Proyecto final

NLP aplicado a análisis de texto

Alias, Gerardo

Cassanelli, Rodrigo

Directora: Ana Haydee Di Iorio

Universidad Nacional de Mar del Plata
Facultad de Ingeniería

12 de agosto de 2019

1. Introducción y objetivos

El correo electrónico, las aplicaciones de mensajería instantánea y las redes sociales se han convertido en un elemento cotidiano de la vida de cada persona. Es habitual que en ellos se registren referencias de las preferencias, actividades e ideologías de cada individuo. Esta situación no resulta ajena a las conductas delictivas; en la actualidad, los dispositivos digitales pueden contener gran cantidad de elementos probatorios para una investigación judicial. Esta situación hace necesario que una investigación requiera de la extracción y análisis de los datos almacenados en los dispositivos electrónicos de un individuo u organización.

Si bien se pueden encontrar audio, imágenes y video, la mayor parte de la información contenida en un dispositivo corresponde a texto digital. El volumen de información en dicho formato es, por lo general, extremadamente grande. Citando como ejemplo al caso de Nahir Galarza (Gualeguaychú, Argentina, año 2017), cuando se analizó el contenido del teléfono móvil de Nahir, se encontraron más de 100.000 mensajes en un solo hilo de conversación. Según las estadísticas, una persona recibe y envía en promedio 20.000 mensajes por mes. Dicha estadística, sólo contempla aplicaciones de mensajería; una investigación judicial abarca, además, documentos, correo electrónico, etc.

Un agravante de esta situación es la cantidad de información útil que se puede obtener respecto del total extraído. Según la "Guía Integral de Empleo de la Informática Forense en el Proceso Penal", Res PG SCBA 483/16, la información que se puede incluir en una causa debe ser relevante para la misma. Con esta premisa, una causa de narcotráfico solo puede incluir aquellos datos extraídos referidos a esta temática. Por esta razón, todo dato extraído debe ser analizado y seleccionado de modo de incluir únicamente aquellos relevantes.

Los relevamientos de institutos especializados tales como el InFo-Lab, laboratorio especializado en informática forense de la ciudad de Mar del Plata, muestran que la necesidad de extracción y análisis de información tiene una tendencia creciente. No obstante, esta tendencia no ha sido acompañada por las técnicas de análisis. Generalmente la información extraída de cada dispositivo debe ser analizada en forma artesanal; es decir, un investigador o perito debe leer en forma sistemática la información en busca de indicios o evidencia relevante para el caso.

La aplicación de una solución informática para detectar evidencia de determinadas conductas en textos digitales podría significar un aumento drástico en la eficiencia y la calidad de este tipo de tareas. Sin embargo, el lenguaje natural y, especialmente, aquel

de uso cotidiano presenta un gran desafío para las técnicas tradicionales que ofrece la informática (comúnmente conocidas como *basadas en reglas*). Intentar resolver este problema por medio de búsquedas tradicionales o, incluso, el uso de expresiones regulares puede verse truncado por la enorme variabilidad de los textos. Además, el uso de este tipo de técnicas requiere de conocimientos en informática de los cuales el perito o investigador judicial generalmente carece. Del mismo modo que un especialista en informática carece del conocimiento específico o "know how" que sí tiene el investigador.

El procesamiento de lenguaje natural o NLP (del inglés Natural Language Processing) es un campo de las ciencias de la computación. Si bien no fue concebido como una rama de la inteligencia computacional, las características del problema a resolver hicieron que paulatinamente se pasará de modelos basados en reglas a modelos que utilizan redes neuronales. En la actualidad, la mayoría de los sistemas orientados al análisis de textos basan su funcionamiento en la utilización de redes neuronales. Este tipo de sistemas "aprende", es decir se entrena, para luego inferir resultados basándose en el contexto en lugar de reglas particulares.

El avance en las técnicas de procesamiento del lenguaje natural hace posible pensar en una solución informática a la problemática antes descripta. En el InFo-Lab , a principios del año 2018, se realizó una prueba de concepto o prototipo para analizar la factibilidad de la aplicación de estas técnicas a las investigaciones criminales. Dicho prototipo obtuvo buenos resultados detectando referencias a drogas o estupefacientes en diversos textos.

Con dicho trabajo como base, el objetivo del presente proyecto es desarrollar un sistema informático orientado a la resolución de este tipo de problemas. Para ello se plantea un producto dividido en dos módulos principales. En la figura 1, se puede observar un esquema de alto nivel de la solución propuesta y de la interacción con cada uno de los componentes principales del sistema.

El primero de ellos es un módulo de administración de modelos para el análisis de lenguaje natural. El mismo se orienta a los usuarios con conocimientos medios / avanzados de informática. Su objetivo es ofrecer interfaces para:

- Analizar los textos y, de este modo, permitir al investigador focalizar su atención sobre los fragmentos con mayor probabilidad de contener evidencia.
- Permitir al administrador del sistema administrar los diferentes modelos, orientados a temáticas particulares, con los que cuente el sistema.
- Permitir crear modelos especializados en distintas temáticas.

- Permitir administrar los datos de entrenamiento de cada modelo y, al mismo tiempo, ofrecer un mecanismo para aplicar los mismos a los modelos.

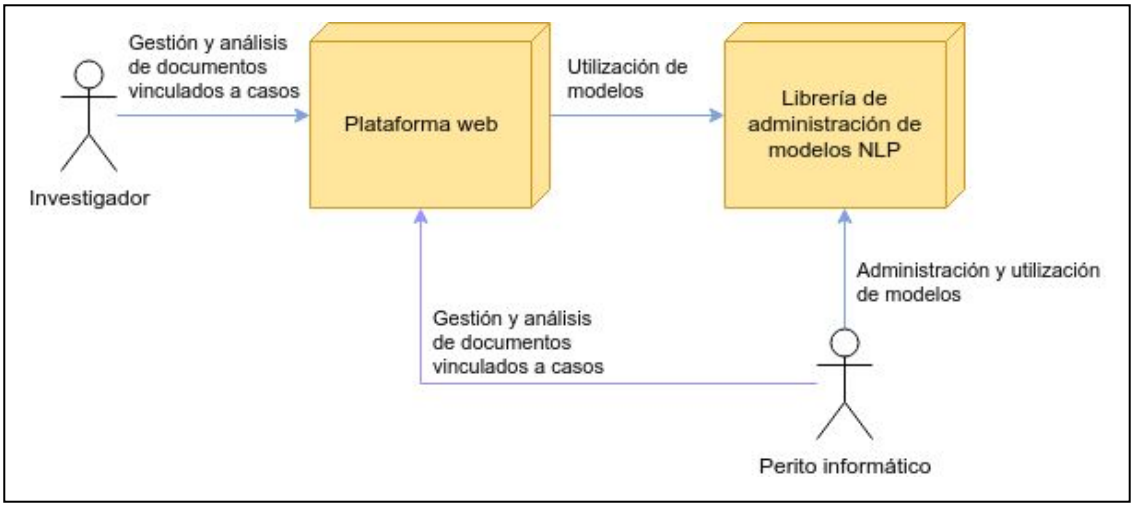

Figura 1. Esquema general de la solución propuesta.

El segundo módulo es una plataforma web que, utilizando las funcionalidades provistas por la librería de NLP, permite a distintos tipos de usuarios, tales como abogados, fiscales e investigadores utilizar modelos para analizar distintos documentos. Dado el tipo de usuario esperado, este sistema debe poder ser operado con conocimientos básicos de informática. Sus principales objetivos son:

- Permitir gestionar los documentos vinculados a investigaciones que lleva a cabo el Ministerio Público, en primera instancia, de manera sencilla y útil.
- Ofrecer al menos un modelo de análisis base orientado a una temática criminal particular.
- Permitir a los usuarios analizar, de forma amigable, los textos asociados a una investigación pertinente.
- Generar informes que muestren información considerada de importancia respecto de una investigación.

2. Marco Teórico

2.1. Procesamiento del lenguaje natural

El procesamiento del lenguaje natural es una rama de la inteligencia artificial que busca entender y procesar datos del lenguaje. Su campo de acción remite a la interacción entre las computadoras y el lenguaje humano, comúnmente denominado lenguaje natural. Particularmente, se ocupa de crear programas o aplicaciones capaces de analizar grandes cantidades de datos en dicho lenguaje.

A pesar de que existen trabajos previos, se puede determinar que los orígenes del procesamiento del lenguaje natural datan de la década del 50'. En 1950, Alan Turing publicó un artículo titulado "Inteligencia", en el cual proponía al "test de Turing" como un criterio de inteligencia. Dicho test consta de una variación de un juego en el cual existen tres partes: por un lado hay un interrogador y, por otro, un hombre y una mujer. Ninguna de las partes puede verse y la comunicación sólo puede realizarse mediante texto. El juego consiste en que el interrogador haga preguntas y, en base a las respuestas, determine quién es el hombre y quién es la mujer. Turing propuso una variante a este juego, en lugar de un hombre y una mujer, habría un humano y una máquina. Si al final del juego, el juez no pudiese diferenciar al humano de la máquina, entonces la máquina ganaría. Al final de este artículo Turing concluyó que, para finales del siglo XX, era posible que existiesen máquinas capaces de jugar a dicho juego.

Con el pasar de los años, varios proyectos experimentaron con el procesamiento del lenguaje natural. Entre 1954 y 1966 se desarrolló el "experimento de Georgetown", el mismo buscaba la traducción automática de textos en ruso a inglés. Este proyecto se desestimó luego de que, tras más de 10 años, no se haya podido obtener avances significativos. En la década del 60' hubo avances notables en los sistemas de procesamiento de lenguaje natural: una simulación de psicoterapeuta (ELIZA) y un sistema de lenguaje natural que funcionaba con bloques de palabras y vocabularios restringidos (SHRDLU). ELIZA fue incluso capaz de mostrar ciertos atisbos de interacción humana.

Los desarrollos continuaron sucediéndose durante toda la década del 70'. El paradigma predominante durante esta etapa fue la idea de estructurar la información del mundo real de modo de que fuera entendible para las computadoras. Para principios de los 80', los sistemas de procesamiento de lenguaje natural se basaban en conjuntos de reglas escritas a mano. En la mayoría de los casos estos conjuntos resultaban altamente complejos.

Sin embargo, en los últimos años de dicha década apareció el concepto de aprendizaje de máquina o "machine learning". El incremento en el poder de procesamiento de las computadoras, sumado a la disminución en el predominio de las teorías lingüísticas de Chomsky, hizo que los esfuerzos en el área del procesamiento de lenguaje natural se vuelquen hacia este concepto. En este punto aparecen las técnicas de part-of-speech tagging (POS) y los modelos estadísticos. Estos últimos están basados principalmente en las redes neuronales, una técnica de la inteligencia computacional, y toman decisiones probabilísticas a partir de determinadas características de los datos. Este tipo de modelo ha demostrado un comportamiento más robusto cuando deben tratar con datos desconocidos o no familiares.

Las investigaciones recientes se han enfocado en cómo aprenden estas redes, es decir, cómo ajustan los pesos para evaluar los datos de entrada. Los modelos probabilísticos se pueden entrenar de forma supervisada, semi supervisada o no supervisada. Mientras que el entrenamiento supervisado requiere que la totalidad de los datos contengan anotaciones, la cantidad de datos con anotaciones disminuye en el entrenamiento semi supervisado y se elimina completamente en el caso de no supervisado.

En la actualidad, las aplicaciones del procesamiento del lenguaje natural son diversas. Si bien este tipo de tareas se encuentra comúnmente entrelazada, existen muchas aplicaciones tanto a nivel sintáctico como semántico. Algunas de las aplicaciones sintácticas son:

- **Lematización**: esta aplicación consiste en obtener la forma base de una palabra. Es el equivalente a eliminar conjugaciones o plurales. Por ejemplo, la palabra "escribiendo" en español tendría como *lema* escribir.
- **Segmentación morfológica:** se basa en obtener los *morfemas* de cada palabra. Esta técnica incrementa su complejidad en función del idioma. Es así como resulta sencilla en un idioma simple como el inglés, pero puede resultar incluso imposible en otros como el hindú. Como ejemplo, la palabra "chico" en español se divide en la raíz "chic" y el afijo "o".
- **Part-of-speech tagging:** consiste en determinar la función de una palabra dentro de una oración. Así como la palabra "juego" podría ser un verbo en un contexto como "Yo juego al fútbol", en otro contexto puede ser tomado como un sustantivo: "Es un juego muy divertido".
- **Parseo:** esta aplicación se utiliza para el análisis sintáctico o gramatical de una oración. Se basa en determinar la relación existente entre las palabras para, de este modo, determinar predicados, sujetos, etc.

