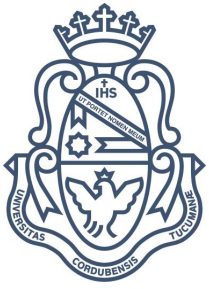
Facultad de Matemática, Atronomía,
Física y Computación

Universidad Nacional de Córdoba



Sistema de recomendación para textos legales

Tesis

que para obtener el título de

Licenciado en Ciencias de la Computación

presenta

Agustín Aldo Capello

Director: Laura Alonso Alemany

Córdoba, Argentina 2018

Agradecimientos

A mis padres, por brindarme la posibilidad de realizar mis estudios.

A Laura, sin ella y su tiempo, entusiasmo y guía detallada, este proyecto no hubiera sido posible de realizar.

A Maico, Ariel, Salomé, Nacho, Jeremías, Eduardo, Alejandro por los recuerdos compartidos y a todos los que me brindaron una mano a lo largo de la carrera.

Resumen

Los Sistemas de Recomendación han mostrado ser un componente importante y hasta imprescindible en varias plataformas.

Su principal atractivo reside en que ofrecen información relevante para el usuario de forma activa, acerca de la base de datos en cuestión, sin necesidad de que el mismo tenga conocimiento sobre los artículos recomendados o la consulta a realizar. La valía de esta información proviene del análisis previo de los datos, y su posible relación con usuarios.

En el presente trabajo realizamos una solución para el desarrollo de un sistema de recomendación de documentos de texto el cual se lo instancia al dominio legal/jurídico, utilizando el corpus de leyes de Argentina, accesible desde la página web oficial `www.infoleg.gob.ar`.

Realizamos la investigación y desarrollo de algunos motores para recomendación de texto, junto a una plataforma para visualizar recomendaciones. Posteriormente se analizan cualitativamente los resultados obtenidos en cada caso.

Palabras Clave: Procesamiento de Lenguaje Natural, Análisis de Datos, Sistema de recomendación basado en contenido, Doc2Vec, TF-IDF, Redes Neuronales, Texto legal.

Clasificación (ACM CCS 2012):

- **Applied computing** – Law
- **Computing methodologies** – Natural language processing

Abstract

Recommender systems have been an important and even essential component in several platforms.

Its main attractive is that it gives relevant information to the user in an active way, about the database in question, without the need for it to have knowledge about the recommended articles or the query to be made. The value of this information comes from the previous analysis of the data, and its possible relationship with users.

In the present work we made a solution for the development of a recommendation system of text documents which we instantiate to the legal/juridical domain, using the corpus of laws of Argentina, accessible from the official website `www.infoleg.gob.ar`.

We carry out research and development of some engines for text recommendation, along with a platform to view the recommendations. Subsequently, the results are analyzed qualitatively in each case.

Keywords: Natural Language Processing, Data Analysis, Content-based recommender system, Doc2Vec, TF-IDF, Neural Networks, Legal text.

Índice general

Capítulo 1

Introducción y Motivación

Los sistemas de recomendación son ampliamente usados en la actualidad y son de especial importancia, no sólo por dar sugerencias acertadas para los intereses de un usuario, sino también porque la calidad de estas recomendaciones contribuyen a que el mismo permanezca en la plataforma consultando por contenido, entre otras cualidades.

En la práctica legal, el acceso digital a documentación jurídica (leyes, códigos, jurisprudencia, entre otras), es un importante recurso. Facilitar el acceso a esa información es crítico, por la gran cantidad de datos. Implica velocidad, exhaustividad en la búsqueda o relevancia, entre otras variables. Tal acceso puede hacer una gran diferencia entre un éxito o un fracaso judicial, además de reducir significativamente el esfuerzo requerido para obtener la información relevante al querer llevar a cabo una acción.

En los últimos años han crecido en número y capacidades las iniciativas comerciales que proveen acceso inteligente a la información o incluso servicios más proactivos, e.g. automatismos para tratar casos fuertemente tipificados.

Sin embargo, la mayor parte de estas iniciativas se han desarrollado para el entorno de Estados Unidos o la Unión Europea, mientras que los desarrollos para Argentina y su región son mucho más superficiales.

El presente proyecto, se ubica dentro del área de Procesamiento

de Lenguaje Natural. En particular, en el área de aplicación de tecnologías de inteligencia artificial a textos legales de Argentina, aunque también es instanciable/adaptable a cualquier conjunto de textos. Se implementa un recomendador junto a una plataforma accesible desde el navegador para realizar consultas, la cual ofrece recomendaciones de normas, a partir de texto de normas, como también a partir de otros textos de carácter legal proveídos por el usuario.

1.1. Descripción de los casos de uso

El caso de uso primario que tendrá el sistema, se dará a través de una interfaz web y consistirá en brindar recomendaciones de normas (leyes, decretos, resoluciones, etc.) relacionadas a ciertas otras normas existentes o textos nuevos. La forma de visualización puede ser simple o distinguiendo tipos de norma.

Es también viable su integración a la plataforma InfoLEG `www.infoleg.gob.ar` [1], utilizando el *back-end* del proyecto. InfoLEG es hoy en día la principal plataforma oficial de consulta de normas nacionales argentinas, utilizada por profesionales del ámbito legal.

Entre casos de uso secundarios, se encuentra la recomendación de normas a partir de otros tipos de texto, no necesariamente normas. Por ejemplo, se pueden recomendar leyes semejantes a artículos periodísticos del dominio legal, como los publicados en `comercioyjusticia.info` [2], o también fragmentos de contratos, textos con inclinación legal, como *tweets* u opiniones de profesionales del ámbito. Un desafío residirá en que los textos recomendados sean realmente adecuados, lo cual se evaluará con usuarios posteriormente a la implementación, o utilizando adecuación semántica respecto al origen.

Capítulo 2

Nociones Preliminares

2.1. Sistemas de recomendación

Los sistemas de recomendación son una sub-clase de los sistemas de filtrado de información. Se encargan de remover información redundante o no deseada de un flujo de información, utilizando métodos semiautomáticos o computarizados, previo a la presentación de la información al usuario.

Un sistema de recomendación busca predecir el "puntaje" o preferencia que un usuario le daría a un artículo o ítem de una plataforma.

Se han vuelto cada vez más populares en la última década, y son utilizados en gran variedad de áreas, incluyendo películas, música, noticias, libros, artículos de investigación, consultas de búsqueda, redes sociales, y productos en general. Son ampliamente usados en la actualidad en plataformas como NetFlix, Last.fm, Pandora, entre muchas otras.

2.1.1. Clases de sistemas de recomendación

Distintos tipos de sistemas de recomendación nacen a partir de distintos enfoques de su diseño, de acuerdo a qué y cómo tratan y usan la información de la base de datos de la plataforma, para dar las reco-

mendaciones. Existen tres enfoques principales: *filtrado colaborativo*, *filtrado basado en contenido* e *híbridos*.

Los métodos de *filtrado colaborativo* están basados en la recolección y análisis de grandes volúmenes de información relacionada a comportamientos, actividades y preferencias de usuarios de la plataforma, seguido de la predicción en cuanto a qué preferirá un usuario basado en la similitud con otros usuarios.

Este enfoque es ampliamente utilizado en plataformas con usuarios registrados, las cuales alojan información sobre ellos, proporcionada por su actividad o bien por indicaciones explícitas del usuario. Una ventaja de éste método, es que no depende del análisis del contenido de los ítems a recomendar de la plataforma, y puede dar recomendaciones acertadas sobre artículos complejos (como películas) sin requerir un "entendimiento"de los mismos [3].

Otro enfoque común es el de *filtrado basado en contenido*, el cual realiza la predicción de los ítems a partir de un análisis (generalmente pre-computado) de información del contenido de los artículos de la plataforma (como su descripción, y otros metadatos) como así también las preferencias del usuario y su historial, los cuales pueden o no existir. Particularmente en diarios (recomendación de noticias en base a la cual se está leyendo) o en el caso de InfoLEG, no necesariamente habrá usuarios registrados, por lo que las recomendaciones serán basadas puramente en el análisis de contenido.

Es posible combinar ambos y más enfoques en un mismo sistema, dando lugar a *sistemas híbridos*. Los cuales también pueden combinar otros sistemas de recomendación como demográficos o basados en conocimiento. Varios estudios empíricos han comparado el rendimiento de estos tres enfoques y demuestran que los métodos híbridos proveen recomendaciones más acertadas. Netflix es un caso de éxito de estos sistemas.

2.2. Medidas de semejanza textual

La semejanza entre textos o similitud semántica, en el área del procesamiento del lenguaje natural, es la magnitud o métrica que existe entre dos o más palabras o términos, obtenida a partir del grado de similitud o parecido entre sus significados o contenidos semánticos.

En la práctica, consta de un modelo matemático calculado a partir de un conjunto de documentos o términos. El mismo es usado para determinar la fuerza con la cual están interrelacionados semánticamente dos unidades de lenguaje o conceptos, a pesar de su diferencia sintáctica, es decir, la idea de distancia entre ellos está basada en el parecido de sus significados.

La similitud semántica sólo incluye relaciones del tipo "es un", a diferencia de la relación semántica que incluye cualquier relación entre dos términos. Por ejemplo "auto" es similar a "autobús" pero también esta relacionado a "carretera" y "manejar".

A continuación explicamos brevemente como lograr esta medida para el conjunto de documentos, utilizando los métodos **Tf-Idf** y **Paragraph Vector**.

2.2.1. Tf-Idf

Del inglés *Term frequency – Inverse document frequency* Tf-Idf es una medida numérica hallada a partir de un conjunto de documentos D para un término t en un documento $d \in D$, que expresa cuán relevante es t para el documento d en función del conjunto de documentos. t puede consistir de una o más palabras consecutivas (*n-grama*) de d.

El valor tf-idf aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia de la palabra en la colección de documentos, lo que permite manejar el hecho de que algunas palabras son generalmente más comunes que otras.

Durante el texto llamamos "modelo", con el mismo nombre, refiriendo al grupo de estas medidas calculadas y ordenadas a partir de los

datos y con ciertos criterios de entrenamiento, junto a otros atributos o meta-datos generados.

Justificación matemática

Sea t un término en un documento d del conjunto de documentos D. Tf-idf es el producto de dos medidas, frecuencia de término (tf) y frecuencia inversa de documento (idf). Existen varias maneras de determinar el valor de ambas. En el caso de $tf(t, d)$ es posible usar la frecuencia bruta de t en d, aunque también es posible usar frecuencias binarias, logarítmicas, normalizadas, etc. Ejemplificando la última:

$$tf(t, d) = \frac{f(t, d)}{max\{f(t, d) : t \in d\}}$$

Donde el denominador es el máximo valor de frecuencia para un término en el documento. De esta forma tf toma valores entre 0 y 1.

idf, por otro lado, es una medida que representa si el término es común o no, en D. Una forma de calcularlo es de la siguiente manera:

$$\text{idf}(t, D) = \log \frac{|D|}{1 + |\{d \in D : t \in d\}|}$$

donde $|X|$ representa la cardinalidad del conjunto X. Luego, la medida tf-idf se calcula como:

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

Finalmente, a partir de los cálculos, es posible obtener la matriz tf-idf M, la cual posee $|D|$ filas y $|V|$ columnas donde V es el vocabulario de términos obtenido, teniendo entonces:

$$M_{ij} = tfidf(t_j, d_i, D)$$

donde $i = 1,..,|D|$ y $j = 1,..,|V|$

Luego de este proceso de entrenamiento, para calcular las similitudes entre los ítems (y así recomendaciones) se utiliza una medida de

similitud vectorial. En general es usada (y en particular en el proyecto) la similitud del coseno [4]. Puede ser entendida básicamente por la división entre el producto punto de dos vectores y el producto de las normas entre ellos. En símbolos:

$$\text{sim} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \ \mathbf{A}, \mathbf{B} \text{ vectores.}$$

2.2.2. Document Embeddings con Doc2Vec

La representación numérica de documentos de texto (e.g. vectores reales) conforma una tarea desafiante dentro del aprendizaje automático.

