de SVM es claramente más alto (0,752 vs 0,486 - 0,511)

- Los valores de las curvas AUC-PR y AUC-ROC de SVM también son superiores (0,880 vs 0,738 y 0,985 vs 0,976 respectivamente)

En síntesis, para este problema concreto, con el dataset disponible y la configuración probada, el modelo clásico SVM + TF-IDF tuvo un mejor desempeño que el modelo de lenguaje RoBERTalex fine-tuneado.

5. Análisis crítico

Aunque RoBERTalex está preentrenado en textos legales y administrativos, su fine-tuning no logró superar al SVM. A continuación se enumeran las principales hipótesis para explicar este resultado:

1) **Relación entre el tamaño de modelo y del dataset:** RoBERTalex es un modelo pre-entrenado con cientos de millones de parámetros, por lo tanto necesita una cantidad importante de ejemplos etiquetados para especializarse en una tarea nueva. El dataset utilizado es pequeño, con unos pocos miles de ejemplos, con pocos positivos en *val* y *test*. En cambio, SVM + TF-IDF tiene la capacidad de aprender bien con menos datos cuando las diferencias entre clases se pueden capturar con patrones de palabras.

2) **Naturaleza de la etiqueta "pertinente":** la pertinencia del fragmento etiquetado no depende solo de que el texto sea jurídico, sino también de que pertenece a un área de conocimiento muy específica. Las *keyword* utilizadas generan pistas léxicas fuertes (palabras y expresiones "gatillo") que un modelo TF-IDF + SVM captura muy bien mediante pesos altos en esas features. En cambio, en un modelo de lenguaje transformer grande, estas se pueden "diluir" si no tiene suficientes ejemplos etiquetados para aprender exactamente esos patrones.

3) **Especialización legal general vs. subdominio CPAU:** RoBERTalex está especializado en el lenguaje legal y administrativo de España y la Unión Europea. El dataset de entrenamiento abarca un subdominio muy específico, con una jerga que es diferente a la utilizada en España para el mismo campo de conocimiento. Esto implica que la especialización previa del modelo ayuda a entender la sintaxis y el estilo jurídico, pero no garantiza que capture automáticamente el criterio de relevancia para este nicho, especialmente con pocos ejemplos etiquetados.

6. Conclusiones

Los resultados muestran que, en las condiciones actuales, el modelo clásico SVM + TF-IDF es la mejor opción operativa y supera en todas las métricas a un modelo de lenguaje especializado. Este resultado de ninguna manera cierra la discusión. En un futuro, con un dataset más grande, se podría repetir la evaluación y verificar la validez de las hipótesis planteadas en el apartado anterior.

Para la labor diaria de relevamiento del Boletín Oficial se diseñó un pipeline compuesto por los siguientes notebooks:

- **BO_SVM:** extrae candidatos del ejemplar del Boletín Oficial cargado, los analiza con el pipeline TF-IDF + SVM entrenado para determinar su relevancia, muestra los resultados de forma legible (tabla ordenada por probabilidad) y los guarda en un archivo CSV
- **BO_SVM_feedback:** interfaz gráfica en Gradio que permite visualizar y revisar las predicciones de BO_SVM. Guarda los datos en CSV master para futuros re-entrenamientos.
- **BO_SVM_retrain:** re-entrena el modelo TF-IDF + SVM con nuevos datos y calcula un nuevo umbral t_f2.

Notebooks →	**06_BO_SVM.ipynb**
	07_BO_SVM_feedback.ipynb
	08_BO_SVM_retrain.ipynb

A nivel personal y profesional, la realización de este proyecto resultó muy enriquecedora. Antes de comenzar la Diplomatura creía que estaba fuera de mi alcance entrenar modelos de Machine Learning o de lenguaje. Sin embargo, también aprendí que es imprescindible capacitarse en dichos temas, al menos en sus conceptos fundamentales. De lo contrario, el desarrollo del proyecto recae exclusivamente sobre el LLM, no se tiene control del mismo y esto puede derivar en resultados poco satisfactorios o, directamente, en un fracaso.