- **Separación de oraciones:** si bien las oraciones son separadas normalmente por signos de puntuación, muchas veces esos signos de puntuación también pueden tener otro significado. Esta aplicación del NLP se encarga de separar correctamente las oraciones de un texto.
- **Stemming:** se enfoca en la obtención de la palabra raíz o forma base de otra. De este modo "hacer" sería la palabra raíz para "hizo" o "hacía". Esta aplicación puede apreciarse sensiblemente similar a la *lematización* y de hecho algunas herramientas de NLP las unifican.
- **Segmentación de palabras:** se trata de la separación de un texto en tokens o palabras. En algunos idiomas esto podría verse como algo trivial, sin embargo en otros no lo es. Además, también han de tenerse en cuenta las contracciones o palabras compuestas que pudieran presentar algunos idiomas como el inglés.
- **Extracción de terminologías:** esta aplicación consiste en extraer términos relevantes del texto.

Existen, además, otros tipos de aplicaciones para estas técnicas. En el caso de las aplicaciones semánticas, se pueden encontrar:

- **Traducción automatizada:** consiste en la traducción desde un lenguaje humano a otro. Es uno de los problemas más complejos a resolver, aunque se han hecho importantes avances.
- **Reconocimiento de entidades (NER):** esta aplicación se basa en detectar dentro de un texto entidades tales como nombres de personas, organizaciones, localizaciones, productos, etc.
- **Generación de lenguaje natural:** se basa en la conversión de los datos almacenados en una base de datos, o indentaciones semánticas, a un lenguaje legible para los humanos.
- **Entendimiento de lenguaje natural:** consiste en convertir segmentos de texto en representaciones tales como la lógica de primer orden.
- **Reconocimiento de texto en imágenes (OCR):** esta aplicación busca obtener texto a partir del texto que pudiese contener una imagen.
- **Respuesta de preguntas:** consiste en que la computadora pueda responder a preguntas formuladas en lenguaje humano.
- **Reconocimiento de implicación textual:** determina la relación entre dos fragmentos de texto en función de si el hecho de que una sea cierta podría resultar en que sea falsa la otra o viceversa.

- **Extracción de relaciones:** en este caso se busca que a partir de un conjunto de texto se pueda determinar la relación entre dos entidades del mismo.
- **Análisis de sentimiento:** su objetivo es polarizar determinados elementos de un texto. Por ejemplo una aplicación sería analizar si un texto posee una connotación positiva o negativa o una palabra está más relacionada con la violencia o con la paz.
- **Identificación de temas:** permite identificar el tema de una parte de un texto.

Existen además otros tipos de aplicaciones, más especializadas, orientadas a resumir o analizar. Además, dentro de este campo de la informática también encontramos a las técnicas para convertir sonido en texto (o viceversa) y para dialogar. Un ejemplo práctico de estas últimas son los *chatbots*.

2.2. Redes neuronales

Una red neuronal es un sistema artificial que intenta emular la forma en que funciona un cerebro biológico. Estos sistemas intentan emular tres conceptos clave: el procesamiento paralelo, la memoria distribuida y la adaptabilidad al entorno. Mientras que en un sistema biológico el elemento fundamental es la neurona, una red neuronal está compuesta por neuronas artificiales. Dichas neuronas se configuran en un arreglo similar al arreglo jerárquico que presenta un cerebro.

Cada neurona de una red neuronal artificial puede verse como una función matemática. Dicha función es conocida como *función de activación* y debe su nombre a que se trata de una función que, a partir de un conjunto de entradas, devuelve un determinado valor que indica si la neurona está o no activada. En la figura 2, se puede observar el esquema básico de una neurona. Como puede observarse la función de activación recibe la sumatoria de las entradas multiplicadas por sus respectivos pesos. El parámetro b_k se denomina *bias* y determina un grado adicional de decisión. En última instancia, la sumatoria de las entradas en conjunto con el *bias* son evaluados por la función de activación. Como lo indica su nombre, esta función es utilizada para determinar si la neurona fue activada o no, hecho reflejado en la salida y_k.

El modelo más simple de neuronas es el modelo de *todo o nada*. En este caso, la función de activación devuelve uno (activado) o cero (no activado) en función de si la sumatoria de las entradas alcanza el umbral de activación o no, cuyo valor es cero en este caso. En la figura 3, se puede ver cómo se evalúa la función de activación de tipo *todo o*

nada. Existen otros tipos de funciones que permiten obtener diferentes resultados e incluso salidas continuas para las neuronas.

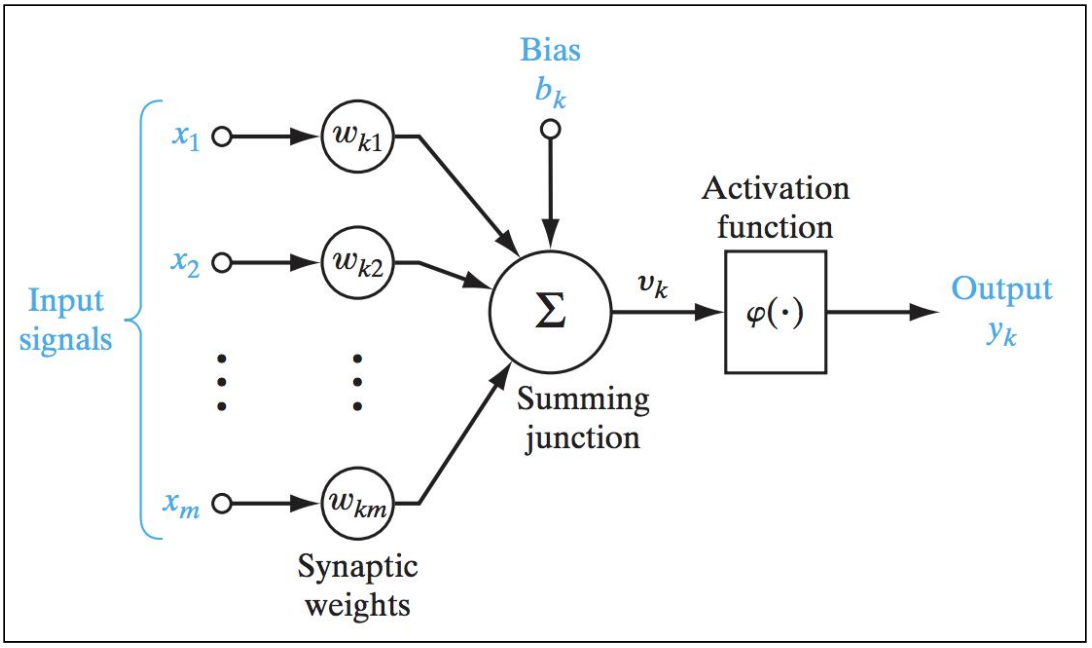

Figura 2. Modelo de una neurona artificial no lineal.[1]

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k$$

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

Figura 3. Función de activación del tipo *todo o nada*.[1]

Las redes neuronales se pueden organizar en diferentes arquitecturas o topologías que representan el modo de conexionado de la red. Cada neurona artificial se conecta con las demás por medio de sinapsis. De este modo, el comportamiento de la red queda definido por la estructura de las conexiones sinápticas. Las conexiones son *direccionales*. Debido a esto, la información sólo puede propagarse en un sentido. Finalmente, las neuronas se organizan en una estructura de capas. Existen tres tipos de capas: de *entrada*, de *salida* y *ocultas*. La característica principal de las capas ocultas es que no tienen conexión con el entorno exterior.

[1] Haykin. S. (2009). *Neural Networks, A Comprehensive Foundation* (3ª Edición).

A partir de los conceptos anteriores, se pueden establecer distintos tipos de arquitecturas. Las arquitecturas pueden definirse a partir del número de capas o del flujo de datos. Considerando el número de capas, se distinguen las redes neuronales *multicapa* y las *monocapa*. Por otro lado, basándose en el flujo de datos, las redes pueden clasificarse en *redes unidireccionales (feedforward)* y *redes recurrentes* o *retroalimentadas (feedback)*. En la figura 4, se pueden observar dos ejemplos distintos de arquitecturas basadas en los conceptos enunciados.

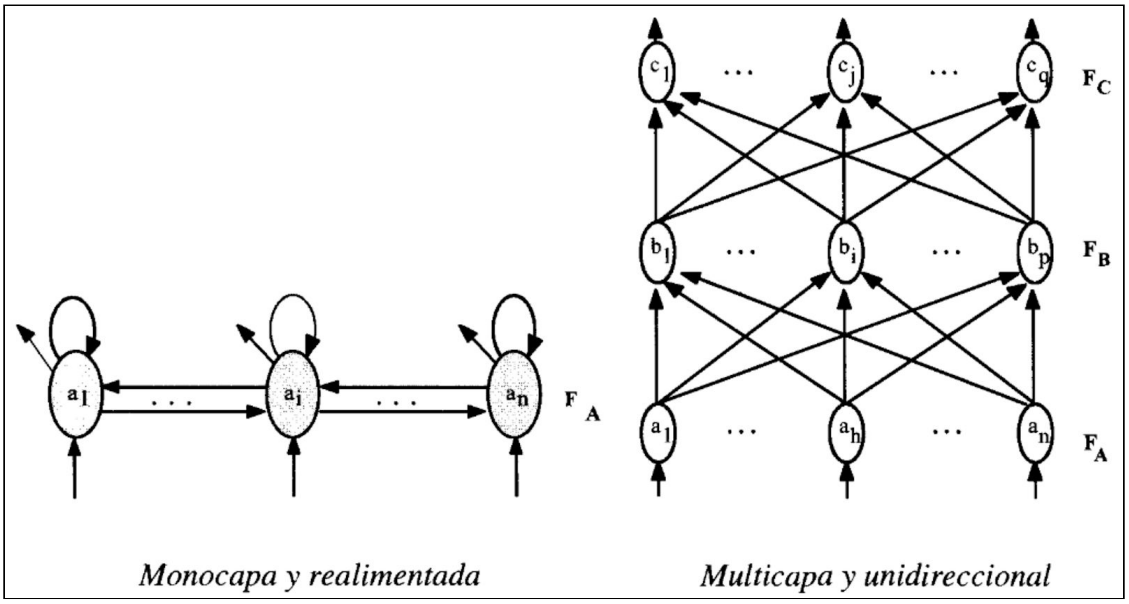

Figura 4. Ejemplos de arquitecturas para redes neuronales.[2]

Cabe destacar que la arquitectura de la red la hace más o menos útil para detectar una región de decisión particular. Mientras que el perceptrón simple solo sirve para un hiperplano de detección, existen diferentes arquitecturas específicas para diferentes planos de detección. En la figura 5 se pueden ver distintos ejemplos de arquitecturas y los planos de detección asociados para los cuales son útiles. El primer ejemplo corresponde a un perceptrón simple. El segundo corresponde a un perceptrón multicapa con una única capa oculta. Finalmente, el tercer ejemplo corresponde a un perceptrón multicapa con dos capas ocultas.

En la figura 6, se puede observar el esquema de un perceptrón multicapa unidireccional. La entrada de cada capa se ve representada por el valor de las entradas multiplicadas por los pesos. A su vez, cada nodo tendrá una función de activación. Mediante la combinación de los valores de salida de la capa anterior, afectados por los

[2] Martín del Brío y B., Sanz Molina, A. (2001). *Redes Neuronales y Sistemas Borrosos* (3ª edición).

respectivos pesos de cada conexión, y la función de activación se determinan los nodos que se activarán de cada capa. De este modo, una determinada señal de entrada atravesará la red activando determinados nodos y produciendo, en última instancia, un valor de salida. Este mecanismo es similar al mecanismo biológico por el cual las señales atraviesan las neuronas de un organismo. Suponiendo un esquema similar al de la figura 6, ante una señal entrante el nodo $X_0(1)$ recibirá la sumatoria de los valores de las entradas ponderadas por los pesos de las conexiones.