Paragraph Vector (o más popularmente conocido como **Doc2Vec**) es un conjunto de técnicas para representar documentos como vectores de longitud fija y baja dimensionalidad (conocidos también como document embeddings).

Fue presentado por **Mikolov & Le** [5] en 2014 en su paper "Distributed Representations of Sentences and Documents". Mikolov, es también uno de los autores de **Word2Vec** [6].

Doc2Vec no es el único método para crear documents embeddings. Sin embargo, ha habido afirmaciones recientes de que doc2vec supera a otros métodos [7].

Para comprender doc2vec es antes necesario comprender word2vec, ya que el primero es una extensión del segundo.

Los métodos basados en word2vec tienen como objetivo computar representaciones vectoriales de palabras (también conocidas como word embeddings). Esta representación puede ser creada usando alguno de los dos algoritmos o modelos incorporados: Continuous Bag-of-Words (**CBOW**) y **Skip-Gram**.

Doc2Vec, por consiguiente, posee dos algoritmos para obtener los embeddings: **PV-DM** (*Paragraph Vector - Distributed Memory*) y **PV-DBOW** (*Paragraph Vector - Distributed Bag of Words*). Cada uno surge de la extensión de los algoritmos wor2vec anteriormente mencionados, respectivamente. Es decir, PV-DM es una adaptación

de CBOW de word2vec, y PV-DBOW lo es de Skip-gram. Dejamos conjuntamente investigación al lector en el artículo [8].

PV-DM

Word2Vec consiste de una red neuronal con tres capas: una de entrada (input), una oculta (o hidden layer) y una de salida (ouptut).

La idea de CBOW, es aprender representaciones de palabras que puedan predecir una palabra dadas las palabras que la rodean. O dicho de otro modo, de actuar como una memoria que recuerde qué es lo faltante (en este caso una palabra) dado un determinado contexto (conjunto de palabras).

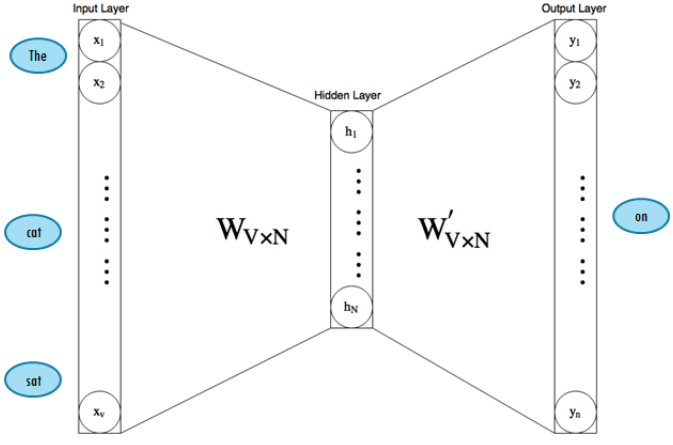


Figura 2.1: Red Neuronal ilustrativa en Word2Vec.

Observemos un ejemplo, en la figura 2.1. Supongamos que se tiene la oración de entrada: “The cat sat on the mat”. El objetivo es aprender la representación de las palabras “the”, “cat”, “sat”, etc.

Para este fin, la red neuronal intenta aprender *features* (representaciones), i.e. matrices de pesos W y W', mirando las palabras en una ventana dentro de la oración, como por ejemplo teniendo {“The”, “cat”, “sat”} como contexto e intentando predecir la siguiente palabra,“on”. (ver fig. 2.2)

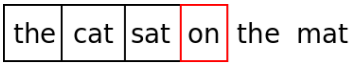


Figura 2.2: Ventana, contexto y palabra objetivo.

Inicialmente, el contexto será dado como instancia de entrenamiento a la capa de entrada como señales mediante un vector con ceros excepto con un 1 en la posición correspondiente a la/s palabra/s del contexto, en la capa de entrada. Esta explicación es simplificada y busca captar el concepto de entrenamiento, la estructura real de la red neuronal es más compleja, teniendo una capa de entrada más extensa de acuerdo al largo prefijado de la ventana.

Notar que se tienen $|V|$ neuronas en la capa de entrada y salida, donde V es el vocabulario del corpus, N es el largo del vector fijado de antemano, el cual usualmente oscila entre 100 y 300.

Luego de ciertas operaciones matemáticas al propagar las señales de entrada por la red neuronal, se obtiene un vector resultante en la capa de salida.

Si fuera la primer iteración, sólo se obtendría "ruido"(i.e. valores no representativos pues la red no se ha entrenado). Pero lo que se realiza ahora, es comparar este resultado con el realmente deseado (objetivo), el cual consiste de un vector con ceros excepto con un 1 en la posición de la palabra objetivo, "on".

A partir de esta comparación, se obtiene una medida de error, y se propaga hacia atrás el error desde la capa de salida para balancear las matrices de pesos de modo que la probabilidad de salida de "on"se maximice, en comparación con otras palabras en el vocabulario.

Como el procedimiento de entrenamiento repite este proceso en un gran número de oraciones, los pesos W y W' se "estabilizan". Luego, W contiene las representaciones vectorizadas de palabras.

Extendiendo el método CBOW a Doc2Vec (PV-DM, figura 2.3), se agregan nodos de entrada adicionales (contexto adicional), un nodo por documento (digamos, identificados inicialmente por los tags: 0, 1, ..., $|Corpus| - 1$). Por lo tanto se extiende aún más la capa de entrada.

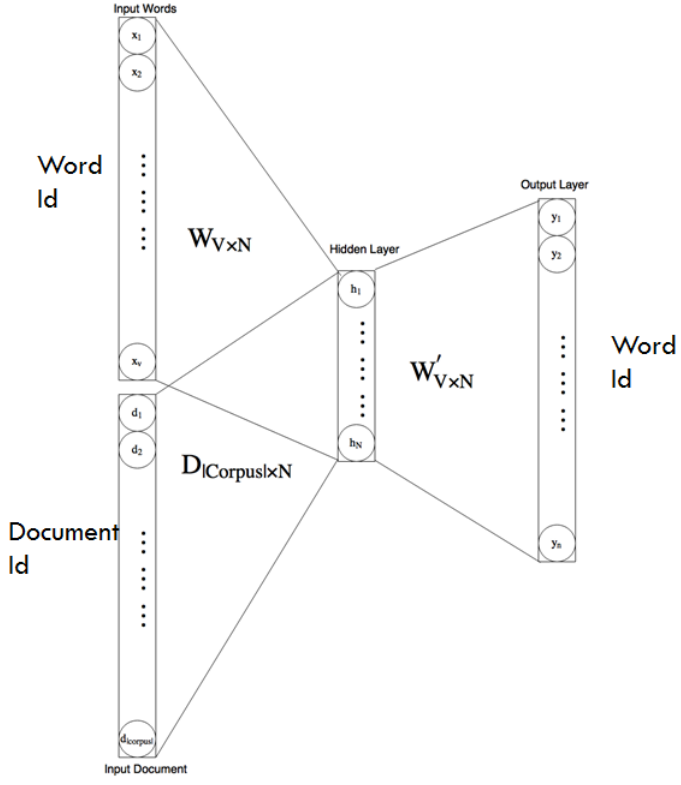


Figura 2.3: Red neuronal ilustrativa en PV-DM

Luego se ejecuta el algoritmo de la misma forma descrita en CBOW, con la señal correspondiente al tag del documento con el cual se está entrenando, para el cual pertenece la oración seleccionada.

Una vez finalizado el proceso de entrenar contextos de palabras junto con el id del documento, se terminan obteniendo en la matriz D document embeddings y en la matriz W word embeddings.

Mediante medidas de similitud, como la del coseno, se hallan los vectores más semejantes a uno fijado, tanto en W, como en D (de interés).

PV-DBOW

El modelo de Distributed Bag of Words (DBOW) es ligeramente distinto (o lo "opuesto") al modelo PV-DM. El modelo DBOW "ignora" las palabras de contexto en la entrada, pero obliga al modelo a predecir palabras muestreadas aleatoriamente del documento (dentro de la "ventana"), en la salida. Para el ejemplo anterior, digamos que el modelo está aprendiendo a través de la predicción de 4 palabras muestreadas. Entonces, para aprender el vector de documento, se muestrean 4 palabras de {the, cat, sat, on, the, mat}, como se muestra en el diagrama 2.4, repitiendo este proceso un gran número de veces análogamente al anterior proceso, para varias muestras en el documento.

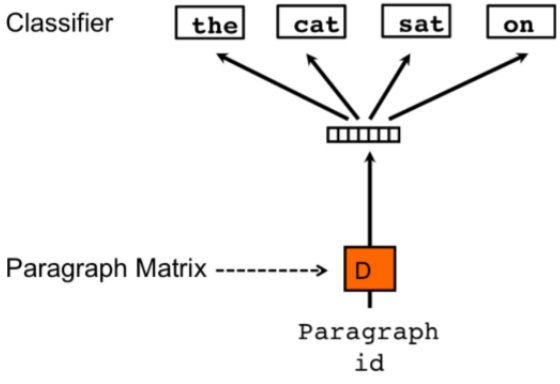


Figura 2.4: PV-DBOW simplificado.[1]

[1]https://medium.com/scaleabout/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

2.3. Trabajo Relacionado

Previo al diseño e implementación del sistema, se han investigado otros proyectos similares existentes en la familia de soluciones, de distintas fuentes internacionales. Se presentan brevemente los resultados.

2.3.1. Otros recomendadores de textos legales

Towards a Legal Recommender System

Entre otros trabajos realizados en el ámbito de recomendación de textos legales, encontramos el caso de **Winkels et al.** [9]. Su objetivo es el de recomendar otras fuentes de leyes relevantes y normas escritas oficiales, en base al documento que tenga en foco el lector (particularmente en el portal `www.rechtspraak.nl`).

Se pone especial énfasis al caso de documentos de jursiprudencia, y la obtención de recomendaciones acertadas sobre normas oficiales que sean relacionadas y convenientes en base al texto, aunque no sean citadas directamente, como también otras fuentes de información parlamentaria.

En especial, tengamos en cuenta la diferencia en el sistema jurídico tanto europeo como anglosajón, con el sistema argentino. En el primero, la aplicación de las normas se da principalmente a través de la jurisprudencia (fallos previos de órganos judiciales), basándose menos en las leyes, lo cual asigna mayor importancia a las recomendaciones.

Para lograr el objetivo, su primer paso es crear una red (network o grafo dirigido con pesos en sus aristas) entre decisiones de la corte (jurisprudencia) publicadas en el portal oficial, y las normas correspondientes citadas en el texto. También se agregan al grafo las referencias existentes entre normas. Por lo tanto cada nodo será un documento de jurisprudencia o norma, y cada arista representa la frase "refiere a". Para lograr capturar las referencias citadas en el texto se utilizan expresiones regulares. Se calcula su *peso* en base a la cantidad de veces que se repite en el texto: $W = \frac{1}{n}$ donde $n = \#$repeticiones de la ref.

Cuando un usuario ingresa a un artículo, el sistema chequea si el último aparece o no en la red construida. En caso afirmativo, el sistema obtiene el sub-grafo (también llamado *ego-graph*) definido en base a cierto peso de tolerancia de distancia a partir del nodo documento del lector. Notar que éste peso es un número real entre 0 y 1, y cuanto menor sea más nodos filtrará. Los nodos de ésta red serán los artículos relacionados a considerar brindados por el recomendador.

La forma de evaluar el sistema se realizó a través de la consulta con usuarios profesionales (e.g. abogados, y otros.).

En la segunda parte de la investigación se busca encontrar los motivos por los cuales un profesional cita a una norma, y encontrar grupos de referencias que tengan características comunes en base a los datos. Para ello se utiliza aprendizaje automático no-supervisado, particularmente *clustering* de referencias usando distintos *features* de las mismas.

A Document Recommendation System Blending Retrieval and Categorization Technologies

Asimismo, se investigó la solución propuesta por **Al-Kofahi et al.** [10]. En él, se plantea que el hecho de obtener recomendaciones para *trabajadores del conocimiento* dista bastante de la idea de dar recomendaciones de productos a consumidores. Además de motivos de privacidad profesional, principalmente por los conocidos problemas de "cold-start" y "shopping for children" de enfoques colaborativos.