Anexo - Prompt de inicio

<introducción>

Estoy cursando la Diplomatura en Inteligencia Artificial aplicada a entornos digitales de gestión en la Universidad de Buenos Aires y mi proyecto de trabajo final consiste en entrenar un modelo para que procese la normativa publicada en el Boletín Oficial de la Ciudad de Buenos Aires y determine cual es “relevante” para el ejercicio profesional de la arquitectura y el urbanismo. Además quiero que el modelo “aprenda” a detectar nueva normativa.

El profesor estuvo de acuerdo con mi idea e indicó que el paso lógico y más potente es entrenar un modelo de clasificación supervisado. Agregó que un modelo supervisado, alimentado con mis propios ejemplos, va a aprender los patrones específicos de la jerga legal y administrativa del Boletín Oficial de CABA que son importantes para los arquitectos.

</introducción>

<hoja_de_ruta>

A continuación me recomendó la siguiente hoja de ruta:

##0. Creación del dataset

- La creación del dataset es la parte fundamental del proyecto. Mi conocimiento como experto en la materia es el recurso que va a hacer que el modelo funcione bien.

- Punto de Partida: Para obtener un primer modelo funcional (un proof of concept), te recomiendo arrancar con un dataset de 300 a 500 ejemplos en total. Esto te va a permitir validar si el enfoque funciona sin invertir meses en etiquetado.

- Modelo robusto: Para un sistema que quieras poner en producción y que tenga un alto grado de confianza, apuntá a un objetivo de 1.500 a 2.500 ejemplos. Con este volumen, un buen modelo puede empezar a generalizar de manera muy efectiva.

- Balance del Dataset: el balance entre clases ("pertinente" vs. "no pertinente") es muy importante. El escenario ideal es un dataset balanceado, es decir, 50% de ejemplos pertinentes y 50% de no pertinentes. Esto ayuda al modelo a no sesgar hacia la clase mayoritaria. En la práctica, es casi seguro que vas a generar muchos más ejemplos "no pertinentes". No hay problema si el desbalance no es extremo (ej. 60/40 o 70/30). Si te encontrás con un desbalance muy grande (ej. 90/10), vas a tener que usar técnicas

específicas durante el entrenamiento (como class_weight en librerías como Scikit-learn) o métricas de evaluación más adecuadas que la simple exactitud (accuracy), como el F1-Score, el AUC-ROC o la matriz de confusión.

##1. Baseline con Machine Learning Clásico

- Vectorización: Convertí tus textos en números. TF-IDF (Term Frequency-Inverse Document Frequency) es el estándar de oro para esto y es muy fácil de implementar con Scikit-learn.

- Clasificador: Usá ese vector TF-IDF para entrenar un clasificador simple como:

- Regresión Logística: Rápida, interpretable y muy efectiva para clasificación de texto.

- Support Vector Machines (SVM): A menudo ofrece un rendimiento ligeramente superior.

- Naive Bayes: Un clásico para texto que funciona muy bien.

Usá Scikit-learn para implementar un pipeline de TfidfVectorizer + LogisticRegression. Medí su rendimiento. Te va a sorprender lo bien que funciona.

##2. Fine-Tuning de un Modelo de Lenguaje

Este es el siguiente nivel y probablemente el que te va a dar la mayor precisión, especialmente para detectar esas "figuras normativas nuevas".

Una vez que tengas el baseline, investigá cómo hacer fine-tuning de un modelo como BERT en español usando librerías como Hugging Face Transformers con tu dataset.

En lugar de usar un LLM gigante como GPT-4o/5 a través de una API, tomás un modelo pre-entrenado más pequeño (y de código abierto) y lo re-entrenás ligeramente con tu dataset etiquetado. El modelo aprende a especializarse en tu tarea específica.

En principio te recomiendo estos modelos:

- BERT para español: Existen versiones de BERT pre-entrenadas específicamente en español. dccuchile/bert-base-spanish-wwm-cased es un excelente punto de partida.

- DistilBERT: Una versión más pequeña y rápida de BERT, ideal para empezar y si los recursos computacionales son una limitación.