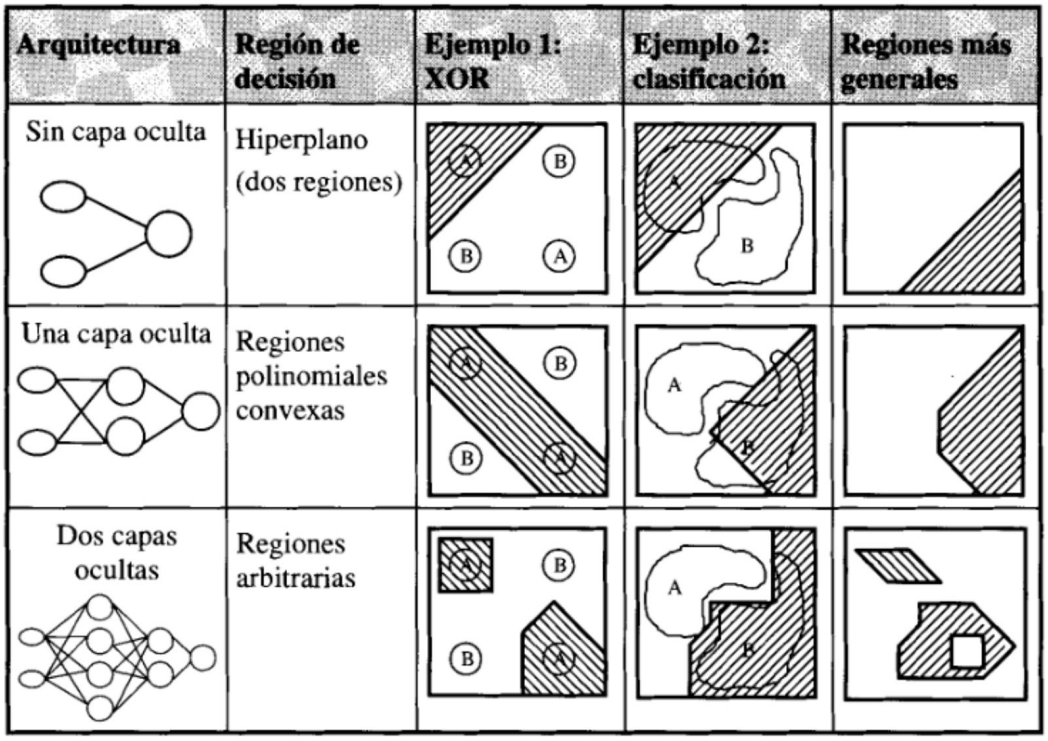

Figura 5. Regiones de decisión obtenidas para distintas arquitecturas de perceptrón.[3]

El valor inicial de los pesos de cada conexión es aleatorio. En este punto aparece el concepto de *aprendizaje de máquina*. El proceso de entrenar una red neuronal consiste en aplicar un algoritmo que ajuste los pesos de las conexiones entre las neuronas para que estas respondan correctamente a un determinado patrón de entrada. Existen diversos algoritmos para el entrenamiento de las redes neuronales. Sin embargo, todos estos tienen en común el concepto de que se debe aplicar los patrones de entrada e ir ajustando los pesos de modo que se haga converger el error de la salida a un valor razonable. El *valor de ritmo de aprendizaje ($\mathcal{E}$)* determina la velocidad de convergencia del proceso de aprendizaje. En la práctica, este valor debe adoptar un valor óptimo, ya que si es

[3] Martín del Brío y B., Sanz Molina, A. (2001). *Redes Neuronales y Sistemas Borrosos* (3ª edición).

demasiado pequeño el aprendizaje será lento y, por el contrario, si es muy grande puede conducir a oscilaciones muy grandes de los pesos. Estas oscilaciones no son aconsejables.

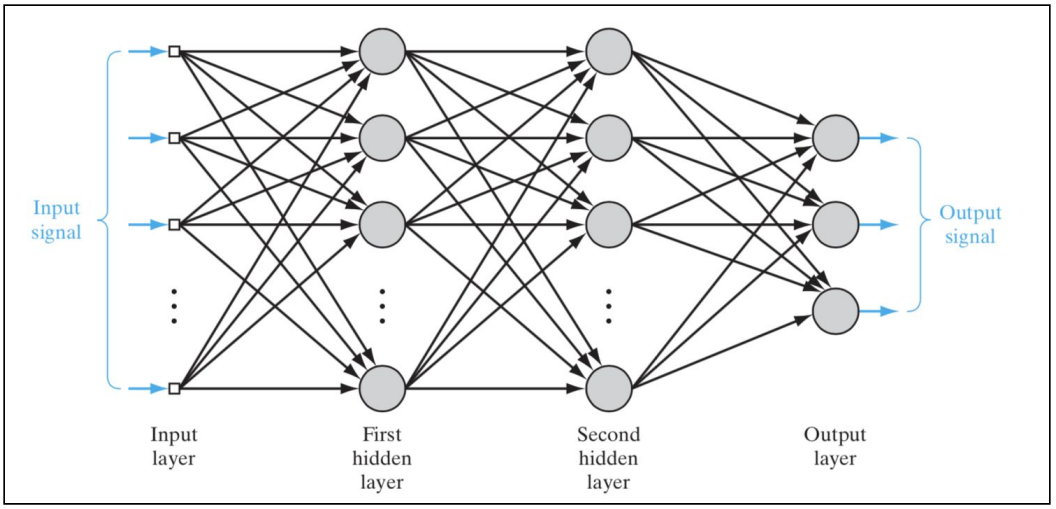

Figura 6. Esquema de un perceptrón multicapa unidireccional.[4]

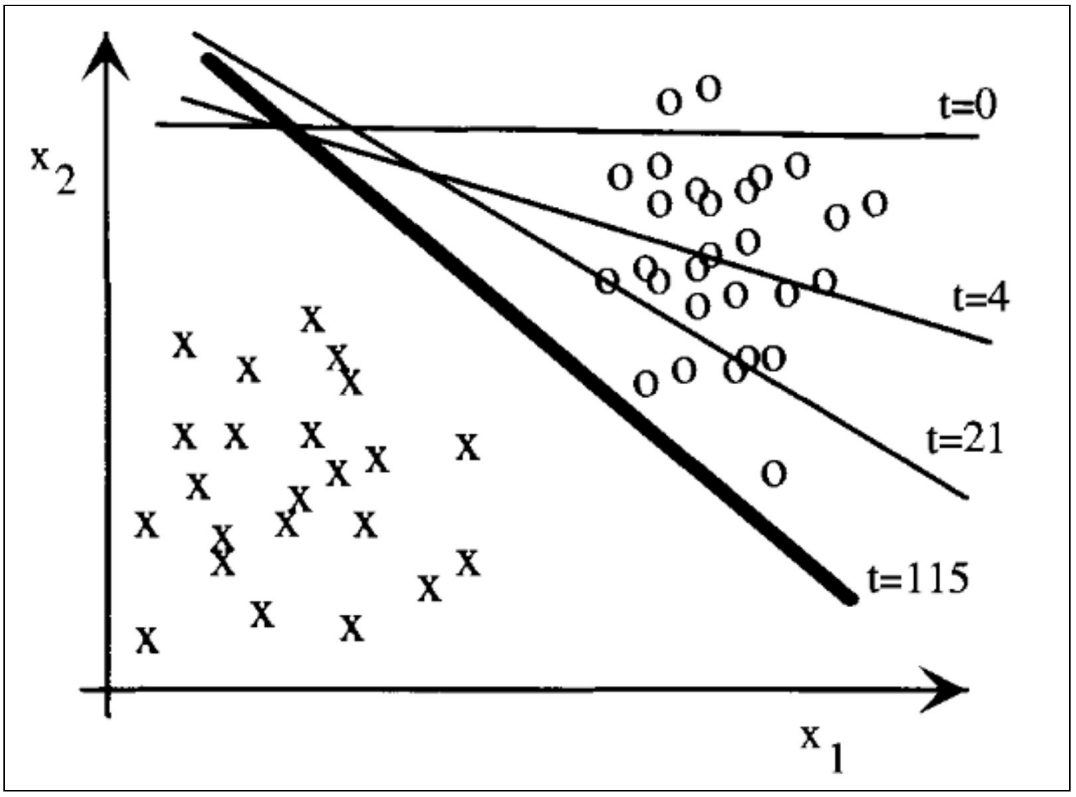

Figura 7. Evolución de las regiones de decisión establecidas por el perceptrón simple.[5]

[4] Haykin. S. (2009). *Neural Networks, A Comprehensive Foundation* (3ª edición).
[5] Martín del Brío y B., Sanz Molina, A. (2001). *Redes Neuronales y Sistemas Borrosos* (3ª edición).

En la figura 7 se puede apreciar la evolución de los patrones detectados por un perceptrón simple. En el mismo se puede apreciar como varía el hiperplano de aceptación con el correr del tiempo.

Las redes neuronales han probado ser eficientes para clasificar, procesar, filtrar y agrupar datos y patrones. También han resultado útiles para el modelado, análisis predictivo y aproximación de funciones. Sus aplicaciones prácticas abarcan las áreas de la biología, empresa, medio ambiente, finanzas, manufactura, medicina y militar. En los últimos años, la aplicación de redes neuronales se ha extendido al ámbito de la justicia y las fuerzas de seguridad. Un ejemplo de estos últimos es la aplicación de redes para el reconocimiento facial.

En el caso particular del procesamiento del lenguaje natural, el aporte de las redes neuronales ha posibilitado grandes avances. Su aporte ha posibilitado resolver, parcial o totalmente, varios problemas que no había sido posible resolver con los sistemas basados en reglas.

3. Estado del Arte

3.1. Aplicaciones informáticas aplicadas a las investigaciones judiciales

En la actualidad, el mercado ofrece varias *suites informáticas* orientadas a resolver problemas propios de las investigaciones. En su mayoría, estas aplicaciones están enfocadas a la gestión y recuperación de información. Por un lado, se encargan de la gestión, es decir, de la administración del proceso completo que involucra la pericia informática. Por otro lado, estas suites cumplen con la función de realizar la extracción de los archivos alojados en los medios de almacenamiento de cada dispositivo involucrado. Las funcionalidades de análisis ofrecidas por estos paquetes suelen ser limitadas o acotadas a un problema particular. Algunos ejemplos de ellas son:

- **Forensic toolkit (FTK):** se trata de una suite orientada a la informática forense. Se la publicita como una aplicación altamente performante y estable, capaz de realizar búsquedas rápidas y que almacena toda la información en una base de datos. Además ofrece la posibilidad de ser integrada con otras aplicaciones para extender sus funcionalidades. Está orientada a la extracción de datos de diversos dispositivos por medio de diversas técnicas, esto incluye las técnicas necesarias para obtener datos protegidos con contraseña o encriptados. Ofrece, además,

algunas técnicas para relacionar patrones en la información extraída. El valor de una licencia puede llegar a superar los US$ 10.000,00.

- **EnCase Forensic:** esta suite ofrece características similares a las de FTK. Ofrece diferentes funcionalidades para recuperar archivos desde diferentes sistemas de archivos, incluso de aquellos que se encuentran encriptados. Además, ofrece soporte para recuperación de archivos de servicios de Microsoft tales como Microsoft Office 365. Se lo promociona como una herramienta premiada por varios años como la mejor aplicación de informática forense del mercado varias veces. Además de la capacidad de extraer datos, permite administrar el flujo de trabajo del caso. Permite procesar la información, indexar los datos extraídos y programar consultas para optimizar las búsquedas. Finalmente, provee algunas herramientas para el análisis de la información y la confección de reportes. Como en el caso de FTK, las licencias tienen valores similares.
- **Oxygen Forensics:** otra suite de aplicaciones que ofrece múltiples funcionalidades para la extracción de datos. Presenta algunas posibilidades que las otras suite no presentan tales como la extracción de datos de *servicios en la nube* o dispositivos *IoT*. Además, en sus últimas versiones ofrece funcionalidades de análisis para relacionar los registros de llamadas o contactos de un dispositivos. Oxygen no se ofrece solo como un producto sino que también se ofrecen capacitaciones.
- **Autopsy forensics:** si bien se lo publica como un gestor de casos. Autopsy ofrece un *pipeline* que lleva a cabo diversas tareas dentro de un proceso. Ofrece la posibilidad de agregar plugins que agregan o modifican una tarea del proceso. Se trata de un software libre y, en la actualidad, es el elegido para las investigaciones en los institutos de Mar del Plata y la Ciudad Autónoma de Buenos Aires. Los usuarios de esta aplicación manifiestan que la misma presenta serios problemas en cuanto a su performance; la demanda de recursos, para los equipos en los que es utilizado, es generalmente excesiva.

Como puede observarse, no se ha encontrado ningún software que combine las funcionalidades de análisis de texto y gestión de documentos requeridas por el usuario. Lo más cercano que pudo hallarse son sistemas que permiten realizar búsquedas por expresiones regulares (necesitando conocimientos específicos para generarlas) o búsquedas literales, como lo permite cualquier procesador de texto.

3.2. Procesamiento de lenguaje natural y búsqueda forense

La búsqueda forense es un campo relativamente nuevo de la informática forense. Se enfoca principalmente en el análisis de datos creados por el usuario tales como email, documentos de las suites ofimáticas o documentos en formato PDF. Dado que es un campo nuevo, aún no existen demasiadas aplicaciones que incursionen en él. La mayor parte de los trabajos son aislados o creados por especialistas para satisfacer necesidades particulares.