Se describe el sistema, llamado **ResultsPlus**, el cual usa una combinación de *information retrieval* y *machine learning*.

Las recomendaciones son generadas usando un proceso de dos etapas: *generación* seguido de *optimización*. En el primer paso, se genera una lista de recomendaciones usando similitud basada en contenido, con el sub-sistema **CaRE**. En la segunda etapa, las recomendaciones son re-rankeadas en base al comportamiento del usuario y datos del uso de documentos.

CaRE se encarga de la extracción de *features* como palabras, pa-

res de palabras, y otros como números clave y citaciones. Combina varios algoritmos de clasificación con pesos asignados, consistiendo de los módulos de Espacio vectorial, Bayesianos (e.g. Naïve Bayes) y KNN. Puesto a que se dispone de varias bases de datos, por ejemplo de tipos ALR (American Law Reports), AMJUR (American Jurisprudence), entre otras, se realiza posteriormente una normalización de los puntajes obtenidos de cada conjunto.

La etapa de optimización de puntajes, en esencia consiste de la estimación del ratio de cliqueo (CTR) para cada recomendación reranqueando la lista antes obtenida. El algoritmo utiliza en su mayoría datos de usuarios y su actividad.

Los resultados obtenidos por el sistema han sido convincentes y se observó cómo la adición de los datos de usuarios mejora el ranking de recomendaciones.

2.3.2. Recomendadores de código abierto

Se realizó la investigación de repositorios de código libre, en la plataforma `github.com` con intención de analizar proyectos ya concluidos relacionados. Entre diversas opciones, hemos encontrado las siguientes fuentes, las cuales en sus repositorios poseen sistemas de recomendación para textos basándose en su contenido. Aunque no ha sido reutilizado explícitamente ninguno de ellos, han servido como referencia y aprendizaje hacia el desarrollo del recomendador de este proyecto.

Para ejecutar y realizar pruebas con los repositorios (todos en gran parte escritos en lenguaje Python) se crearon entornos virtuales e instalado sus requerimientos en cada caso.

- **Content-based Recommendation Engine [11]:**

  Consiste en un motor de recomendaciones usando el vectorizador Tf-Idf de Scikit-Learn. Consiste de un proyecto simple en el cual los datos iniciales se cargan a través de un archivo csv (cada línea un ítem), con poca flexibilidad para cargar distintos tipos

de textos y corpus largos. Entre otras limitaciones que hemos encontrado:

- El cálculo de semejanzas usando las similitudes del coseno para conjuntos de datos grandes (desde 1GB de texto) solicita al sistema operativo gran cantidad de espacio en memoria, lo cual hace finalizar el programa con excepción *MemoryError*. Además, hemos encontrado opciones de código libre más eficientes para dicho cálculo.
- Los pasos para crear un motor es ejecutando un servidor implementado utilizando el microframework Flask, transfiriendo los datos de ítems a él mediante los endpoints /train y /predict al ejecutar la aplicación web. Una vez que finaliza el entrenamiento, se pueden hacer consultas al servidor en ejecución.
- Las recomendaciones calculadas son almacenadas en una base de datos en memoria generada por Redis. Tal cálculo se pierde al cerrar el programa, es decir las recomendaciones no son almacenadas en el disco duro.

Ha sido muy útil para:

- Comprobar inicialmente el funcionamiento de un sistema de recomendación con datos simples.
- Entender la lógica de entrenamiento y cálculo de recomendaciones con anterioridad a consultas.
- Considerar deployments a Heroku.

- **Text recommendation system developer built in Python and Dash by Plotly (tRECS) [12]** Este proyecto consiste de un constructor de sistemas de recomendaciones, donde se guía al usuario a través de la limpieza de sus datos, construcción de modelos y en última instancia, se crea un sistema de recomendación accesible desde la interfaz. El usuario también puede visualizar

algunos modelos y otras características de sus datos. La carga de datos se realiza a través de un archivo .csv ubicado en el dispositivo. Luego de seleccionar las columnas de "etiquetas" y "datos" en la interfaz, pertenecientes al archivo .csv, el mismo se carga internamente como un *DataFrame* Pandas. Luego se guía al usuario en el siguiente paso a cómo desea preprocesar su texto con algunos filtros: minúsculas, eliminar HTML, eliminar palabras que sólo ocurren en un documento, eliminar números, entre otros. También opciones de *Stem* y *Lemmatize*. Se ofrece la construcción de distintos modelos a partir de estos datos: Tf-Idf, LSA (Latent Semantic Analysis), LDA (Latent Dirichlet Allocation) y SpaCy. El servidor y la interfaz están implementados usando *Dash by Plotly*. De éste proyecto ha sido útil conocer el microframework Dash, el cual permite crear interfaces puramente en Python, sin necesidad de implementar usando herramientas de front-end (Html, CSS, JavaScript, templates, etc.). Ha sido limitante para adaptarlo a este trabajo, pues es deseable que la solución que buscamos perdure en el tiempo, además del uso de herramientas de almacenamiento de datos más estructuradas que csv y necesidad de eficiencia con el uso de memoria y mejores tiempos de entrenamiento para volúmenes de datos superiores. Además sería necesario adaptar el preproceso de texto y posterior adaptación de reglas, ambos propiamente legales.

- **word2vec-recommender [13]** Consiste en un toolkit python presentado en la Pycon India 2016, a través del cual se puede construir un motor de recomendación con el historial de navegación y las reseñas generadas por usuarios utilizando la técnica de Word2vec (tecnologías: Google Word2Vec, Gensim, Numpy, Flask, Redis, entre otras). Permite recomendar artículos en la plataforma Amazon, descargando las reseñas de productos escritas por usuarios. Utilizamos este sistema como referencia para comprender la implementación del *streaming* necesario para entrenar un modelo en la suite de Gensim [14]. word2vec es amplia-

mente usado en la actualidad para producir *word embeddings*.

No se ha reutilizado su código, puesto que su implementación es muy adaptada al tipo de datos que usa (revisiones de artículos Amazon en cierto formato de texto), lo cual hace más costoso su adaptación que una nueva implementación, teniendo en cuenta también su modularización y la diferencia de requisitos a cubrir en nuestro proyecto.

- Se han clonado e instalado repositorios de otros recomendadores:
  - Content Based Text Recommendation (cbtr) [15]: artículos de wikipedia.
  - Gutenberg [16]: recomendador por contenido que utiliza la base de datos completa de libros del Proyecto Gutenberg.
  - Tweet recommender system [17]: recomendación de Hashtags a partir del texto.

2.4. Sobre el corpus

Por ***corpus*** entendemos al conjunto de datos existente que será el *input* del sistema. El corpus utilizado en este proyecto consta de un directorio con 121136 archivos en formato .xml. Cada archivo contiene información acerca de exclusivamente una norma. La ubicación de este directorio está prefijado en el archivo de configuración del sistema.

2.4.1. Fuente

El corpus inicialmente se obtuvo realizando *Web Scraping* utilizando la herramienta *Scrapy* en Python para tal fin. Los scripts han sido desarrollados por *Cristian Cardellino* [18] y posteriormente adaptados al proyecto. En esencia esta técnica consiste en un programa que envía solicitudes http usando ciertos criterios, recibiendo la respuesta http y extrayendo los datos recibidos. En nuestro caso, estos datos consistirán del *id* de la norma y su texto, entre otros metadatos. De esta forma, se han capturado todas las normas accesibles del buscador de la página `www.infoleg.gob.ar/`.

2.4.2. Estructura

La estructura fundamental xml de cada norma está representada en el siguiente diagrama.

Entre otros atributos encontrados en el xml:

- El atributo *id* de la etiqueta <law> es el id de la norma, análoga al id de la misma en la plataforma Infoleg.

- *law_id* de la etiqueta <link> es el id de la norma referida en el texto (no todas las normas referidas en el texto están incluídas).

- *begin* y *end* son enteros que representan posiciones en el texto.

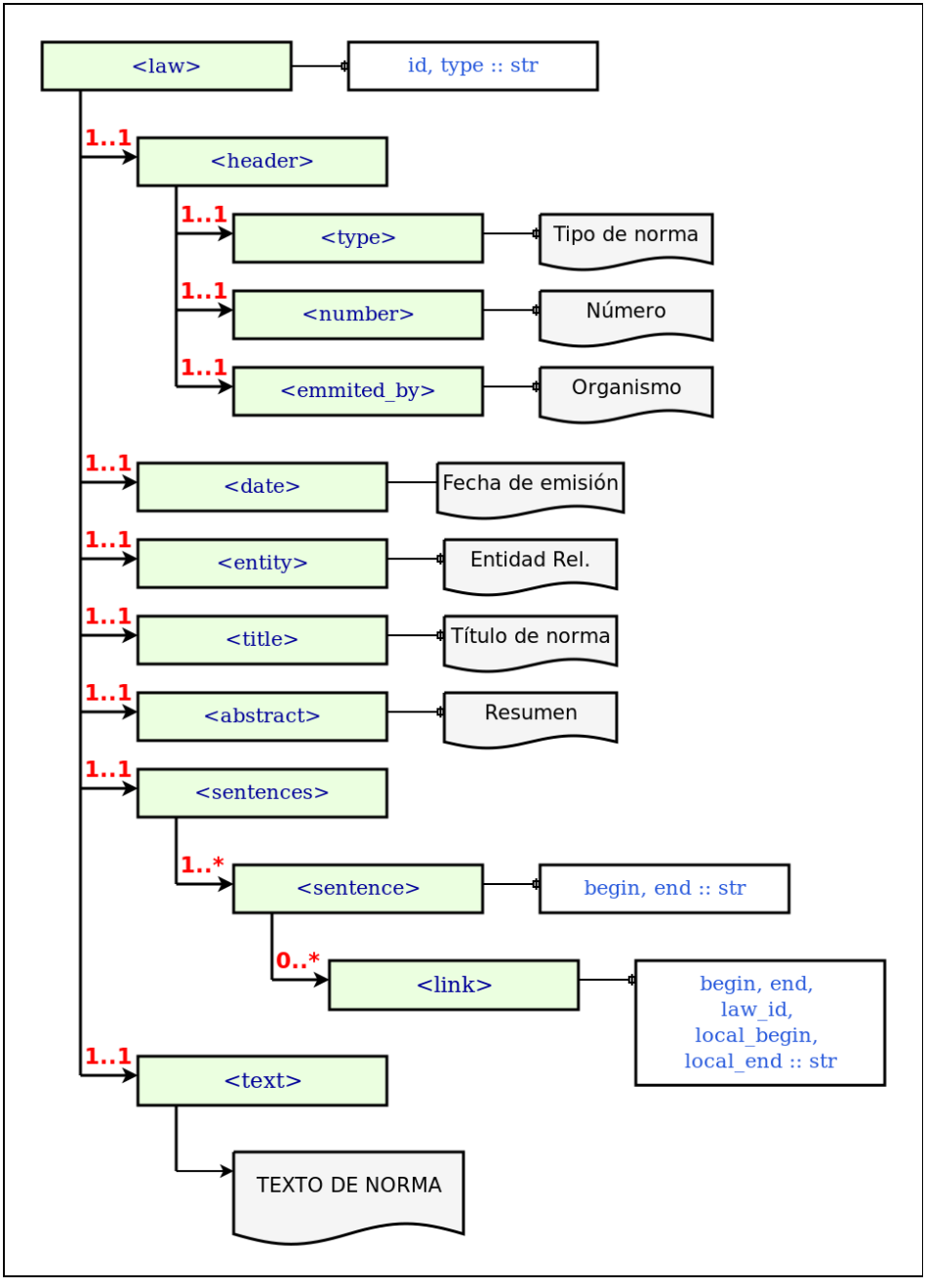


Figura 2.5: Diagrama de estructura de una norma en formato .xml

2.4.3. Estadísticas

Presentamos algunas estadísticas relevantes del corpus, que darán noción sobre el tipo de texto tratado, con el fin de dar a conocer sus principales características y así entender cómo afectará a la construcción del motor y consideraciones a tener en cuenta en la etapa de preproceso, aprendizaje del modelo, manipulación, etc.