Ventaja: Este enfoque entiende el contexto y la semántica de una manera que TF-IDF no puede. Si una nueva ley usa terminología novedosa pero dentro de un contexto semántico similar a las que ya clasificaste como "pertinentes", el modelo tiene una alta probabilidad de clasificarla.

Al re-entrenar un modelo con tus ejemplos, le estás "enseñando" qué significa la relevancia en el dominio específico de la arquitectura y el urbanismo en CABA. El modelo aprende a ponderar ciertas palabras y frases de una manera que un modelo genérico nunca podría.

Para detectar figuras normativas nuevas, un modelo bien generalizado (entrenado con un dataset diverso y de buen tamaño) debería poder identificar un texto como "pertinente" incluso si la estructura exacta de la norma es inédita, siempre que el contenido semántico se alinee con lo que ha aprendido.

</hoja_de_ruta>

<instrucciones>

- Debés guiarme durante el desarrollo del modelo.

- Debemos documentar cada paso que demos, ya que al profesor le interesa más el proceso que el resultado final que obtengamos. De todas formas, quiero aspirar a lograr un MVP.

- Para el desarrollo del proyecto vamos a usar Google Colab y los archivos se van a almacenar en Google Drive. Deseo que esté en la nube porque voy a trabajar desde diferentes máquinas. Finalmente quiero poder subir el modelo a GitHub.

- Tené en cuenta que este es mi primer proyecto de esta naturaleza y que no tengo conocimientos de desarrollador, apenas conocimientos básicos de Python.

- Tampoco quiero que me lleves de la mano, vamos a trabajar en equipo.

</instrucciones>

Dame tu opinión sobre el plan. No inicies ninguna acción.

Luego de acordar con ChatGPT el plan de trabajo, le pasé mi propuesta para la construcción de los datasets con el siguiente prompt:

Plan para armar el dataset

1. Consideraciones iniciales

- La normativa se publica diariamente en el Boletín Oficial de la Ciudad de Buenos Aires en un documento PDF.

- En el año se publican miles de normas, pero las que se refieren al ejercicio profesional de la arquitectura no llegan al centenar en ese período. Debemos tener en cuenta este factor a la hora de obtener un dataset balanceado.

2. Plan de extracción de datos

- Tengo los ejemplares de los Boletines Oficiales de los últimos 5 años en los que se ha publicado normativa relevante. Esos son los archivos que vamos a utilizar para extraer los datos.

- Mi idea es un split temporal (train años 2018–2023, val año 2024, test año 2025) ya que el modelo debe aprender a detectar tanto las modificaciones que se van produciendo en el tiempo como las nuevas normas que surjan relacionadas con la temática de interés.

- Tengo una lista de Python con keywords que nos van a ayudar a hacer un filtrado inicial. Es ineficiente analizar todas y cada una de las normas que se publican. Es un archivo que contiene 3 listas:

- NORMAS: lista regex con nombres de normas que refieren al ejercicio profesional, que actualmente están vigentes y que debemos detectar normas posteriores que las modifiquen y/o deroguen.

- TEMAS: términos clave relacionados con el ejercicio profesional de la arquitectura y el urbanismo.

- ACCIONES_NORMATIVAS: verbos que denotan una acción normativa.

- Vamos a crear un script en Python que localice NORMAS, TEMAS y las combinaciones de estas con ACCIONES_NORMATIVAS (estas últimas solas no nos interesan) en el texto del Boletín Oficial y nos lo devuelva con el contexto que lo rodea. NORMAS + ACCIONES_NORMATIVAS o TEMAS + ACCIONES_NORMATIVAS indican mayor posibilidad de relevancia.

(Luego te paso el archivo “keywords” para que lo analices)

3. Script de obtención de candidatos:

- Actualmente uso un script de Python para filtrar candidatos con las keywords mencionadas en el punto anterior, que luego reviso manualmente. Te lo voy a pasar para que lo analices y lo adaptes a nuestro proyecto.

Quiero que me des tu opinión del plan. No empieces a hacer nada hasta que definamos claramente el pipeline.