Durante búsqueda de aplicaciones similares a la que el presente proyecto plantea, se ha encontrado el proyecto LES. El mismo fue desarrollado en la Unión Europea en el 2011 (publicado en el año 2015) y tiene como objeto analizar la performance de las técnicas de procesamiento de lenguaje natural en casos de investigación. Se enfoca en aplicar la capacidad de reconocimiento de entidades (NER) a casos de fraude económico. El proyecto combina la extracción de datos, indexación de los archivos obtenidos y, finalmente, aplica el reconocimiento de entidades para encontrar términos claves que permitan relacionar. Este trabajo es de las pocas aproximaciones existentes que utilizan procesamiento de lenguaje natural para abordar el problema. Sin embargo, no deja de ser una aproximación teórica o potencial al problema.

La Subsecretaría de Informática del Ministerio Público de la Provincia de Buenos Aires ha estado comenzando a investigar sobre la factibilidad de aplicar LUIS. Este sistema, desarrollado por la empresa Microsoft, se encuentra orientado al procesamiento de lenguaje natural para la creación de chatbots. La plataforma se enfoca en detectar las principales palabras de una frase. Esto, permite obtener una idea general de cuál es la intención u objetivo de la misma. Recientemente se ha agregado la capacidad de reconocer entidades nombradas (NER). Si bien puede ser una buena aproximación al problema, la posibilidad de personalizarlo es limitada y posee poca flexibilidad ante aquellos conjuntos de datos con mala ortografía o baja calidad de escritura. Otra gran desventaja radica en que el sistema requiere de una licencia que cual aplica pagos a nivel transacción, hecho que genera un costo variable a cada investigación.

3.3. Módulos de procesamiento de lenguaje natural

Existen múltiples módulos que ofrecen funcionalidades orientadas al procesamiento de lenguaje natural. Cada uno de ellos cuentan con prestaciones específicas y orientadas a la resolución de un problema particular.

Existen módulos orientados a múltiples funciones tales como reconocimiento de voz, *chatbots*, traductores, correctores, etc. Sin embargo, en el marco del presente proyecto, solo son relevantes aquellos módulos que permiten realizar un análisis del texto. Entre los mejores módulos para procesamiento de lenguaje natural podemos encontrar:

- **CoreNLP:** este módulo fue desarrollado por el grupo Stanford. Está escrito en Java y es conocido por su excelente velocidad. Ofrece funciones tales como Part-of-Speech (POS), etiquetado, aprendizaje de patrones y reconocimiento de entidades entre otras. A pesar de estar escrita en Java, posee wrappers para utilizar otros lenguajes tales como Python. Es una librería ampliamente usada en ambientes de producción y su implementación ha sido mejorada con el pasar de los años.
- **NLTK:** se trata de una de las librerías de NLP más conocidas. Tiene un diseño modular que lo hace ideal para adaptarse a distintas problemáticas por medio de la composición de módulos. NLTK ha ayudado a resolver varios problemas asociados a diferentes aspectos del procesamiento de lenguaje natural. Existen libros y guías que sirven como ayuda para poder utilizar este módulo. Involucrarse con NLTK sin leer estas guías es casi una tarea imposible; este módulo tiene una curva de aprendizaje compleja y posee varias limitaciones internas. Este último aspecto constituye en su principal desventaja.
- **TextBlob:** esta librería no puede considerarse como una librería NLP por sí misma. En su lugar, se trata de una interfaz montada sobre NLTK y que agrega componentes tales como analizadores de sentimiento. Su uso permite resolver parte de la complejidad de NLTK, al mismo tiempo que permite crear rápidamente prototipos.
- **Gensim:** este módulo presenta limitaciones en cuanto a la versatilidad que tienen los enumerados anteriormente. Sin embargo, su punto fuerte radica en la comparación entre distintos documentos. Gensim permite realizar un análisis que determine las similitudes entre dos documentos.
- **SpaCy:** esta librería *open source* está escrita en Cython. Mientras que NLTK ofrece más de 50 variantes para abordar la solución de un problema, SpaCy solo ofrece una. Si bien esto puede parecer contraproducente, se compensa porque en lugar de tener que encontrar el camino óptimo, en la mayoría de los casos la solución provista por SpaCy es la mejor. En términos de rendimiento, se puede considerar que esta librería tiene una performance similar al de una librería escrita en C.

4. Metodologías y/o herramientas utilizadas

4.1. Metodologías

4.1.1. Metodologías para el desarrollo del software

Durante, y previo al comienzo del desarrollo de software, es indispensable la necesidad de disponer de alguna metodología que permita realizar, de forma sencilla, la planificación y seguimiento del proyecto.

Existen innumerables metodologías que resultan aplicables en la actualidad. Sin embargo, aquellas que se encuentran en auge, desde hace ya tiempo, y que son mayormente implementadas, son las metodologías ágiles. Estas mismas, tienen como principio fundamental el desarrollo iterativo e incremental de software, donde los requisitos y soluciones evolucionan según la necesidad del proyecto.

Dentro de los métodos ágiles más utilizados en la actualidad se encuentran: Kanban, Scrum, Scrumban, Programación Extrema (XP), entre otros.

Si bien se eligió utilizar el modelo de Scrum, resultó ser imposible ejecutarlo de forma pura. Esto puede observarse ya desde un principio debido a la cantidad de roles mínimos (Scrum Master, el Product Owner, stakeholders y equipo de desarrollo), necesarios, para implementar una metodología Scrum. No obstante, a pesar de ser un equipo compuesto por dos integrantes únicamente, la esencia del modelo trato de ser respetada en todo momento.

Dentro del proceso de desarrollo del sistema, resultó indispensable establecer un calendario con reuniones asiduas con los clientes. Tales, no sólo sirvieron para la presentación e interacción de los los incrementos finalizados, sino que también derivaron en la principal fuente de retroalimentación del sistema. A su vez, como la metodología lo estipula, en estas reuniones, las cuales se llevaron a cabo en su gran mayoría en el momento de finalización de un sprint, era en los momentos en los cuales los clientes tenían la oportunidad de no sólo expresar el deseo de adicionar nuevas funcionalidades deseadas para el sistema sino también el de modificar algunas ya desarrolladas.

En cada sprint, ambos integrantes trabajaron sobre requerimientos distintos de manera independiente, sin implicar esto que ante la aparición de percances o dificultades durante el mismo, no existiera la consulta entre tales para alcanzar una solución viable. Esta independencia durante el desarrollo permitió una mayor aceleración en los tiempos

de entrega total del sistema, cosa que no hubiera sido posible si en todo momento hubiera existido una actitud de trabajo interrelacionado en cada sprint.

Si bien no se contó con un Product Owner que dispusiera de un backlog estructurado, como lo establece la metodología y puede verse en la Figura 8, mediante el cual estipular aquellos requerimientos de mayor prioridad y urgencia a desarrollar durante el siguiente sprint, cada uno de los integrantes dispuso de manera independiente de un log de tareas. A destacar es que estos logs o registros, ejercieron las veces de tablero Kanban, resultando muy útiles durante el desarrollo del proyecto.

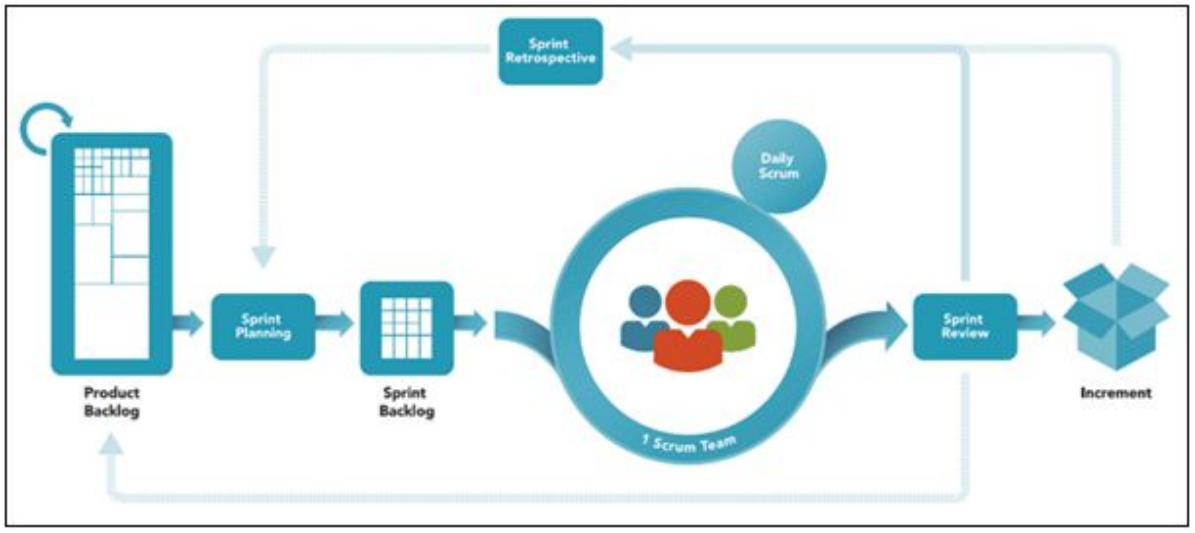

Figura 8. Marco de trabajo bajo metodología Scrum.[6]

A destacar son dos características fundamentales, presentes en la Figura 8, del modelo de Scrum que no se pudieron cumplir de manera estricta, la longitud de tiempo estipulada para el sprint y la reuniones diarias de parado. Si bien los sprints fueron realizados y en consecuencia a los mismos se obtuvo, en la mayoría de las ocasiones, una muestra presentable hacia los clientes, los tiempos de desarrollo no siempre se mantuvieron entre las dos y cuatro semanas estipuladas. Puesto que hubo períodos de tiempo en los cuales se pudo trabajar de manera más intensiva, sea por distintas causales, que en otros.

Las reuniones diarias resultaron en cierta medida imposibles de practicar debido a que no siempre el desarrollo se llevó a cabo en las mismas salas ni al mismo tiempo. Incluso, a pesar de haber coincidido en tiempo y espacio, no existía la posibilidad de disponer de un Scrum Master que las dirigiera. De esta forma, si bien la comunicación fue

[6] Schwaber, K y Sutherland, J. (2017). *The Definitive Guide to Scrum: The Rules of the Game*.

desempeñada de buena manera, no fue mediante reuniones diarias a un horario estipulado, sino más bien de forma informal cuando resultaban necesarias.

Gracias a la implementación adecuada del modelo Scrum es que se alcanzó un producto de calidad que cumple con los requerimientos de los clientes, dentro del tiempo estimado y disminuyendo los esfuerzos que podrían haberse generado debido a modificaciones solicitadas por los clientes.

4.1.2. El diseño

Como se enunció anteriormente, el desarrollo del proyecto se organizó en torno a los postulados de las metodologías ágiles. Como consecuencia, la estructura de cada uno de los componentes del sistema fue evolucionando conforme avanzó el desarrollo del proyecto.

Sin embargo, esto no significa que no existiese diseño desde el inicio. El primer paso fue definir la arquitectura del sistema. Al diseñar la arquitectura, se definieron los módulos/componentes principales y las interfaces entre ellos. De este modo, se definió cada uno de los componentes principales del sistema, sus responsabilidades y la forma en que se debían relacionar con los demás componentes.

A medida que se avanzó iterativamente con el proyecto, se fueron diseñando cada uno de los módulos en detalle. Se definieron diagramas de clases para modelar su estructura y, en los casos en que la complejidad de un proceso lo hizo necesario, diagramas de actividad para modelar el comportamiento.

4.1.3. El proceso de desarrollo del software

El proceso de desarrollo del software se organizó en tareas, características (*features)* y resolución de bugs. Cada una de estas tareas tiene un objetivo particular y el ciclo para su desarrollo consistió en: detectar la necesidad, agregar la tarea a los logs de trabajo, definir una semana / sprint para su ejecución, ejecutarla en una rama separada del código, validarla y unirla a la rama principal (o maestra) del software.

Con las tareas se agruparon trabajos orientados a modelar la estructura del software en general. Algunos ejemplos de las tareas ejecutadas fueron la creación y aplicación de un manejador de errores o el diseño de un módulo particular.

En caso de las características, se agruparon trabajos que tenían por finalidad agregar una nueva funcionalidad. Como ejemplo de este tipo de tareas se puede mencionar: agregar la funcionalidad para analizar textos o para entrenar un modelo.

Finalmente, los bugs respondieron a los errores detectados en el software. La diferencia principal entre un bug y los demás tipos de tareas fue que, por su naturaleza, no fueron esperados; por esta razón muchas veces no existía tiempo disponible para resolverlos. Generalmente, los bugs fueron priorizados para tener una noción de cuán urgente era su resolución y, en función de ello, planificar cuándo trabajar sobre los mismos.