Figura 2.6: Nube de palabras representando la frecuencia de palabras en el corpus

1. Cantidad de normas: **123.624** (actualizable)
2. Cantidad de palabras luego de etapa de preproceso (sin *stopwords* y otros caracteres especiales ignorados): **94.501.630**
3. Peso total del corpus: 1.528.020 KB $\approx$ **1,5 GB**
4. Tokens por norma promedio: **764,4**

5. Normas por tipo:

Tipo de Norma	Frecuencia	% del corpus
Resolución	67909	54.93
Decreto	18761	15.18
Disposición	13147	10.63
Comunicación	10062	8.14
Ley	7297	5.90
Decisión Administrativa	3687	2.98
Decreto/Ley	584	0.47
Nota Externa	582	0.47
Decisión	473	0.38
Directiva	308	0.25
Instrucción	260	0.21
Acordada	209	0.17
Acta	148	0.12
Circular	146	0.12
Convenio	14	0.01
Laudo	13	0.01
Recomendación	8	0.01
Acuerdo	7	0.01
Nota	5	0.00
Providencia	2	0.00
Protocolo	1	0.00
Interpretación	1	0.00

6. Normas más referenciadas:

Tipo y Número	Cantidad
Resolución 1310/2012	45
Resolución 1200/2012	44
Decreto 27/2018	37
Resolución 1268/2017	35
Ley 27431	29

Capítulo 3

Arquitectura del Sistema

3.1. Introducción

3.1.1. Primeras aproximaciones

Abordamos las primeras aproximaciones hacia el diseño y arquitectura del sistema de recomendación de textos legales. Más adelante daremos detalles sobre su interfaz e interacción con el usuario.

Como hemos descrito anteriormente, dependiendo del problema, diferirá la elección de qué tipo de sistema de recomendación (colaborativo, por contenido) es utilizado para dar las recomendaciones. Su arquitectura variará por ende, en base al mismo y los datos del dominio.

En nuestro caso, el tipo del sistema de recomendación será de *content-based filtering* y el tipo de los datos será texto. No abarcaremos el caso del uso de datos de usuarios registrados en la plataforma (si los hubiere), como por ejemplo sus intereses sobre ciertas normas, historiales de búsqueda, tiempos de lectura, etc. propios de sistemas híbridos o de *collaborative filtering*, aunque puede ser implementado a futuro. Generalmente en los últimos, los tipos de datos de sus dominios son más complejos de analizar por su contenido en sí (e.g. películas, música, obras de arte, etc.) lo cual conlleva al uso de meta-datos y

datos de sus usuarios.

Por lo tanto, las recomendaciones del sistema serán obtenidas exclusivamente por la extracción y procesado del contenido de cada norma y su posterior análisis con otras normas definido por el modelo elegido y el algoritmo de vectorización.

Esta elección se debe a que, en principio, la consulta de normas se da en mayor medida por usuarios no registrados, muchos de los cuales realizan consultas rápidas sobre la plataforma. Actualmente la plataforma InfoLEG [1] no posee públicamente la función de registro de usuarios.

La arquitectura básica puede ser representada por el diagrama 3.1 de la vista de componentes y conectores del sistema, la cual lo ve como unidades en ejecución.

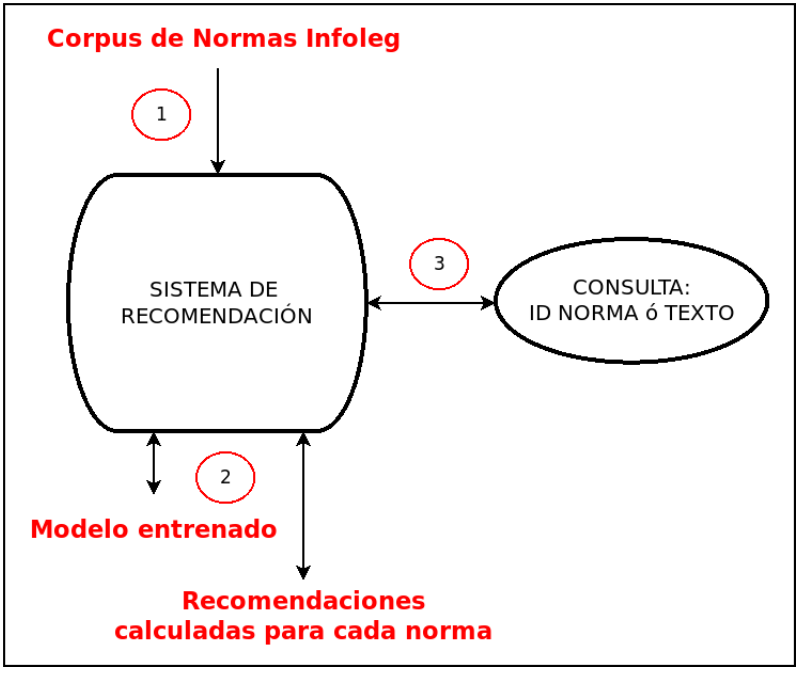


Figura 3.1: Arquitectura del sistema simplificada.

3.1.2. Tomando decisiones de diseño

Detallando más el diseño, y a partir de las investigaciones realizadas previamente, sabemos que para dar recomendaciones, antes es necesario poseer un motor de recomendación que tenga "conocimiento"de la base de datos de ítems a recomendar: obtener recomendaciones de normas a partir de la misma o un texto desconocido, respecto a cientos de miles de documentos y haciendo el cálculo en tiempo real, podría llevar demasiado tiempo y tampoco sería una buena decisión de arquitectura, pues el tiempo de respuesta limitaría el tiempo de análisis de la base de datos y por lo tanto su precisión.

El conjunto de documentos que se pueden consultar en el caso de solicitar id's de normas es bastante estático (sólo se modifica a la hora de que aparezcan nuevas leyes en la base de datos). Por lo tanto, el entrenamiento del modelo debe realizarse con anterioridad a la recepción de consultas del usuario.

Por consiguiente, se construye una interfaz web para la interacción con el sistema y realizar consultas de recomendaciones. Desde la misma se pueden consultar normas por su **id** en la base de datos o bien por su **tipo** (ley, decreto, etc.) junto a su **número**, como también un fragmento de **texto libre**. Dicha interfaz, podría omitirse en caso de que el *back-end* del sistema sea utilizado por otra plataforma. A partir de las investigaciones realizadas, conocimiento del autor, e implementaciones en librerías de los modelos y herramientas fundamentales usadas en el proyecto, se implementa en Python 2.7.

Entrenamiento del modelo

El entrenamiento del modelo y el cálculo de semejanzas se podría efectuar en el momento que el administrador ejecute el servidor. En este caso, ambos podrían ser obtenidos a partir del entrenamiento en tiempo de ejecución y ser almacenados en memoria RAM, desechando estos datos al apagar el servidor. Como hemos visto, en desarrollos relacionados en repositorios de código libre, una forma de hacer esto

es utilizando *Redis* [19] como base de datos en memoria.

En nuestro caso, optamos por realizar el entrenamiento y calcular semejanzas offline, guardando en una base de datos de recomendaciones los resultados. Analizaremos el rendimiento de diferentes motores de bases de datos, como *SQLite3* y *MongoDB*. Por otro lado, el motor entrenado (objeto Python) se guardará usando la librería *Pickle* para una posterior carga, junto a la base de datos, al ejecutar el servidor. El objeto python del modelo se almacena para las consultas en forma de fragmentos de texto, pues es necesario poseer el vectorizador para inferir un vector en el modelo a partir de ese texto y obtener recomendaciones.

3.2. Etapas

Ya analizadas las posibles decisiones de diseño y con intención de clarificar los pasos ejecutados en el proyecto y guiar al lector durante las distintas secciones, presentamos el flujo de datos dentro del sistema que puede ser resumido en tres grandes etapas. Las mismas representan la arquitectura final del proyecto.

A pesar de que no necesariamente cada etapa debe finalizar por completo en tiempo de ejecución para que comience la siguiente, las ordenamos por dependencia de datos:

A - Creación de Corpus

- **Scraping** Se realiza el *scraping* (extracción de información) de normas del sitio `www.infoleg.com`. Se almacenan los archivos HTML descargados de los encabezados y texto completo para cada norma.
- **Preproceso de datos** Se crea/actualiza el corpus, procesando los archivos HTML descargados con otro módulo python ubicando los datos de interés en un archivo XML para cada norma (se crea el directorio para el corpus).

B - Entrenamiento del modelo

- **Preproceso de texto** Una vez obtenido el corpus, se comienza el entrenamiento de los motores de recomendación. Durante la etapa se procede a cargar las normas desde el directorio de normas establecido en el archivo de configuración. Se realizan las etapas intermedias que implica pre-procesar texto: *tokenizar* el texto de cada norma (segmentación de la cadena de texto inicial en sub-cadenas o palabras llamadas *tokens*), aplicar filtros a los tokens obtenidos (minúsculas, caracteres especiales, entre otros), y aplicar reglas manuales intrínsecamente relacionadas al tipo legal del texto. Una vez pre-procesada una norma, se

guarda su resultado (lista de tokens) en una entrada de la base de datos del corpus, junto a sus metadatos (fecha, título, entre otros). Guardar el pre-proceso de cada norma optimiza tiempos de entrenamiento de nuevos modelos.

- **Vectorización** El objetivo de esta etapa es la transformación de las normas como listas de cadenas inicialmente, a vectores donde cada elemento es un número real. Dependiendo del modelo/algoritmo (Tf-Idf, Doc2Vec) que se utilice para vectorizar, variará el tamaño y contenido de los vectores.
- **Cálculo de semejanzas** Para cada norma (vector) se calcularán los vectores más semejantes (pre-determinadamente se hallan los primeros 20). Independientemente del modelo, este cálculo es análogo a encontrar los 20 vectores ("vecinos") más cercanos en el espacio vectorial generado. Ambos modelos vistos lo realizan internamente mediante el cálculo de similitudes del coseno.
- **Almacenamiento** En el momento de haber calculado las recomendaciones para una norma, las mismas serán almacenadas en la base de datos de recomendaciones. Los datos serán el modelo seleccionado, el id de la norma objetivo y el tipo de norma recomendada (si se solicitó recomendaciones "normales" el campo será "generic", si se solicitó recomendaciones separadas por tipos, este campo será el tipo de la norma recomendada). Junto a estos campos, estarán el id y probabilidad de cada recomendación.

  Finalmente, el modelo entrenado se almacenará en el directorio definido en formato *pickle*, para consultas sobre fragmentos de texto al sistema.

  El diagrama 3.2 muestra el diagrama de flujo de datos caracterizado por esta etapa.