4.1.4. Control de versiones

La naturaleza iterativa de las metodologías ágiles hacen que, con el correr del tiempo, diferentes elementos del proyecto sean modificados. Tanto el software desarrollado, la documentación de diseño, manuales, etc. sufrirán cambios a medida que se avanza con el proyecto. Contar con un sistema de control de versiones es muy importante para poder detectar cuándo y qué cambios se realizaron.

Para la documentación del proyecto, se decidió implementar un control de versiones manual. Es decir, se llevó registros de cada documento particular del sistema. Para cada cambio realizado en un documento se registró: la versión, la fecha del cambio y los cambios / adiciones realizadas. En la figura 9, se puede ver un ejemplo de la estructura para el registro de cambios de un documento.

En cuanto a la administración de versiones del software, se utilizaron dos metodologías: por un lado los cambios realizados al código se controlaron utilizando un software de control de versiones y, por el otro, los distintos hitos dentro del desarrollo se controlaron por medio de documentación.

Versión	Fecha	Change log
1.0	05-01-19	Versión inicial
1.1	23-02-19	- Se agrega rotulado.
1.2	10-03-19	- Se modifica diagrama para incluir al modulo de control del sistema y al task manager. - Se simplica el modulo de usuario, cambiando su nombre y unificando sus funciones en un solo componente.

Figura 9. Registro de cambios en la documentación.

En el caso del software de control de versiones se utilizó Git. El motivo de esta elección se basó principalmente en la experiencia con el uso de la herramienta. Esto se sumó a la disponibilidad de herramientas tales como Github, en este caso para el alojamiento y trabajo colaborativo.

Finalmente, las distribuciones o entregas parciales del software se documentaron por medio de registros internos. Cada nuevo cambio agregado al software se incluyó en un registro. Como puede observarse en la figura 10, cada entrada contiene el identificador asociado con la planificación de la tarea, una breve descripción y su tipo. De acuerdo con la planificación, a intervalos regulares se definieron hitos o *releases* que suponen un entregable parcial del producto. Cada uno de estos hitos quedó reflejado en dicho registro y, para cada distribución, se realizó la subida del software a Pypi.org, un gestor de paquetes de Python. A su vez, cada uno de los cambios introducidos, y registrados en este producto, se volcó al archivo de changelog que acompaña al software.

Task ID	Descripción	Tipo
TF-0045	Eliminar archivos del prototipo del repositorio	Task
TF-0035	Control de la configuración del modulo de procesamiento de palabras.	Feature
TF-0051	Task manager: Implementación del diagrama de clases.	Task
TF-0050	Cambiar atributo de identificación de los modelos.	Task
TF-0044	Prevenir que la edición de nombre haga que se pierda relación con su directorio de modelo.	Feature
TFI-0002	No se puede crear un modelo a pesar de que no hay ninguno con ese nombre.	Bug
TF-0053	Implementación funcional del administrador de tareas	Feature
TFI-0005	No se puede borrar un modelo durante la misma sesión de su creación.	Bug
TF-0054	Obtener el estado de una tarea	Feature
TF-0055	Obtener los listados de tareas activas y completadas del task manager.	Feature
TF-0062	Abortar una tarea desde el task manager.	Feature
TFI-0004	Color no válido para el logger en plataforma Windows	Bug
Release: v 0.1.6		

Figura 10. Registro de cambios en el software.

4.2. Herramientas utilizadas

4.2.1. Librería de procesamiento de lenguaje natural

La decisión inicial de seleccionar a SpaCy como librería de procesamiento de lenguaje natural a utilizar fue, en principio, sugerida por el Info-Lab. Previo al desarrollo de este proyecto, se realizó una prueba de concepto utilizando dicha librería. En ese momento, no hubo análisis de las ventajas o desventajas que pudiera tener el módulo. Sin embargo, al momento de comenzar con el presente proyecto, se compararon algunas de las posibles alternativas. Dicho análisis no sólo reforzó la decisión, sino que permitió comprender porque fue sugerida en un principio.

En primer lugar, la curva de aprendizaje de SpaCy es mucho más eficiente que la de otras librerías similares. Particularmente, este hecho es notorio si se la contrasta

contra NLTK que es una de las más populares. En relación con el proyecto, este es un factor clave. Una librería que requiera de mucho tiempo para obtener resultados puede derivar en que se dedique más tiempo a dominar la tecnología que a resolver el problema. Si se tiene en cuenta que, en un principio, se intentaba evaluar la factibilidad de estas tecnologías cobra mucho más sentido la elección realizada. Cabe destacar que, como se ha mencionado previamente SpaCy, no es menos potente que otras librerías, a la vez que abstrae el usuario de encontrar una configuración óptima.

El segundo factor a tener en cuenta en la elección es el tokenizer. El tokenizer de SpaCy puede ser completamente personalizado. Como se puede ver en la figura 11, el primer elemento en el *pipeline* de SpaCy es el *tokenizer*. Este componente es fundamental para el problema a resolver ya que, como se sabe, el mismo requiere de una flexibilidad adicional para poder entregar al segundo componente, el *tagger*, una entrada lo más correcta posible. Dado que la entrada de datos posee múltiples errores ortográficos, poder modelar el comportamiento del *tokenizer* es clave para que el sistema funcione.

Si bien la performance no es un requerimiento crítico. La arquitectura de esta librería está basada en Cython, factor que permite ofrecer una *API* sencilla en Python con un rendimiento similar al de un programa escrito en C. Por lo antes expuesto, la performance puede ser comparable a la de CoreNLP.

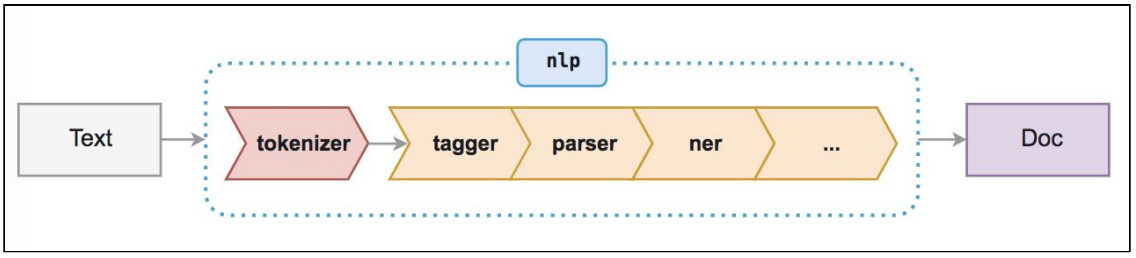

Figura 11. El pipeline de SpaCy.[7]

Finalmente, SpaCy es de código abierto y está en constante mantenimiento. Esto la pone por encima de las demás en términos de adaptabilidad. Del mismo modo, NLTK también es de código abierto aunque no otras tales como CoreNLP. Este factor puede representar una ventaja si el proyecto requiere trabajar directamente sobre la librería en desarrollos futuros o ante la aparición de bugs.

[7] Honnibal, M., y Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing*. Recuperado de: https://spacy.io/usage/processing-pipelines

En conclusión, SpaCy reúne las condiciones necesarias para poder alcanzar el objetivo propuesto. La combinación de facilidad uso con las posibilidades de personalizar el *pipeline* hacen de esta herramienta ideal para el proyecto. Así mismo, no se observa ninguna penalización significativa en términos de performance respecto de otras opciones.

4.2.2. Lenguaje de programación

Desde un principio, uno de los requerimientos principales que surgió por parte de los clientes fue el de llevar a cabo, de ser posible, el desarrollo de la aplicación en el lenguaje de programación interpretado Python. Si bien no era uno de los lenguajes con los que mayor experiencia se contaba, adquirir los conocimientos necesarios no fue un impedimento.

Otro motivo a considerar para la selección de este lenguaje deriva de la librería para el procesamiento de lenguaje natural seleccionada. Dado que la *API* de SpaCy está basada en Python, resulta más sencillo que el resto del desarrollo también sea hecho en Python.

Si bien Python no puede ser considerado tan performante como otros lenguajes tales como Java o C++, su performance no es mala y algunas optimizaciones como Cython (utilizada por SpaCy), permiten tener una performance similar a la de un programa escrito en C al mismo tiempo que se conserva la sintaxis de Python. La sintaxis de Python es sencilla, débilmente tipada y fácil de mantener. En consecuencia, no solo escribir un programa en este lenguaje resulta simple, sino también llevar a cabo aquellas modificaciones que puedan surgir a posteriori. Este último factor no es menor cuando se trata de un proyecto que explora nuevos enfoques y que pudiera requerir cambios.

Finalmente, Python tiene una ventaja determinante relacionado con la naturaleza del proyecto. Este lenguaje es uno de los más elegidos en desarrollos relacionados con la inteligencia artificial. Este hecho hace que exista una comunidad muy grande que lo respalda y puede ser una fuente de consulta. Así mismo, la mayor cantidad de bibliotecas relacionadas con inteligencia artificial están escritas en Python.

En base a todo lo estipulado es que se terminó definiendo a Python como el lenguaje principal de codificación de todo el sistema. Tal es así, que con el fin de mantener las compatibilidades en su máxima expresión, se decidió no sólo desarrollar la librería encargada del procesamiento de lenguaje natural en Python, sino también el sistema web que la consume.

4.2.3. Sistema de gestión de base de datos para la administración de modelos

Uno de los requerimientos del proyecto es permitir administrar distintos modelos de análisis. Al mismo tiempo, es necesario que se administren distintas configuraciones, ejemplos de entrenamiento que consisten de etiquetas, etc.

La naturaleza de los datos requiere de estructuras flexibles. Modelar este tipo de estructuras de datos dentro de un esquema relacional no es en extremo complejo, pero en muchas ocasiones requiere de demasiadas tablas o relaciones complejas e innecesarias. Las denominadas bases de datos *NoSQL* que se basan en estructuras de datos no relacionales son la solución para esto. Existen diferentes modelos que pueden ser orientados a grafos, documentos u objetos.

En algunas ocasiones, las base de datos no relacionales suelen penalizar en rendimiento respecto de las bases de datos relacionales. No obstante, las características del proyecto implican pocas operaciones de escritura / lectura. Del mismo modo, la concurrencia en el acceso de datos está pensada para ser muy baja.

En conclusión, dada la naturaleza de los datos y la frecuencia / naturaleza de las operaciones, se consideró mucho más eficiente que el soporte de datos para el sistema de administración de modelos se haga sobre un sistema de gestión no relacional. El motor elegido fue un motor de bases de datos documentales, en este caso MongoDB.

4.2.4. Utilización de patrones

Un elemento muy importante en el desarrollo de cualquier sistema de software son los patrones de diseño. Existen diferentes categorías de patrones orientados a ser utilizados en distintas etapas del proceso de ingeniería de software.

Los patrones de arquitectura aplican a la estructura del sistema. En el caso de este proyecto, y por las características del mismo, se definió una arquitectura cliente servidor. La motivación para ello se funda en:

- Necesidad de centralizar y restringir a un único nodo las funciones de administración de modelos.
- Los servicios de análisis deben ser provistos por un centro especializado en informática forense, pero pueden ser consumidos desde distintos puntos de la provincia de Buenos Aires o del país.
- La necesidad de ofrecer servicios de análisis a clientes remotos. Estos servicios son un subconjunto de los ofrecidos por el módulo de administración.

A nivel diseño se han aplicado diversos patrones de los denominados *Gang of Four (GoF)* . Estos patrones se han utilizado a distinto nivel y con diferentes objetivos en el desarrollo del sistema.

A nivel estructural se debieron tomar diferentes decisiones. A nivel general se definió tener un único punto de acceso o comunicación entre la presentación y el administrador de modelos. Para ello, se aplicó el patrón *controller*, de modo de establecer un único controlador que reciba todas las solicitudes del sistema. En segundo lugar, se observó la existencia de varios módulos, cada uno de los cuales tiene varias clases. Por este motivo, se decidió aplicar el patrón *Facade*. A través de este patrón, cada módulo del sistema presenta un único punto o interfaz a través del cual los demás módulos del sistema interactúan con él.

Dado que el sistema tiene muchas entidades que necesariamente deben tener una única instancia, se decidió aplicar patrones creacionales. Pasar las referencias de cada entidad a los puntos donde debían ser utilizadas era complejo. Así mismo, dada la utilización de hilos de proceso paralelos, aplicada en algunos procesos, se hace incluso más complejo obtener referencias a dichas entidades. Por los motivos antes mencionados, el patrón *Singleton* resultó una salida eficaz que simplificó enormemente la tarea de desarrollar el sistema.

En cuanto a comportamiento se han utilizado patrones para simplificar ciertos aspectos del desarrollo del sistema. En algunas situaciones es necesario que determinadas entidades conozcan, o escuchen, los cambios que se producen otras. Para esta tarea el patrón *Observer* resultó realmente útil y eficiente. Además, en determinados puntos del sistema se detectaron tareas que eran realmente similares. Con el objeto de tener un código más limpio, entendible y mantenible, se utilizó en la mayoría de estos casos el patrón *Template*. Gracias a este patrón, se crearon plantillas con la lógica compartida dejando en ellas puntos para personalizar.

Para concluir, la aplicación de patrones ha simplificado enormemente el desarrollo. En varios casos, se han transformado operaciones realmente complejas en procedimientos más simples. Además de bajar la complejidad, han posibilitado que se creen funcionalidades que más allá de ser difíciles podrían haber resultado imposibles sin su utilización.

4.2.5. Sistema web

A pesar de que se manejaron como posibilidades de desarrollo tanto al framework Flask como a Django, finalmente se terminó optando por el último. Esto fue debido a no solo las

facilidades de desarrollo que brinda la plataforma, sino también las funcionalidades ya incorporadas que posee.

La meta fundamental de Django es facilitar la creación de sitios web complejos, poniendo énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio "*No te repitas*". A lo largo de todo el desarrollo, estos principios fueron respetados, logrando principalmente una conexión sencilla y correcta con la librería de procesamiento de lenguaje natural (ambos codificados en Python), como a su vez la posibilidad de realizar rápidas presentaciones de iterables ante los clientes. Si bien el principio de *"No te repitas"* en un primera se instancia se respetó para lograr una mayor simpleza en el código escrito, luego se pudo visualizar la potencialidad de este, como en el caso de las urls, ya que Django permite asociar un nombre simple a cada una de ellas evitando así la necesidad de escribir toda la url completa cada vez que se lo requiere. Todo esto implica además la ventaja de evitar repetir expresiones complejas en diversos lugares, conllevando así a además disminuir la aparición de errores por equivocaciones en la escritura. No resulta de menor importancia el hecho de que gracias a las facilidades que otorga Django el sistema sea escalable en función a los futuros requerimientos que podrían aparecer.

Al contar Django con un ORM (mapeador objeto-relacional) incorporado evita las complejidades adyacentes asociadas a la interacción directa con un gestor de base de datos a la vez que facilita la creación de tablas y relaciones mediante la abstracción de las mismas a simples clases de Python.

Dentro de todas las posibilidades que otorga Django, la de seleccionar el gestor de base de datos a utilizar es una de ellas. SQLite se presenta como el gestor por defecto de Django, permitiendo crear bases de datos relacionales e interaccionar con ellas mediante el ORM. Se decidió mantener la configuración predeterminada de la base de datos debido a que no resultan necesarias mayores funcionalidades que las otorgadas por el gestor SQLite.

Django es un framework que toma el patrón de diseño conocido como Modelo–Vista–Template (Figura 12), el cual surge de una redefinición del famosísimo Modelo-Vista-Controlador. Como se denota en el nombre, este patrón le otorga gran importancia a los templates, los cuales terminan sirviendo de base a ni más ni menos que las pantallas con las que el usuario interactúa al utilizar el sistema web. Además, Django ofrece todo un conjunto de facilidades que pueden ser implementadas o no, por los usuarios, al momento de desarrollar el FrontEnd. Como por ejemplo, la posibilidad de heredar templates html o anexarlos a otros, logrando así evitar la repetición de código innecesario y mantenerlo más limpio.

Si bien Django ofrece grandes herramientas que facilitan en gran medida el desarrollo del FrontEnd; sigue siendo necesario, en cierto punto, el uso de otros lenguajes de programación para resolver peculiaridades que sin estos no sería posible. Tal es el caso de JavaScript y la librería jQuery, ambas resultaron indispensables para perfeccionar ciertos detalles del FrontEnd que con Django no es posible.

A su vez, para trabajar con los estilos de las pantallas se implementó el framework Bootstrap desarrollado por Twitter. Esta es una de las otras tantas herramientas utilizadas que permitieron ahorrar tiempo en el desarrollo hacia un sistema afable para con el usuario.

Ninguna de las herramientas utilizadas fueron elegidas a la ligera. Sino más bien, fueron seleccionadas siempre gracias a los beneficios que dispensan y las dificultades que le resuelven al desarrollador. Tal es así el éxito rotundo de algunas de ellas como Bootstrap de Twitter, o Django, framework en base al cual se encuentra desarrollado Instagram.

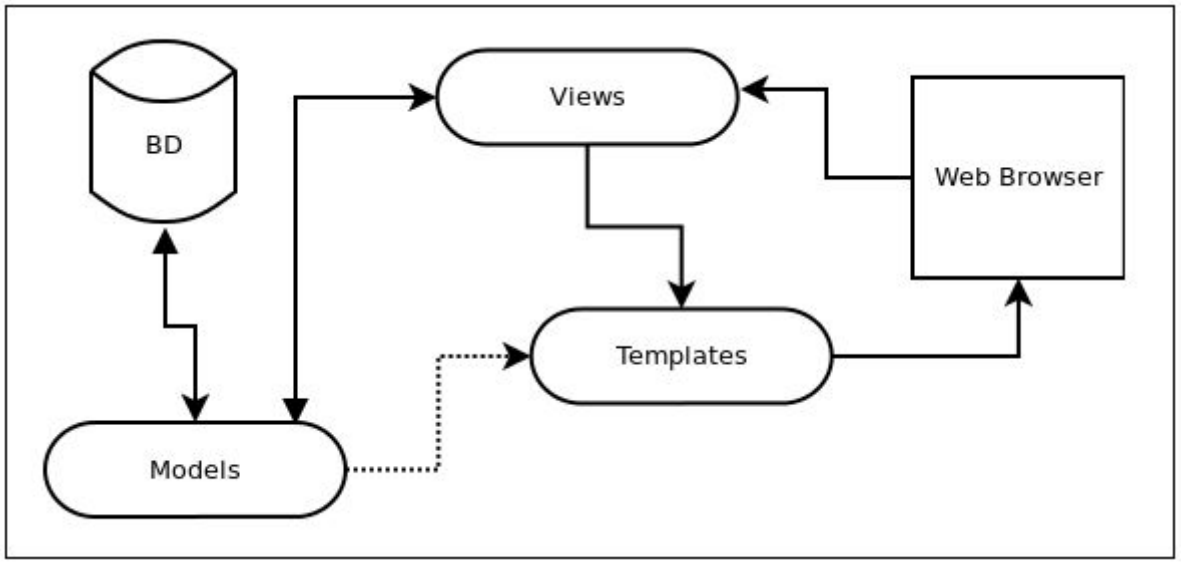

Figura 12. Modelo-Vista-Template[8]

4.2.6. Hosting y despliegues

El alojamiento de la aplicación y el equipo destinado a actuar como servidor están a cargo del cliente, en este caso el Info-Lab. No fue necesario realizar ninguna previsión en

[8] Costa Guzmán, H. (2018-2019) *Patrón MVT: Modelo-Vista-Template*. Recuperado de: https://docs.hektorprofe.net/django/web-personal/patron-mvt-modelo-vista-template/

términos como se iba a alojar la aplicación u obtener un dominio para la misma. La administración y mantenimiento de los servidores también corre por cuenta del cliente.

Si bien a priori es una ventaja, esta disposición implica dos posibles condicionantes: la plataforma (sistema operativo) y el acceso limitado. Por un lado el sistema operativo será Microsoft Windows Server, lo cual implica que la aplicación a desarrollar deberá ser portable o, como mínimo, correr sobre este sistema operativo. Del mismo modo, el acceso para trabajar con los servidores será limitado y requiere ser realizado por personal de la Universidad Fasta. Para paliar este último aspecto, se realizaron pruebas de despliegue a equipos que cuenten con la misma arquitectura y plataforma.

5. Trabajo realizado

5.1. Antecedentes

Tal como se ha mencionado previamente, el presente proyecto nació a partir de un prototipo. Dicho prototipo fue concretado en el Info-Lab en el marco de las prácticas profesionales supervisadas (*PPS*) de uno de los autores del presente en conjunto con un alumno de la carrera Ingeniería en Computación, Gastón Mignone.

El objetivo de la realización de este trabajo fue establecer la factibilidad de aplicar técnicas para el procesamiento de lenguaje natural a la identificación de indicios de narcotráfico en mensajes de Whatsapp. Para este fín, fue sugerida la utilización de la librería SpaCy en conjunto con el lenguaje de programación Python.

Dada la inexperiencia de ambos practicantes con ambas herramientas, la primera fase consistió en adquirir los conocimientos y el dominio necesario para poder aplicarlas. Esta etapa duró aproximadamente dos semanas, al cabo de las cuales, se estaba en condiciones de comenzar a realizar pruebas.

Para la realización de las pruebas, y para poder adaptar la solución a una entrada de datos real, se contó además con los datos volcados de la base de datos de la aplicación Whatsapp de dos teléfonos móviles. Con estos datos, se pudo analizar cualitativamente los datos que debía recibir el prototipo. Los datos presentaban serios problemas para ser interpretados por una computadora. A la complejidad inherente que de por sí presenta el lenguaje natural, se le añaden las omisiones, los errores ortográficos y gramaticales, jergas y demás que hacen de la detección mucho más difícil. Ante esta situación se optó por dos enfoques de trabajo, ambos destinados a tratar con un componente del *pipeline* de SpaCy: el *tokenizer* y el *reconocedor de entidades nombradas (NER)*.

5.1.1. El tokenizer

En cuanto al tokenizer, se buscó la forma de que los errores de escritura no se propaguen a las etapas posteriores del proceso, al menos en los casos con posibilidad de dar positivo. Para resolver este problema se decidió aplicar una aproximación a la lógica difusa. Al aprovechar la posibilidad de agregar excepciones al *tokenizer*, se pueden crear casos para que, al detectar palabras escritas incorrectamente, estas sean asociadas a la raíz de la palabra que realmente se quiere detectar. De este modo, se puede asociar el texto "vndio" o "bende" a la raíz "vender". Esta aplicación del procesamiento de lenguaje natural, conocida como *lematización*, desmostró dos ventajas:

- Al poder darle un lema y una posición sintáctica (*part-of-speech*) a las palabras escritas incorrectamente con relevancia, las operaciones posteriores en el pipeline tienen mayores probabilidades de poder interpretar correctamente el sentido semántico de cada fragmento del texto. Hasta el momento, cada palabra mal escrita introducía ruido a las etapas posteriores ya que la función de cada token se veía alterada. Esta mejora logró que el reconocedor de entidades reciba un input con estructura que, si bien no es la ideal, es más fácil de interpretar.
- En segunda instancia, se detectó que, en base a la raíz de cada término, es decir a través de la lematización, podía detectarse si ese término podría estar relacionado con una temática (en ese momento, el narcotráfico). Este hecho fue el que motivó la idea de "modelos". Cada modelo podría crear una determinada cantidad de excepciones para luego analizar los textos y, finalmente, comparar los resultados contra un conjunto de raíces relevantes para la temática buscada.

En este punto, crear modelos con el *tokenizer* adaptado surgió como una idea factible. La tarea fue, entonces, orientada a generar una colección de sustantivos y verbos para crear excepciones en el *tokenizer*. El trabajo no fue demasiado complejo en cuanto a los sustantivos, pero los verbos y la gran variedad de conjugaciones resultaron ser un obstáculo. Por este motivo se decidió investigar la posibilidad de conjugar automáticamente los verbos y obtener los plurales de los sustantivos. Las librerías disponibles en Python para conjugar verbos en español son pocas y, en muchos casos, arrojan resultados poco precisos. Debido a esto, se optó por desarrollar un módulo capaz de realizar esta tarea. El desarrollo fue complejo dado que el español cuenta con una gran cantidad de reglas, excepciones y casos de verbos irregulares. Con el fín de lograr resultados precisos, se decidió incorporar una estructura que contemple las reglas de

conjugación expuestas por la Real Academia Española. Al mismo tiempo, se implementó la posibilidad de utilizar excepciones para verbos particulares, ya sea por pertenecer a jergas o por no ser cubiertos por las reglas antes mencionadas. El resultado final fue una librería capaz de, a partir del infinitivo de un verbo, obtener de forma precisa sus conjugaciones para los tiempos verbales más utilizados.

El próximo paso fue obtener los términos difusos. Para ello, se generó un procedimiento capaz de deformar los términos aplicando los errores más comunes que cometería una persona que habla español nativo; confundir ciertos caracteres (por ejemplo "v" por "b" y viceversa u omitir caracteres con tilde), omitir un carácter por completo, intercambiar el orden de ciertos caracteres o duplicar alguno de ellos. Para controlar el grado de deformación que sufrían los términos, se utilizó el concepto de la *distancia de Damerau Levenshtein* que establece la distancia entre dos palabras.