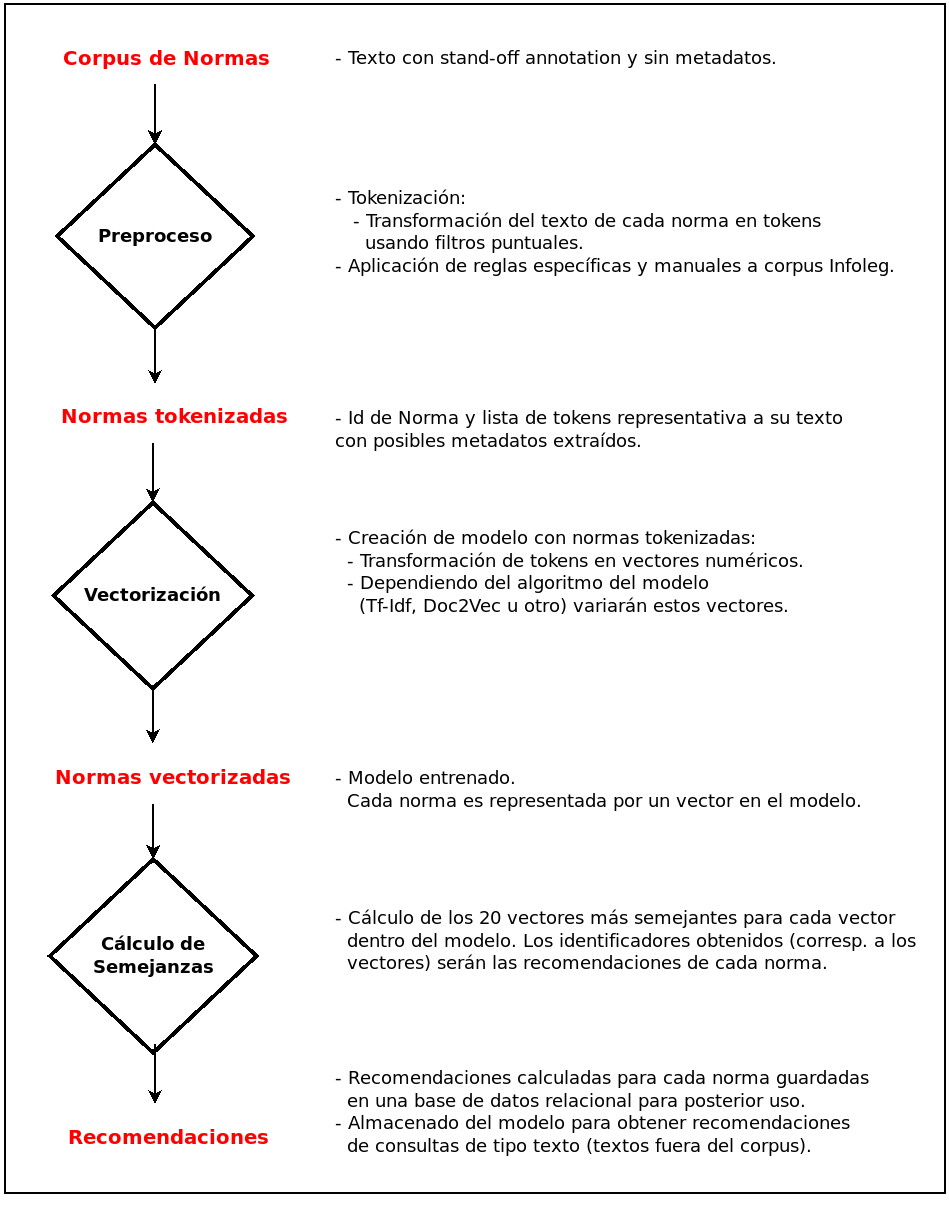


Figura 3.2: Diagrama de Flujo de Datos en el entrenamiento (**B**).

C - Servidor Web Posteriormente al entrenado y almacenado de los modelos, el servidor puede ser puesto en ejecución, el cual cargará la base de datos y los modelos guardados. Responderá peticiones de clientes por recomendaciones, mediante la interacción/consulta a través de la interfaz. Las recomendaciones obtenidas se mostrarán en la interfaz.

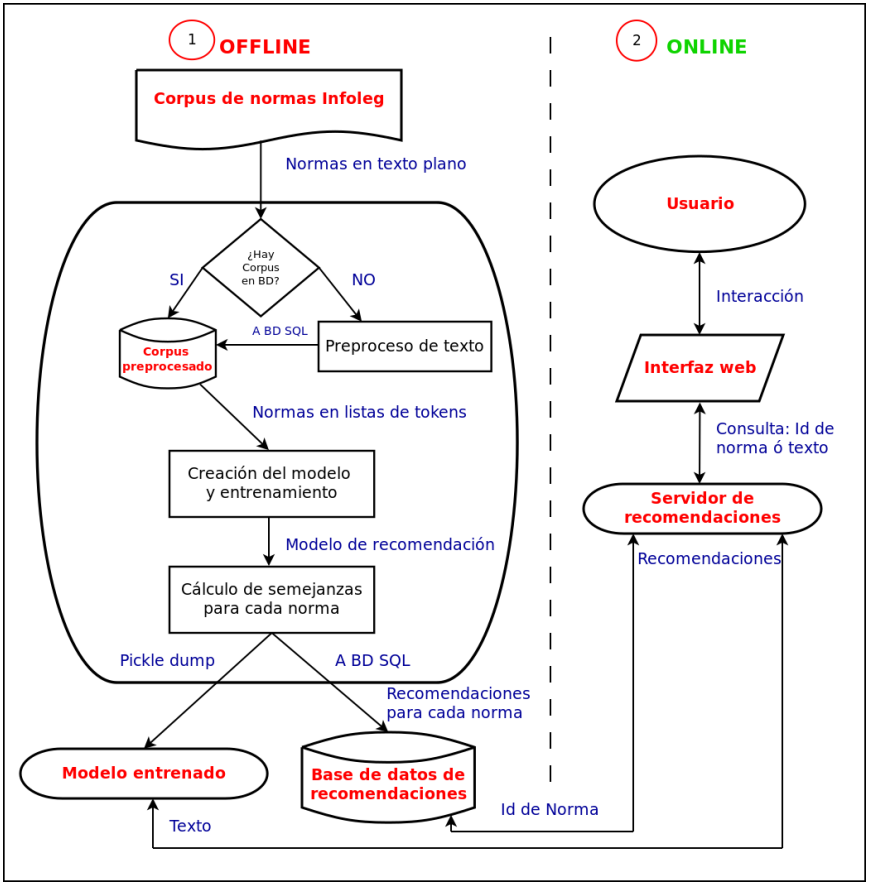


Figura 3.3: Arquitectura del sistema (componentes y conectores).

Capítulo 4

Implementación

En este capítulo describimos los detalles de la implementación del sistema.

4.1. Herramientas Utilizadas

Se han utilizado diversas librerías Python para llevar a cabo el proyecto, las cuales listamos a continuación. También herramientas de código libre, entre otras mencionadas anteriormente.

4.1.1. Vectorización de texto y cálculo de semejanzas

Scikit-Learn

De esta librería se ha utilizado la clase `TfidfVectorizer` [20]. Convierte una colección de documentos en una matriz de *features* TF-IDF, usada como el vectorizador de normas para los motores basados en TF-IDF del sistema.

Gensim

`Gensim` es una plataforma de código abierto en Python para modelado vectorial de textos y modelado temático. Está diseñada especícamente para manejar grandes colecciones de textos, usando *streaming* de datos y algoritmos incrementales eficientes.

Utilizamos la clase `gensim.models.doc2vec.Doc2Vec` [21] para motores de vectorización del sistema, la cual implementa el algoritmo Doc2Vec antes mencionado.

`sparse_dot_topn`

Para calcular la semejanza entre dos vectores de valores TF-IDF, son utilizadas usualmente las *similitudes del coseno*, la cual puede ser vista como el producto punto normalizado entre vectores.

Para realizar éste cálculo en el modelo construido (y así obtener recomendaciones), inicialmente se empleó la función `cosine_similarity` de la librería `sklearn`, no obteniendo buenos resultados en cuanto a consumo de memoria para grandes cantidades de datos.

Por ello se optó por la librería de código libre `sparse_dot_topn`, presentada en la página [22] y con repositorio github [23].

La herramienta mejora la velocidad para el cálculo de las distancias un 40 % y reduce el consumo de memoria. Está escrita en Cython y se incluye en el repositorio del proyecto.

4.1.2. Interfaz

Dash by Plotly

Dash [24] es un *framework* de Python que permite construir interfaces web interactivas, principalmente para aplicaciones web analíticas. Está construido sobre Plotly.js, React y Flask, y permite agregar fácilmente dropdowns, formularios y gráficos ligados a la aplicación de análisis, utilizando sólo código Python de base.

El framework permitió construir la interfaz del sistema en líneas de código python, incluyendo interactividad y estilos a la aplicación (css, javascript).

4.1.3. Almacenamiento

SQLite3

Para almacenar datos tanto del corpus como de recomendaciones se ha utilizado el motor de base de datos relacional `SQLite3`. La herramienta permitió crear bases de datos, tablas e insertar valores y realizar consultas de forma rápida desde python con sentencias SQL.

pickle

Para almacenar los motores entrenados se utilizó `pickle` [25], librería que permite guardar en disco objetos python. Fue necesario hacerlo, para conservarlos hasta el momento de ejecutar el servidor (posterior al entrenamiento) y poseer el vectorizador para las consultas de tipo texto.

4.1.4. Preproceso de texto

re

Librería para utilizar expresiones regulares en python. Se usó en el preproceso, sobre el texto de las normas en la definición de filtros particulares.

nltk

Se utilizaron las *stopwords* del castellano de NLTK, para excluir estas palabras de cada norma.

4.1.5. Multithreading

joblib, threading

Luego de finalizar la versión secuencial del proyecto, se añadió concurrencia con el fin de mejorar tiempos de cómputo. Se identificaron cálculos independientes entre sí en etapas intermedias del entrenamiento, asignando a cada tarea un proceso y un núcleo de CPU. Se usó `joblib` [26] principalmente para utilizar concurrencia en la construcción de motores que distinguen por tipo de norma (cada hilo un tipo).

4.1.6. Gráficas

wordcloud, matplotlib

Han servido para realizar nubes de palabras, gráficos y visualizar estadísticas durante el proyecto.

4.1.7. Interactividad intérprete Python

ipython, jupyter

Se han utilizado siempre durante la implementación del proyecto, ya sea por aprendizaje, o para realizar pruebas y comprobar funcionamiento, de las herramientas utilizadas y de etapas intermedias en el código fuente.

4.1.8. Otras

- `docopt`: Para establecer la interfaz de scripts python en línea de comandos bash.

- `lxml`: Para cargar, guardar archivos `xml` y extraer sus datos.

- `collections` para diccionarios y contadores, `time` para control de tiempo, `os` para manejo de directorios, `csv` para almacenado de datos en formato .csv.

4.2. Hardware

Este proyecto se ha implementado sobre el hardware que provee el Centro de Computación de Alto Desempeño (CCAD) de la Universidad Nacional de Córdoba, tanto para pruebas como para despliegue.

JupiterAce

La computadora que ha sido dada como recurso primario para realizar el proyecto. Entre otras especificaciones, posee un procesador Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz (12 núcleos) y 132GB de memoria RAM. Accesible vía `ssh`. Fue necesaria pues se requirió gran cantidad de memoria RAM y capacidad de cómputo para finalizar el entrenamiento de los modelos en tiempos adecuados, los cuales no serían posibles en una computadora personal. Sin esta computadora, el proyecto no hubiera sido posible de realizar.

Nabucodonosor

Ha sido utilizada junto a JupiterAce para similares tareas, por un período menor de tiempo.

4.3. Módulos del sistema

En la presente sección se detallan decisiones tomadas en la implementación y funcionalidades de sub-módulos, presentes en el código fuente del proyecto, almacenado en el repositorio público Bitbucket `bitbucket.org/acapello/thesis` [27]. Organizaremos la sección de acuerdo a las etapas caracterizadas en la arquitectura del proyecto.

En la figura 4.1 se muestra la dependencia de los módulos python del sistema. Se han quitado algunos módulos adicionales ajenos a la tarea principal.

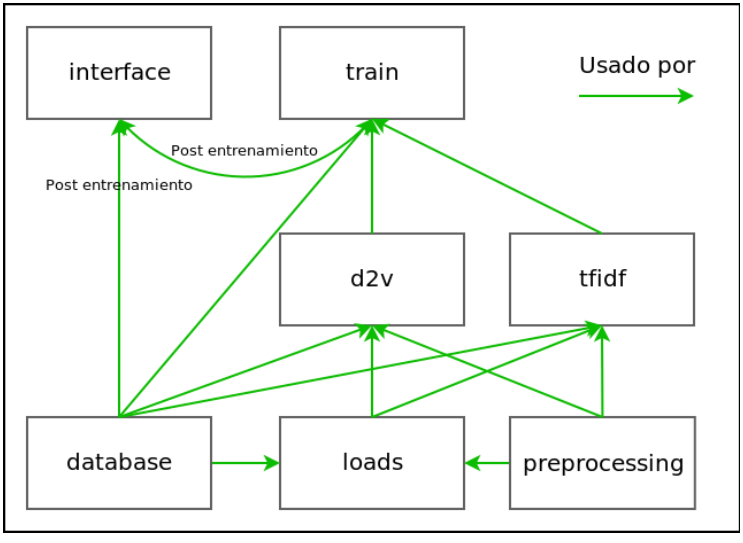


Figura 4.1: Diagrama de módulos del sistema simplificado.

- `database`: funciones conexión y creación de bases de datos y consultas usadas para el corpus y las recomendaciones. `d2v` y `tfidf` lo utiliza para insertar, `loads` para cargar desde una base de datos normas, y `train` para crear una base de datos.
- `loads`: funciones de carga de normas desde el directorio, definición de iteradores sobre el corpus en directorio y en la BD.

- `preprocessing`: preproceso de texto (tokenize). Usado por `loads` al cargar normas, y por `d2v, tfidf` al preprocesar fragmentos de texto nuevo.
- `d2v`: motores basados en Doc2Vec (simple y por tipos de norma).
- `tfidf`: motores basados en Tf-Idf (simple y por tipos de norma).
- `train`: script para el entrenamiento de cada motor
- `interface`: interfaz web del sistema, utiliza los motores entrenados (consultas tipo texto) y la base de datos de recomendaciones.

4.3.1. Scripts

Programas python ejecutables por línea de comandos. Debajo del directorio `project/recommenders/scripts/`.

Entrenamiento

Script `train.py` el cual es utilizado para entrenar los motores de recomendación. Uso en consola de comandos:

```
train.py [-m <model>] [-o <file>]
train.py -h | --help
m = [d2v | tfidf | d2v-bt | tfidf-bt]
```

Estadísticas

Para obtener estadísticas sobre el corpus se utilizó el script `stats.py`. Entre otras, obtiene palabras más mencionadas, tipos de normas y su frecuencia y porcentaje sobre el corpus, normas más referenciadas, volcando los datos a archivos `.csv`. Se crea un gráfico de nube de palabras (wordcloud) con las palabras tokenizadas del corpus.

Evaluación

Script `eval.py` para que el sistema de `n` recomendaciones para una norma con id `k` usando el modelo de tipo `m` a partir de la base de datos de recomendaciones ya construida.

4.4. Creación de Corpus: scraping y preproceso

Para obtener el corpus, nos valimos de la herramienta *Scrapy*, la cual permitió *scrapear* o realizar *crawling* de la plataforma Infoleg.

En particular, el script descarga todas las normas HTML desde el link[1] donde N varía de 0 hasta el último id de norma publicada a la fecha, el cual hoy día ronda los 320000.

Tambíen se descargan los HTML's de los links "texto completo" o bien "texto actualizado" (en caso de existir).

Luego de ésta etapa, se realiza un preproceso de los HTML descargados a para obtener normas en formato XML donde los datos extraídos se almacenan ordenadamente en etiquetas. De cada uno de éstos HTML se extraen metadatos, como tipo de ley y número, título, entre otros y el texto completo de la norma.

El script principal desde el cual se partió y realizó scraping fue provisto por el Dr. Cristian Cardellino [18], FaMAF UNC.

[1] `servicios.infoleg.gob.ar/infolegInternet/verNorma.do?id=N`

4.5. Entrenamiento

En la presente sección detallamos implementación de los pasos mencionados en la etapa de entrenamiento de los motores del sistema.

4.5.1. Preproceso de texto

Para entrenar un modelo, se utiliza la lista de *tokens* (sub-cadenas de texto) de cada norma del corpus. Dichos tokens representan el contenido de una norma, y son obtenidos a partir de la fragmentación y uso de filtros aplicados al texto plano de cada norma. Para realizar la tokenización (y preproceso) en el proyecto se utiliza la función `tokenize` de `preprocess.py`. Input: Texto. Output: lista de tokens. Etapas:

1. pre-procesado del texto plano de la norma (input texto, output lista de tokens) con `preprocess_string` de `gensim.parsing.preprocessing` usando como filtros al texto:
   - Quitar tags de la forma <w*> (aparición de posibles tags)
   - Quitar espacios sobrantes y \n
2. Luego, a cada token:
   a) convertirlo a minúsculas
   b) ignorarlo si está dentro del conjunto de palabras específicas a ignorar (nº por ejemplo)
   c) ignorarlo si es un *stopword* del castellano (artículos, pronombres, preposiciones, etc.). (`nltk.corpus.stopwords`)
   d) quitar caracteres no deseados como comillas ", apóstrofe ', o el símbolo º, usando expresiones regulares (`re`)
   e) tratar selectivamente el caso que el token contenga alguna sub-palabra que contenga los caracteres en el conjunto . - \/ y el mismo esté rodeado por números

- Si sucede, entonces consideramos que se trata del número de una ley, decreto, fecha, etc. (e.g. 12.548 o 12/07/17 o 124-57) y por lo tanto conservamos el token con este formato
- si no sucede, fragmentamos el token en los puntos que contenga caracteres del conjunto ( ) : , ; e.g. palabras del tipo canada. o juarez, o ciudad,en. Si el fragmento es no vacío, lo conservamos (será un nuevo token)

De esta forma se obtiene la lista de tokens final para cada norma.

4.5.2. Bases de Datos

Con el fin de almacenar recomendaciones, se crea una base de datos al inicio de la etapa de entrenamiento, con las siguientes tablas que serán llenadas en tiempo de ejecución.

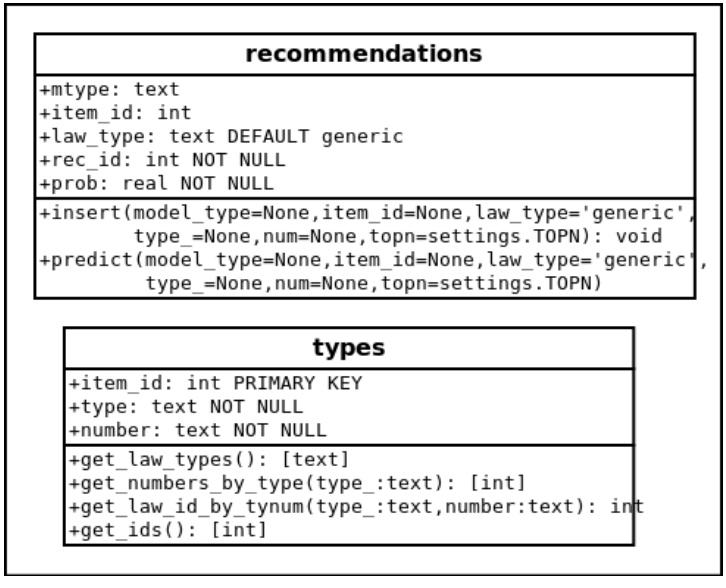


Figura 4.2: Base de datos de recomendaciones (tablas y funciones).

- mtype: Es el tipo de modelo, con el cual se entrenó la recomendación, puede tomar los valores [d2v | tfidf | d2v-bt | tfidf-bt]
- item_id: el id numérico de la norma en `infoleg.gob.ar`
- law_type: Tipo de la norma para la cual es la recomendación. Si es "Ley"por ejemplo, la recomendación es para el conjunto de normas de tipo Ley, si es "generic", la recomendación es para todo el conjunto de normas. Toma valores: [generic | Ley | (Decreto, Decreto/Ley) | Resolución | Others]
- rec_id: Id numérica de la norma recomendada.
- prob: probabilidad de recomendación (semejanza, entre 0 y 1).
- `insert` y `predict` son funciones para insertar y dar valores en la base de datos, respectivamente.

Como adición a estas tablas, se crea una para el corpus en tiempo de entrenamiento, para optimizar futuros aprendizajes de modelos. Se almacenan los tokens de cada norma de la etapa de preproceso. El motor de base de datos permitirá realizar consultas personalizadas con facilidad.

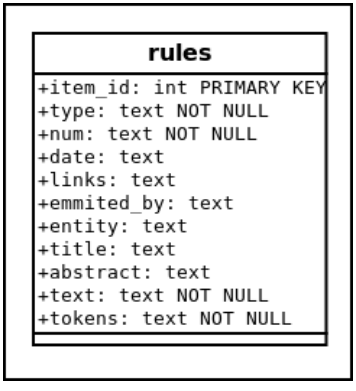


Figura 4.3: Tabla en la base de datos del corpus.

En un inicio, para las tablas anteriormente descritas, se implementó el almacenado (y cargado) de los datos tanto de leyes como de los resultados (recomendaciones) en una base de datos SQL mediante *SQLite3*.

Se analizaron cuidadosamente los resultados, y los tiempos de consulta en tiempo real, a través de la interfaz, principalmente.

Se optó por explorar otro motor de base de datos, usada para documentos, NoSQL, *MongoDB*. Se implementó el almacenado y cargado usándolo, en esencia usando como atributos las mismas columnas de las tablas descritas anteriormente.

MongoDB es un motor que permite realizar consultas de la misma forma que se realizan en una base de datos relacional, pero básicamente almacenando/entregando los datos en formato JSON. Es más fácil de implementar y usado en el ámbito, los datos quedan almacenados en disco, y permite concurrencia (múltiples hilos accediendo y escribiendo en la base de datos) lo cual en SQLite3 no lo permite por defecto.

Luego de varias pruebas y consultas, encontramos que el rendimiento del módulo SQL es ligeramente superior en tiempo real, a pesar que el tiempo en entrenamiento sea un tanto mayor por no permitir fácilmente concurrencia. Se tiene en cuenta que los resultados son de carácter relacional, y su implementación es ampliamente optimizable en cuanto a tiempos de consulta, a posteriori . Lo hemos elegido para la implementación final del sistema.

4.5.3. Modelos de vectorización

Para realizar la vectorización de los *tokens* de cada norma, se han implementado en el sistema los dos motores antes mencionados. Ambos se han sometido a experimentos para analizar su desempeño y calidad de recomendaciones.

Recomendaciones generales/simples

- **Doc2Vec**

  Implementación por Gensim del método Doc2Vec [21].

  La gran capacidad de esta librería permitió hacer realidad la tarea núcleo del proyecto, que vincula los cálculos matemáticos internos para obtener el orden de las recomendaciones.

  Se han usado los siguientes parámetros en el entrenamiento del modelo:

  - **vector_size**=300. Tamaño del vector.
  - **dm**=0. Algoritmo de vectorización. Se usa PV-DBOW.
  - **alpha**=0.01. Tasa de aprendizaje inicial.
  - **min_alpha**=0.0001. La tasa de aprendizaje desciende linealmente a éste valor a medida que el entrenamiento progresa.
  - **window**=10. Tamaño de la ventana.
  - **min_count**=1. Las palabras con frecuencia en el corpus menor a este valor son ignoradas en entrenamiento.
  - **workers**=16. Cantidad de sub-procesos.
  - **epochs**=50. Cantidad de iteraciones en entrenamiento del modelo.
  - **dbow_words**=0. valor 1 para construír vectores de palabras, 0 caso contrario.

Internamente, las consultas textuales se obtienen mediante inferencia a un vector en el modelo partiendo de los tokens del texto (infer_vector() en gensim).

Las similitudes en Doc2Vec se hallan con la función most_similar() que internamente computa el cálculo de la similitud del coseno.

Se usaron valores por defecto de la librería para el resto de los parámetros del modelo que no aparecen en la lista.

- **Tf-Idf**

  Motor usando el vectorizador TF-IDF de Scikit-Learn [20]. Hemos usado los parámetros:

  - **analyzer**='word'. Los *features* están compuestos de palabras.
  - **ngram_range**=(1, 3). Establece el largo de n-gramas (1, 2 y 3) a tener en cuenta durante el entrenamiento. Es decir, en las columnas de la matriz también tendremos bigramas y trigramas.
  - **min_df**=0. También llamado valor *cut-off*, para ignorar los términos que tienen una frecuencia de documento estrictamente inferior a éste umbral.