Los trabajos realizados sobre el *tokenizer* dejaron como base para el presente proyecto un punto de partida sobre el cual comenzar a trabajar. Si bien la cantidad de elementos reutilizados fue baja, los conceptos fueron valiosos para los trabajos posteriores.

5.1.2. Reconocimiento de entidades nombradas

El objetivo de esta tarea consistió en estudiar las distintas formas en cómo se podrían proveer los ejemplos de entrenamiento y establecer un procedimiento para entrenar la red neuronal encargada de reconocer entidades. Los primeros intentos no fueron exitosos como consecuencia de una cantidad escasa de ejemplos. Los tiempos computacionales, el tiempo requerido para generar ejemplos de entrenamiento y el ruido que generaba el input de datos hicieron que esta tarea fuera en extremo compleja.

Los primeros intentos de entrenar al componente de SpaCy, encargado de reconocer entidades, aportaron múltiples conclusiones. En primer lugar, si el set de datos de entrenamiento solo está enfocado a un subgrupo de entidades, los modelos tienden a desbalancearse y perder la capacidad de detectar otras entidades. En cuanto a la cantidad de ejemplos, se comprobó que al aplicar sets de entrenamiento muy pequeños, en lugar de perfeccionarse, los modelos tienden también a funcionar peor. Finalmente, se detectó que un ejemplo de entrenamiento debía, necesariamente, etiquetar todas las entidades y no solo las que interesaban al modelo. No hacerlo, implica que el modelo tienda a perder la capacidad de reconocer entidades que ya reconocía con anterioridad.

Etiquetar ejemplos de entrenamiento demostró ser una tarea tediosa. Hacerlo sin ningún tipo de herramienta auxiliar puede volverse, incluso, más complejo. Con el fín de

hacer más eficiente esta tarea, el Info-Lab desarrolló una pequeña herramienta auxiliar para etiquetar ejemplos. Con la ayuda de la misma, se pudo crear un set de ejemplos robusto que cuenta con alrededor de 1.500 oraciones con sus respectivas etiquetas. Tras entrenar un modelo con estos ejemplos, se observaron mejoras notorias en los resultados de los análisis de texto realizados.

El mayor problema afrontado durante esta etapa fueron los tiempos computacionales. Con el set de ejemplos antes mencionado, aplicar 50 iteraciones de entrenamiento demandaba en promedio más de 24 horas en un equipo con prestaciones promedio. Este hecho no solo impactó en el estudio del prototipo, sino que hacía que cada una de las pruebas demandará tiempos extremadamente altos.

Las conclusiones obtenidas del desarrollo del prototipo arrojaron información valiosa para los trabajos posteriores. Por un lado, arrojaron datos importantes respecto de cómo se debían confeccionar, seleccionar y aplicar los ejemplos de entrenamiento. Por otro, se construyó un set de ejemplos de entrenamiento que consta de 1.500 ejemplos para entrenar un modelo orientado a la búsqueda de indicios de narcotráfico. Finalmente, el problema de los tiempos computacionales quedó pendiente, pero se estableció que sería uno de los factores a tener en cuenta en los trabajos futuros.

5.1.2. Del prototipo al producto

Si bien el prototipo mostró potencial e información valiosa para una posible solución, aún no se contaba con una idea clara de cómo aplicar esto en un producto final. En este punto, se identificaron dos grandes necesidades a satisfacer, cada una de las cuales aplicaba a un grupo de usuarios con características particulares. Por un lado, el profesional que tiene conocimientos específicos tales como un abogado o un fiscal pero carece de conocimientos avanzados de informática. Por otra parte los peritos informáticos, que no se especializan en algún tipo de delito, pero tienen conocimientos avanzados de informática y diseñan soluciones o extraen información para el grupo anterior. Por este motivo, la solución final se dividió en dos grandes módulos: la librería de NLP y la plataforma web para el análisis y gestión de documentos. Con estos requerimientos como punto de partida, se definió el alcance de cada uno de los componentes macro del sistema.

A través de reuniones con personal del Info-Lab y del Ministerio Público se definieron los requerimientos de la plataforma web. Anecdóticamente, una de las características relevadas dio el nombre con el que se referencia a la plataforma: la necesidad de contar con tres buscadores que aplican técnicas diferentes hizo que la

plataforma sea nombrada como **Cerberus**, en alusión a la bestia mitológica griega de tres cabezas. Finalmente, se definió que este producto debe:

- Gestionar los documentos, en formato docx o de texto plano, de un caso al mismo tiempo que habilita la carga de estos y la realización de distintos tipos de análisis sobre los mismos.
- Permitir el análisis de documentos mediante la utilización de modelos que apliquen técnicas de procesamiento de lenguaje natural.
- Dado que las búsquedas por términos y por expresiones regulares son normales en este ámbito de las investigaciones, el sistema debe contar con esta capacidad.
- Ofrecer un modelo orientado a temáticas de narcotráfico y otro orientado a temáticas de delitos económicos.
- Contar con una interfaz de usuario adaptada a los requerimientos del grupo de usuarios al cual se orienta. La interfaz de usuario debe ser simple e intuitiva.

En contrapartida, el administrador de modelos NLP es una herramienta orientada a los peritos informáticos. Se definió que orientar el esfuerzo a crear un modelo único y robusto sería infructuoso, dada la dinámica de los datos a analizar y el limitado conocimiento específico o *know how* en la materia. Por esta razón, esta librería necesariamente debería ser una herramienta flexible que permita generar, gestionar y mejorar modelos. De este modo, se definió que el producto final debe:

- Ser una herramienta integral para la administración de modelos NLP.
- Permitir crear modelos orientados a la búsqueda de referencias a distintas temáticas.
- Permitir almacenar, aceptar, rechazar y administrar ejemplos de entrenamiento para cada modelo. De este modo, ofrecer herramientas para poder construir una plataforma colaborativa que permita la mejora continua de los modelos.
- Ofrecer los medios para aplicar los datos de entrenamiento sobre los modelos.
- Exponer una interfaz para realizar el análisis de texto plano (ya sea texto o archivos con dicho formato) mediante la utilización de alguno de los modelos disponibles.
- Contar con un método para ajustar los falsos positivos que pudiera detectar un determinado modelo.

5.2. Desarrollo del proyecto

Como se explicó anteriormente, en una primera instancia se definió la arquitectura del sistema. A partir de dicho diseño, se confeccionó el documento de arquitectura (*Anexo 1*)

en el cual se definen los componentes del sistema y las interfaces entre ellos. En dicha arquitectura se pudieron diferenciar dos grandes módulos: el módulo de administración de modelos y la interfaz de usuario o FrontEnd.

Por razones de conveniencia, se planteó separar el desarrollo de ambos componentes. Cada uno de los integrantes del equipo se especializó en uno de los módulos. Esto no significó que se elimine la coordinación interna, sino que estableció un esquema de trabajo donde cada uno de los integrantes se especializó en uno de los componentes del sistema. Esto incluyó la interacción con el cliente y las demostraciones de los módulos. Al mismo tiempo, se coordinaron los servicios que debía proveer el módulo de administración para satisfacer los requerimientos del FrontEnd a través de la definición de una interfaz.

5.2.1. El administrador de modelos (API NLP)

El desarrollo del módulo de administración de modelos comenzó a partir del prototipo anteriormente construido. El primer objetivo de este trabajo fue tomar los elementos y/o conceptos útiles del trabajo anterior para construir a partir de ellos un módulo completamente funcional.

Como se mencionó anteriormente, el proyecto se planificó y se ejecutó de manera iterativa incremental. Cada nueva característica se planificó, diseñó e implementó. Cuando se diseñó la arquitectura del sistema, se definieron dos grandes módulos; el módulo de administración y la interfaz de usuario. Dentro del módulo de administración, se establecieron los submódulos y las interfaces entre ellos. De este modo, las tareas subsiguientes se orientaron a implementar uno por uno cada uno de estos submódulos.

Preparación del repositorio, funcionalidades generales y paquete de Python

La primer tarea de desarrollo realizada fue crear el repositorio e instalar las dependencias necesarias. A continuación fue necesario implementar funcionalidades secundarias que servirían de apoyo durante todo el proceso. Las funcionalidades de linting fueron de gran importancia, ya que al no contar con un IDE especializado como es el caso de Visual Studio o Netbeans, las ayudas al momento de programar son realmente útiles y necesarias.

Si bien parecen tareas triviales, definir la estructura del paquete, crear el repositorio y configurar el entorno del desarrollo son tareas muy importantes. Realizarlas correctamente en la etapa inicial de cualquier proyecto contribuye a crear un entorno idóneo para el trabajo y, al mismo tiempo, constituye los cimientos de cualquier proyecto.

El módulo de administración utiliza varias funcionalidades generales tales como la librería de bases de datos, procedimientos para la lectura / escritura de archivos, etc. Si bien no fue posible establecer la totalidad de estas funcionalidades a priori, una buena parte de ellas fue creada desde el inicio del proyecto. Junto con los demás aspectos mencionados, la identificación e implementación de las funcionalidades comunes es también parte de los procesos fundacionales del proyecto.

Módulo de procesamiento de palabras

Esta fue la primer característica sobre la que se trabajó. Como se mencionó anteriormente, durante el trabajo con el prototipo, se estableció la necesidad de crear una lógica difusa para el tratamiento de palabras clave. Si bien se había hecho una aproximación a una funcionalidad similar durante el desarrollo del prototipo, dicha implementación era bastante mala. El módulo existente consistía de un conjugador, que funcionalmente era muy bueno pero a nivel implementación era muy poco mantenible o escalable, un generador de token difusos y un conversor de sustantivos. La lógica de estos desarrollos se encontraba altamente acoplada al prototipo para generar modelos, al punto que separar la lógica útil de cada funcionalidad fue una tarea en extremo compleja. El diseño final de la estructura de este módulo puede verse en el *Anexo 2*.

En cuanto a las características desarrolladas, el trabajo resultante ofrece la capacidad para conjugar cualquier verbo en español a los tiempos verbales más utilizados comúnmente, obtener el plural de sustantivos y aplicar una serie de transformaciones, orientadas a obtener la lógica difusa de un término. Si bien el sistema ofrece una configuración por defecto, el administrador del mismo puede, en todo momento, crear nuevos temas de configuración para cualquiera de los submódulos. Es decir, puede agregar o quitar excepciones a configuraciones de verbos irregulares, definir reglas y excepciones para la transformación de sustantivos y, finalmente, establecer el orden y tipo de transformaciones que puede aplicar la lógica difusa.

Toda esta lógica fue completamente re factorizada respecto a la anterior presente en el prototipo y, siguiendo los diseños establecidos, modularizada. De este modo, se obtuvo un resultado escalable, mantenible y configurable.

El módulo de administración de modelos

Uno de los requerimientos del sistema es ofrecer la posibilidad de administrar diferentes modelos de análisis orientados a distintas temáticas. Para satisfacer este requerimiento, se estableció que exista un submódulo encargado de administrar a los mismos. Toda tarea

que administre o utilice las funcionalidades de un modelo debe utilizar los servicios de este módulo. En el *Anexo 3* se puede ver el diagrama de clases definido para este módulo.

Si bien, como se afirmó anteriormente, toda operación que se ejecute sobre un modelo debe utilizar este módulo, esta etapa se centró únicamente en la administración de los datos de estos. Es decir, en el manejo de los datos administrativos de cada uno de los ellos. Las funcionalidades implementadas fueron: la posibilidad de agregar un modelo a la base de datos, de editar sus datos y, por último, de eliminarlo.

El hecho de que algunas funcionalidades no se hayan implementado, no significa que no se hayan diseñado y analizado; procesos tales como la creación de nuevos modelos, el análisis de textos o el entrenamiento de los modelos fueron tenidos en cuenta durante este trabajo. Debido a esto los métodos que implementan dichas funcionalidades fueron diseñados e incluídos en el código, sin embargo, sus implementaciones se dejaron vacías.

El controlador del módulo de administrador

Como puede observarse en la arquitectura del sistema, el módulo de administración tiene cuatro componentes principales: el controlador de sistema, el administrador de tareas, el módulo de aplicación y el módulo de administrador. El módulo de administrador tiene un submódulo principal que actúa como *Facade*, es decir, como punto de acceso principal. Todo servicio provisto por este módulo, con excepción de las funcionalidades ofrecidas por el módulo de aplicación, deben pasar por este controlador.