Recomendaciones entregadas por tipos de norma

En este caso, para lograr la recomendación diferenciando/distinguiendo distintos tipos de norma, se programan dos nuevos motores en cada caso, principalmente porque buscar entre las primeras 100 o 500 recomendaciones en un motor simple, no es garantía de encontrar recomendaciones para cierto tipo (puede que sean todas Leyes y Decretos, por ejemplo).

Actualmente, el sistema está configurado para entregar diferenciadamente recomendaciones de: **Leyes**, **Decretos** (que incluye Decretos

y Decretos/Ley, por su relevancia jurídica), **Resoluciones**, y **otras normas** (entre ellas Disposiciones, Decisiones, Actas, etc.)

- **Doc2Vec** El motor realiza lo siguiente:

  1. Crea varios hilos, uno por cada tipo de norma (concurrencia)
  2. Cada hilo crea un modelo Doc2Vec con las normas de ese tipo disponibles en la base de datos.
  3. Finalizados, se construyen las recomendaciones, iterando sobre el corpus, donde se infiere un vector a partir de los tokens de cada norma, para cada modelo Doc2Vec.

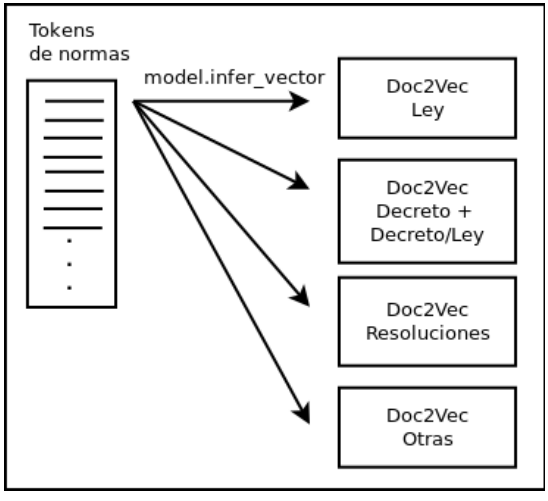


Figura 4.4: Obtención de recomendaciones por tipo con Doc2Vec.

- **Tf-Idf**

  - Se crea un motor Tf-Idf general (de todas las normas)
  - Se crea un hilo para cada tipo de norma, cada uno entrena un modelo Tf-Idf con el vocabulario del vectorizador general, y las normas de ése tipo

- Cada hilo calcula las similitudes del coseno usando la matriz general y la matriz de su tipo obtenida, almacenando las recomendaciones en la base de datos

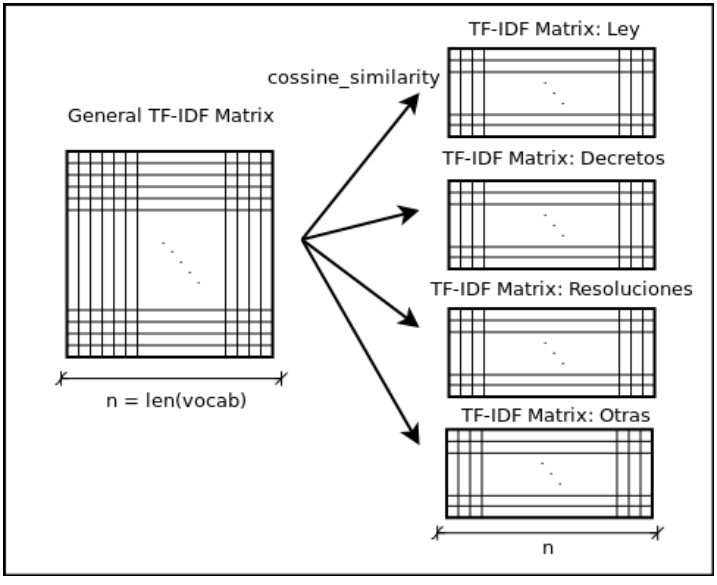


Figura 4.5: Obtención de recomendaciones por tipo con TF-IDF.

4.6. Servidor Web

A la hora de diseñar una interfaz web para el sistema junto a un servidor en Python se consideraron varias opciones, entre los frameworks web existentes para dicho lenguaje.

La elección depende de la medida y las características del proyecto. Entre ellas podemos encontrar necesidad de personalización de la interfaz, estilos específicos, estructuras de urls complejas, entre otras.

Se ha considerado el framework *Django*, y los microframeworks *Flask*, *Bottle*, *Dash*, entre otros.

Se ha elegido Dash puesto que su solución abarca las necesidades y se adapta a características buscadas, como interacción y estilos, todo esto en un mismo lenguaje (Python) sin extender demasiado la solución.

La interfaz principal del sistema consta de botones para elegir el modelo de vectorización (posteriormente eliminado al elegir uno definitivo), consultar por id de norma, tipo y número de norma (ambas referidas a la plataforma infoleg) o entrada de datos en forma de texto, y dar recomendaciones generales o diferenciadas por tipo de norma. Entre otras ventanas podemos encontrar la vista de texto completo de una norma.

Al ejecutar el proyecto, se puede apreciar interactivamente la solución construida. A continuación, añadimos imágenes de muestra de la interfaz del sistema. Se omiten algunos detalles.

Figura 4.6: Portada.

Figura 4.7: Búsqueda con tipo y número de norma.

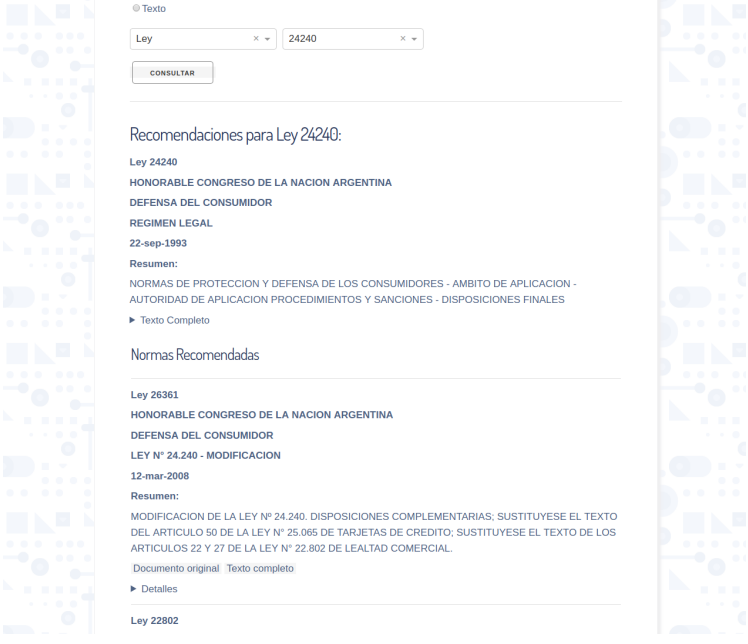


Figura 4.8: Entrega de recomendaciones: simple.

Figura 4.9: Entrega de recomendaciones: por tipo de norma

Figura 4.10: Consulta por recomendaciones con un fragmento de texto.

Figura 4.11: Consulta por recomendaciones con un fragmento de texto.

Capítulo 5

Experimentos

En este capítulo comparamos la calidad de los resultados que ofrece cada uno de los métodos implementados en el recomendador de textos: tf-idf y doc2vec. Se realizó una evaluación indicativa, con algunos casos testigo.

5.1. Método de evaluación

Se compararon diferentes modos de funcionamiento del sistema. En el Cuadro 5.1 observamos los resultados obtenidos usando textos completos de normas de InfoLeg para encontrar normas relacionadas, en el Cuadro 5.2 mostramos los resultados cuando se usan textos libres para encontrar normas relacionadas. Primero, con respecto al cálculo de semejanzas entre documentos, se compararon las recomendaciones basadas en tf-idf y las basadas en doc2vec. También se compara el rendimiento en la recomendación de normas en general (*general*) y distinguiendo por tipos (*por tipo*). Los tipos de documento que se distinguen son: Ley, Decreto (que incluye Decretos y Decretos/Ley), Resolución y Otros.

Para evaluar el rendimiento del recomendador se seleccionaron 5 documentos y 5 fragmentos de texto libre, de diferentes largos. En el

Apéndice A se muestran ejemplos usados para evaluación. Para cada uno de estos 10 ejemplos se obtuvieron recomendaciones usando tf-idf y doc2vec, generales y clasificadas por tipos de norma, en ambos casos.

Cada uno de estas recomendaciones fue puntuada por un evaluador humano, usando la planilla que se muestra en la Figura 5.1. Se evaluó los modelos respecto a adecuación semántica real entre los textos a partir de su lectura, como también con noción legal en la evaluación. Los evaluadores asignaron puntuación "bueno", "malo" o "medio" a cada una de las 3 primeras recomendaciones para cada norma/texto, de la misma forma para cada categoría en los resultados distinguidos por tipo. Se incorpora como bueno 100 %, medio 50 % y malo 0 % al sub-total. Cada puntaje de cada celda para un evaluador de los cuadros que se muestran son obtenidas por el promedio de estos puntajes.

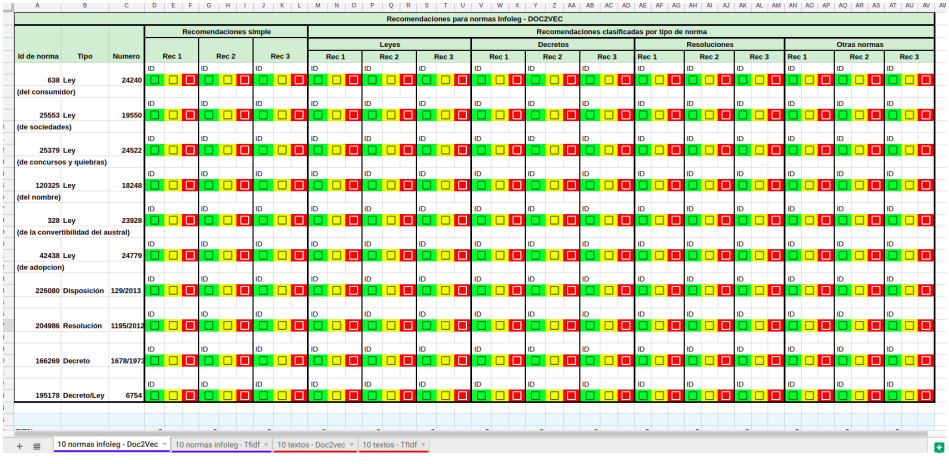


Figura 5.1: Planilla de evaluación para un evaluador. [28]

Los evaluadores fueron un miembro de desarrollo del proyecto (evaluador 1) y un profesional del ámbito legal (evaluador 2).

	tf-idf		doc2vec	
	general	por tipo	general	por tipo
evaluador 1	90 %	50.25 %	100 %	87.5 %
evaluador 2	86.11	49.16 %	94.44 %	71.66 %
promedio	88.05 %	49.71 %	97.22 %	79.58 %

Cuadro 5.1: Evaluación del rendimiento de diferentes métodos para el cálculo de semejanza entre textos de normas, distinguiendo tipos de norma (*por tipo*) y sin distinguirlas (*general*).

	tf-idf		doc2vec	
	general	por tipo	general	por tipo
evaluador 1	83.33 %	70.58 %	53.33 %	45.09 %

Cuadro 5.2: Evaluación del rendimiento de diferentes métodos para el cálculo de semejanza entre texto libre y textos de normas, distinguiendo tipos de norma (*por tipo*) y sin distinguirlas (*general*).

5.2. Análisis de Resultados

5.2.1. Normas relacionadas a normas

En el Cuadro 5.1 observamos que, usando textos completos de normas doc2vec tiene un rendimiento claramente superior a tf-idf. Esto se debe entre otros motivos, a que no necesariamente porque una norma tenga términos iguales a otra indica que estén relacionadas. Es decir, puede que exista relación respecto a la/s palabra/s pero no con la finalidad de la norma, en términos legales. Doc2Vec obtuvo mejor puntaje en medida debido a ésto, por considerar el contexto (en una ventana) de las palabras, a diferencia de tf-idf.

Observamos también que la distinción por tipo de norma afecta negativamente al rendimiento. Este empeoramiento en el rendimiento se da porque, en el caso de diferenciación por tipo, no necesariamente existen textos relacionados en una determinada categoría. Una

posible solución para este problema es utilizar las probabilidades de recomendación para dar o no recomendaciones de un tipo de norma, estableciendo cierto "*threshold*" al menor puntaje considerado (digamos 0.15). Por ejemplo, es probable que al solicitar recomendaciones de "Ley del consumidor" no existan en la base de datos Resoluciones relacionadas, por lo tanto los documentos recomendados van a ser evaluados como malos. También hay que tener en cuenta que el número de documentos recomendados es más grande (3 documentos en el caso de recomendación *general* y 12 documentos en el caso de recomendación *por tipo*), con lo cual aumentan las posibilidades de encontrar documentos no relacionados. En un futuro se subsanaran estos dos problemas.

5.2.2. Normas relacionadas a textos libres

En el Cuadro 5.2 mostramos los resultados cuando se usan textos libres para encontrar normas relacionadas. En general, si el texto objetivo referencia con énfasis a una norma, ésta aparecerá entre los resultados.

En este caso ocurre el fenómeno inverso que en el anterior: tf-idf ofrece mejores resultados que doc2vec. Nuestra hipótesis para este resultado es que doc2vec modela bien los documentos que se usaron para entrenar, y funciona bien para documentos largos pero no para textos cortos. Quizás esto se deba a cómo es implementado el método infer_vector en Gensim, o pues las iteraciones configuradas para el método no son suficientes.

En ambos casos se ha notado que cuanto más largos los textos de consulta, mejor es la recomendación. Sumado a éste último parámetro, si los textos poseen vocabulario del corpus, entonces las recomendaciones por texto mejoran aún más.

En cualquier caso, será necesario en trabajo futuro hacer una exploración de los parámetros de entrenamiento de doc2vec. Los resultados en doc2vec a partir de texto nuevo podrían mejorarse, quizás variando los parámetros del modelo, como también aumentando aún

mas `epochs` en infer_vector(). Aún así el tiempo de respuesta es bastante inferior al de Tf-Idf en este caso.

5.2.3. Otros comentarios

Luego de extensas pruebas, se pudo comprobar que el modelo doc2vec usando el algoritmo PV-DBOW respecto a PV-DM (parámetro dm=1 en la definición para este último) fue mejor en nuestro caso. Aunque no se han notado diferencias marcadas en las recomendaciones in-corpus en cada caso, sí se ha notado que PV-DBOW mejora bastante los resultados en el caso de consultas textuales.

En el cuadro 5.3 se observan los tiempos de entrenamiento para cada motor. Mejoraron notablemente luego de utilizar la tokenización ya almacenada en base de datos, y el uso de concurrencia en el caso del entrenamiento de los motores por tipo de norma.

	tf-idf		doc2vec	
	general	por tipo	general	por tipo
tiempo	1.28 horas	1.54 horas	2.28 horas	9.46 horas

Cuadro 5.3: Tiempos de entrenamiento en JupiterAce (FaMAF).

Capítulo 6

Conclusiones y trabajo futuro

6.1. Aportes

El resultado principal de esta tesis es un sistema de recomendación de textos legales basado en semejanza textual sobre el dominio de la legislación argentina (base documental InfoLeg). Se desplegó el sistema end-to-end, desde la recolección de datos a la interfaz gráfica de usuario final.

A raíz del proyecto, se comprendieron los fundamentos y funcionamiento de formas de representar documentos como vectores: **Paragraph Vector** y **Tf-Idf**.

Se ha logrado implementar una solución adaptada al contenido, la cual brinda recomendaciones para y de textos existentes en una plataforma o inexistentes, lo cual tiene aplicación directa al portal `www.infoleg.gob.ar`, como también adaptable a otra clase de textos, y de gran utilidad para conjuntos de textos no tan estructurados, como por ejemplo fallos judiciales, entre muchos otros.

Es de real interés tanto para estudiantes de Derecho como para avanzados en la profesión por utilidades de aprendizaje, investiga-

ción, y ayuda-memoria al exponer textos semejantes al que se está leyendo. A diferencia del buscador, el recomendador, brinda la posibilidad de seguir consultando contenido de interés (y con una medida de relevancia) de la plataforma, sin realizar búsquedas específicas.

Por otro lado, es notable el desafío de optimizar cada vez más los resultados, sea a través de optimización de datos de entrada, exploración y optimización de parámetros de modelos, e incluso selección del modelo adecuado y su combinación con otros métodos (véase trabajo futuro, 6.2.2).

Las optimizaciones realizadas progresivamente llevaron varias iteraciones teniendo end-to-end's del sistema, con aportes tanto de back-end como de front-end.

Notamos que las recomendaciones obtenidas, en base a las devoluciones y evaluaciones realizadas, son de utilidad y altamente relacionadas al texto objetivo. En particular se destaca el funcionamiento de Doc2Vec para el caso de recomendaciones in-corpus y Tf-Idf para textos por fuera del corpus.

6.2. Trabajo futuro

En la presente sección agrupamos nuevas funcionalidades e ítems a explorar para realizar mejoras al sistema existente.

6.2.1. Mejoras de usabilidad

- Mejoras en los snippets: texto de resumen de cada recomendación que se muestra al usuario.
- Wikificación de los textos, para facilitar la consulta de los documentos referenciados (otras leyes, códigos, etc.) o eventualmente la consulta enciclopédica de los términos que se usan.
- Realizar deploy a Heroku [29], y/o poder acceder a la aplicación públicamente.

6.2.2. Mejoras en la calidad de los resultados

- Incorporar reconocimiento automático de entidades nombradas como preproceso a los textos, en particular, reconocimiento de entidades nombradas específico para el dominio legal de la Argentina.
- Explorar el uso de diferentes *word embeddings* para doc2vec.
- Caracterizar el conjunto de documentos como un grafo, donde las aristas son los documentos y los arcos son las citas entre documentos (modificatorias, menciones, etc.). Sobre este grafo, aplicar técnicas de asignación de relevancia como por ejemplo PageRank.
- Explotar meta-datos de las normas. (Entidades, títulos, resúmenes, etc).
- Exploración de parámetros en doc2vec, tf-idf.
- Explorar el entrenamiento optimizado de motores al agregar nuevas leyes, sin la necesidad de re-entrenarlos completamente (usando modelos existentes).

6.2.3. Mejoras en la cobertura del sistema

- Incorporar diferentes documentos legales (legislación provincial, sentencias de altos tribunales, otros).
- Automatización de la actualización del corpus y entrenamiento del sistema.

Bibliografía

[1] InfoLEG. información legislativa y documental. http://www.infoleg.gob.ar/?page_id=310.

[2] Comercio y Justicia. https://comercioyjusticia.info/blog/category/leyes-y-comentarios/.

[3] Collaborative Filtering. en.wikipedia.org/wiki/Collaborative_filtering.

[4] Similitud del coseno. https://en.wikipedia.org/wiki/Cosine_similarity.

[5] doc2vec article. https://arxiv.org/abs/1405.4053.

[6] Word2Vec paper, Mikolov. https://arxiv.org/abs/1301.3781.

[7] Document Embedding with Paragraph Vectors. https://arxiv.org/abs/1507.07998.

[8] Artículo Medium Doc2Vec. https://medium.com/scaleabout/a-gentle-introduction-to-doc2vec-db3e8c0cce5e.

[9] Rachoud Winkels, Alexander Boer, Bart Vredebregt, and Alexander von Someren. Towards a legal recommender system. In Proc. of the 27th Int'l Conf. on Legal Knowledge and Information Systems, 2014.

[10] Khalid Al-Kofahi, Peter Jackson, Mike Dahn, Charles Elberti, William Keenan, and John Duprey. A document recommendation system blending retrieval and categorization technologies. 2007.

[11] Content Based Recommendation Engine. https://github.com/groveco/content-engine.

[12] tRECS text recommendation system developer built in python and dash by plotly. https://github.com/TeeOhh/tRECS.

[13] word2vec-recommender. https://github.com/manasRK/word2vec-recommender.

[14] Suite Gensim. https://radimrehurek.com/gensim/tutorial.html.

[15] cbtr content based text recommendation. https://github.com/guaq/cbtr.

[16] gutemberg a content-based recommender system for books using the project gutenberg text corpus. https://github.com/jldbc/gutenberg.

[17] tweets gui based recommender system written in python which recommends hash-tags for corresponding text. https://github.com/SuryanshTiwari/Tweet-recommender-system.

[18] Cristian Cardellino. https://crscardellino.github.io/.

[19] Redis. https://redis.io/.

[20] TfIdf Vectorizer, vectorizador Tf-Idf de Scikit Learn. http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

[21] Doc2Vec. https://radimrehurek.com/gensim/models/doc2vec.html.

[22] Similitudes del coseno optimizadas, presentación. https://bergvca.github.io/2017/10/14/super-fast-string-matching.html.

[23] Similitudes del coseno optimizadas, repositorio. https://github.com/ing-bank/sparse_dot_topn.

[24] Dash by Plotly. https://dash.plot.ly/.

[25] Pickle. https://docs.python.org/2/library/pickle.html.

[26] Joblib. https://joblib.readthedocs.io/en/latest/generated/joblib.Parallel.html.

[27] Repositorio del proyecto. https://bitbucket.org/acapello/thesis/src/master/.

[28] Planilla de evaluación para el evaluador 2. https://docs.google.com/spreadsheets/d/1Vt9Z0ttA89uUs6RnD0_igsVtffyth_DBsxjFRDENrbE/edit?usp=sharing.

[29] Heroku. https://www.heroku.com/.

Apéndice

.1. Ejemplos

.1.1. Ejemplo de Ley

HONORABLE CONGRESO DE LA NACION ARGENTINA

CONMEMORACIONES

DIA NACIONAL DEL REMERO

15-oct-2003

Resumen:

DECLARASE DIA NACIONAL DEL REMERO EL 11 DE ABRIL DE CADA AÑO.

Texto Completo:

CONMEMORACIONES

Ley 25.793

Declárase Día Nacional del Remero el 11 de abril de cada año.

Sancionada: Octubre 15 de 2003.

Promulgada de Hecho: Noviembre 10 de 2003.

El Senado y Cámara de Diputados de la Nación Argentina reunidos en Congreso, etc. sancionan con fuerza de Ley:

ARTICULO 1º — Declárase Día Nacional del Remero el 11 de abril de cada año, fecha conmemorativa del natalicio de don Alberto Demiddi.

ARTICULO 2º — Comuníquese al Poder Ejecutivo.

DADA EN LA SALA DE SESIONES DEL CONGRESO ARGENTINO, EN BUENOS AIRES, A LOS QUINCE DIAS DEL MES DE OCTUBRE DEL AÑO DOS MIL TRES.

— REGISTRADA BAJO EL Nº 25.793 —

EDUARDO O. CAMAÑO. — JOSE L. GIOJA.

.1.2. Ejemplo de fragmento libre

Se aprobó "Ley Justina": los mayores de edad son donantes de órganos presuntos

Por unanimidad, la Cámara de Diputados le dio sanción definitiva al proyecto de "ley Justina", que dispone que todas las personas mayores de 18 años sean donantes de órganos o tejidos, salvo que en vida dejen constancia expresa de lo contrario.

El proyecto, que recibió 202 votos afirmativos y que también había sido aprobado por unanimidad en el Senado, está inspirado en el caso de Justina Lo Cane, una menor de 12 años que murió en noviembre pasado en la Fundación Favaloro mientras aguardaba un trasplante de corazón.

La contribución fundamental de la reforma es que invierte el proceso por el cual las personas pasan a integrar el registro de donantes: al crearse la figura del "donante presunto", ya no se requiere dejar voluntad expresa por la afirmativa sino que se garantiza "la posibilidad de realizar la ablación de órganos y/o tejidos sobre toda persona capaz mayor de 18 años, que no haya dejado constancia expresa de su oposición a que después de su muerte se realice la extracción de sus órganos o tejidos".

En el caso de los menores de edad, "se posibilita la obtención de autorización para la ablación por ambos progenitores o por aquel que se encuentre presente".