Este controlador administra y coordina todos los procesos. Debido a esto, el diseño de este módulo se realizó con extremo cuidado. Su diseño puede apreciarse en el *Anexo 4*. En el mismo se pueden apreciar cómo se comunica con los demás submódulos y las funcionalidades propias que posee. Al igual que se aclaró antes, las funcionalidades que sí se diseñaron, pero no se implementaron quedaron en el código pero como un método vacío.

Al final de esta etapa del desarrollo, se contaba con la conexión con el módulo de procesamiento de palabras y el administrador de modelos. Todas las funcionalidades, provistas por dichos módulos, se volvieron accesibles desde este punto del sistema.

El proceso de creación de modelos

La creación de modelos fue sin duda uno de los procesos más complicados para modelar. Por este motivo, se confeccionó un diagrama de actividad para modelar el comportamiento del sistema durante este proceso. En el *Anexo 6* puede observarse dicho diagrama que

explica el proceso en detalle. Las operaciones principales llevadas a cabo durante este proceso son:

- Interactuar con el módulo de procesamiento de palabras para crear las conjugaciones, transformaciones y deformaciones necesarias para generar las excepciones al *tokenizer*. Este paso trata de manera separada a los verbos y sustantivos. Mientras que los sustantivos sólo se transforman a plural (y solo si fuese posible), los verbos se conjugan a los tiempos verbales más utilizados. De este modo, el sustantivo "droga" se podría transformar a "drogas", mientras que el verbo "fumar" se podría transformar en cada forma y tiempo de aquellos considerados. A su vez, cada palabra se deforma de acuerdo a la configuración seleccionada del módulo para simular los errores de escritura.
- Varias operaciones a disco para guardar resultados temporales. Dependiendo de las características del modelo a crear, la cantidad de términos generados puede ser numerosa. Dado que estas operaciones pueden sobrecargar la memoria, posibilidad que se acrecienta si suponemos operaciones concurrentes, se tomó la decisión de guardar los resultados intermedios como archivos temporales.
- Interactuar con el administrador de modelos, quien utiliza la librería de SpaCy para cargar en memoria el modelo en por defecto.
- A través del administrador de modelos, actualizar el tokenizer del modelo con las excepciones guardadas en los archivos temporales. Para ello, se cargan las excepciones generadas anteriormente de los archivos temporales y, finalmente, se serializa el modelo actualizado a disco.
- A través del administrador de modelos, guardar los datos descriptivos del modelo en la base de datos.
- Eliminar los archivos temporales generados y finalizar el proceso.

Durante este desarrollo se observó una particularidad que luego se observó también al implementar otras funcionalidades. Algunas operaciones se ejecutan en una especie de pipeline, es decir, la salida de una alimenta o condiciona la ejecución de la siguiente. Pero se espera que se comporten como un *transacción*, es decir o se ejecutan y tienen efecto todas las suboperaciones o no se ejecuta ninguna. Una decisión a tener en cuenta durante la implementación de características con esta particularidad fue establecer el orden de secuencia, mediante el cual, ante un fallo podían deshacerse los cambios realizados hasta ese punto o no afectar el funcionamiento posterior del sistema.

El prototipo desarrollado previamente, fue una guía para este proceso. Sin embargo, muchas de las decisiones tomadas durante su construcción se revisaron y, en la mayor parte de los casos, se mejoraron. Muchos de los procesos implementados en el prototipo no seguían una separación de responsabilidades, de hecho, muchos módulos estaban altamente acoplados, con lo cual reutilizarlos fue imposible.

El proceso de análisis de texto

Este desarrollo fue uno de los más importantes dado que analizar textos es la función principal del sistema. Si bien aún no se contaba con medios para entrenar el *reconocedor de entidades (NER)*, se podía utilizar las capacidades con las que cuenta el modelo estándar en español. Del mismo modo, el proceso de creación de modelos ya era funcional y permitía obtener resultados a partir de los datos obtenidos por el análisis de *part-of-speech (PoS)* y la *lematización*. Como ya se mencionó, al asociar determinadas palabras con una raíz o *lema* durante el proceso de creación se puede, mediante el proceso inverso, obtener resultados al comparar contra dicha raíz.

Para poder comparar el resultado del análisis sintáctico contra lo definido al momento de crear el modelo, el prototipo utilizaba una lógica compleja que incluía técnicas de *meta programming*. Al evaluar estas técnicas, se determinó que no eran la mejor opción para procesar los resultados de un modelo personalizado. Dado que ahora se contaba con una base de datos con información del modelo, se planteó que las reglas de detección de un modelo podrían ser un objeto de configuración. Es decir, si al momento de crear el modelo, se guarda un objeto de configuración que represente todos los elementos a detectar, se puede utilizar dicho objeto para contrastar más tarde los resultados arrojados por la librería de procesamiento de lenguaje natural.

El concepto antes mencionado se materializó como un analizador. El analizador es un componente del sistema que recibe un set de reglas de análisis, representado por el objeto antes mencionado. Con dicha configuración el analizador procesa los resultados arrojados por el análisis sintáctico de SpaCy. Si se supone un modelo preparado para detectar indicios de narcotráfico, el cual detecta “vender” y “droga” como *positivos*, y si se analiza con este modelo el siguiente texto:

“Juan vnde drogas”

En primer lugar, cabe aclarar que vende se ha escrito mal a propósito. El resultado del análisis NLP devolverá tres tokens o términos: “Juan”, “vnde” y “drogas”. De cada token, el análisis sintáctico brinda determinada información; el texto, la oración a la que

corresponde, su *lema*, su función dentro del texto *(PoS)*, etc. Suponiendo que el modelo creado este preparado para asociar vender y droga como indicios, los tokens detectados se asociarían a los *lemas*: "Juan", "vender" y "droga" respectivamente. El analizador recibe entonces como entrada dichos tokens y compara el set de reglas del modelo, también definido al crearlo, contra los *lemas* de cada token. De existir coincidencias se produce lo que se conoce como un *positivo* y este resultado se marca como tal.

Siguiendo el mismo ejemplo, el análisis sintáctico propio del procesamiento del lenguaje natural, asocia cualquier conjugación del verbo al mismo *lema*. Es decir, "vendiendo", "vendí", "vendía", etc. son asociados a la misma raíz. Este hecho hace que la generación de modelos sea muy sencilla, solo se debe especificar, en este caso, que "vender" es uno de los términos buscados. La técnica para crear los modelos explicada anteriormente provee un grado más de flexibilidad. La lógica difusa hace que el *tokenizer* del modelo detecte también los términos mal escritos hasta una distancia deseada. Gracias a ello, términos como "vnde" o "bendia" también son asociados a "vender" durante el análisis.

Pero este es no es único análisis que ofrece el sistema, en paralelo se implementó la detección de entidades nombradas. Si bien esta funcionalidad fue implementada en este punto del desarrollo, no pudo ser validada contra entidades personalizadas ya que los procesos de entrenamiento no existían aún. No obstante, si se continúa con el ejemplo antes mencionado, el resultado del análisis de las entidades nombradas devolvería "Juan" como una entidad del tipo persona, ya que persona se encuentra dentro del conjunto de entidades pre entrenadas del modelo en español.

Por último, el análisis de textos requiere de una herramienta adicional. Detectar palabras clave puede ser útil en algunos ámbitos, mientras que otros no lo es tanto. Por ejemplo, un narcotraficante experimentado no habla explícitamente de los estupefacientes que trafica, en su lugar aparece la figura del *argot*. Lo más normal se de un pseudónimo en las conversaciones de estas personas. Los investigadores del Info-Lab han citado casos en los que, por ejemplo, se decía "baterías" en lugar de la sustancia o droga. Por este motivo, se implementa una funcionalidad que contabiliza la aparición de cada término y agrupa las palabras de acuerdo a su *part-of-speech (PoS)*.

Con todas estas herramientas, el proceso de análisis devuelve tres elementos de utilidad para las búsquedas; el listado de los positivos (utilizando lematización), el listado de las entidades detectadas (utilizando *reconocimiento de entidades nombradas*) y un mapa de frecuencia de todos los términos agrupados por su función sintáctica (utilizando *part-of-speech*).

Logs y manejo de errores

La complejidad de los procesos hizo que ante una única operación sucedan muchas cosas que el administrador del sistema no podía ver a simple vista. Encontrar la fuente de un bug o un error durante el desarrollo se volvió en extremo complejo. Además, se observó que detectar la fuente de errores cuando la aplicación comience a funcionar en producción se volvería imposible sin una herramienta que lleve cuenta de ello.

Por este motivo, se trabajó en un módulo de log. El objetivo de este módulo fue encapsular la lógica para loguear los eventos del sistema. Si bien hasta el momento los logs se imprimen por consola, se estima modificar esto para que se guarden a un archivo cuando la aplicación salga a producción. Encapsular la lógica de los logs, hace que este cambio sea trivial. Además, contar con los logs ha sido de vital importancia para debuggear y encontrar la fuente de los errores detectados durante el desarrollo.

Otro aspecto importante ha sido el manejo de errores. Una operación puede fallar por varios motivos, que pueden ser ajenos o no a la entrada provista por quien consume el servicio. Llevar banderas o propagar la causa de los estos mediante el retorno de los métodos no es una buena práctica, de hecho tiende a crear un código sucio y difícil de mantener o entender. Para lograr un código limpio y, al mismo tiempo, capaz de identificar errores es necesario contar con un módulo que propague y contenga los mismos. Con ese objetivo, se creó un módulo capaz de *lanzar* y luego capturar e interpretar las excepciones ocurridas al momento de informar los resultados.

Aplicando dicho módulo, un error que se produce en una operación básica tal como la lectura de un archivo puede ser despachado como una excepción y, luego, ser capturado por el mismo manejador. Con esta estrategia el método sólo contiene lógica que concierne al objetivo del método, mientras que los errores son manejados en forma externa.

Uno de los mayores desaciertos cometidos durante este proyecto fue implementar este módulo en una etapa muy tardía del desarrollo. Las ventajas que ofrece este módulo no han podido ser aprovechadas en su totalidad, dado que para el momento que se incorporó la mayor parte de las características del sistema estaban implementadas y, por consiguiente, fue necesaria una refactorización importante del código. Es decir, en un principio se implementó un código complejo debido a la ausencia de este componente y más tarde, debido a su implementación, el código se volvió mucho más sencillo. Haberlo implementado desde el inicio hubiese sido un gran acierto.

Estos módulos fueron implementados en conjunto con documentación que los respalda y puede ser de utilidad para los administradores de sistema. En los *Anexos 12 y 13*

se pueden observar los distintos logs y errores en conjunto con los códigos que los identifican y sus detalles particulares.

El paquete de Python

Una vez que las funcionalidades básicas de creación y administración de modelos, en conjunto con la capacidad para analizar textos fueron desarrolladas, se contaba con un software que, si bien estaba incompleto, ya se podía empaquetar y exportar. Fue en este punto de desarrollo del módulo que se comenzó a utilizar un esquema de distribuciones o *releases*. A intervalos regulares, que coinciden con un conjunto de desarrollos planificados para un determinado periodo, se comenzó a realizar distribuciones del paquete.

El gestor de paquetes de Python es más difícil de utilizar que el de otros lenguajes tales como Javascript (*npm*). Para poder crear el archivo de configuración del paquete y los archivos periféricos que definen qué y cómo incluir los archivos en la distribución, se requirió de un tiempo de investigación. Incluir las dependencias en la instalación del paquete fue sencillo pero, dado que el modelo en español de SpaCy se instala de forma separada, fue necesaria la creación de scripts adicionales para poder automatizar la instalación.

Una vez resuelto este problema, se detectó un nuevo contratiempo: la *portabilidad*. El desarrollo del módulo se estaba llevando a cabo en un equipo con sistema operativo MacOS y, como es de sabido por la comunidad de Python, los paquetes de Python suelen tener problemas al instalarse en equipos que usan sistemas operativos de Microsoft. El servidor donde debía instalarse el software corre sobre una plataforma con sistema operativo Windows Server. Por este motivo, era indispensable que el paquete pudiera instalarse y ejecutarse en dicha plataforma. Afortunadamente, el único problema a corregir fue un nombre de variable que usaba un caracter no permitido en dicha plataforma. Sin embargo, se debe recalcar que no haber considerado esto fue un error grave dado que el problema podría haber sido más grande e, incluso, un motivo para desperdiciar gran parte del trabajo realizado.

Una vez finalizada esta tarea, el paquete se encontraba completamente configurado e, incluso, el proceso de *build* y distribución estaba automatizado mediante un script. Gracias a este trabajo, las distribuciones siguientes pasaron a ser un asunto trivial. Cada versión del paquete que se distribuyó ha quedado alojada el sitio *pypi.org* y la instalación consiste en simplemente ejecutar un comando del gestor de paquete *pip*.

El administrador de tareas

Las operaciones de análisis, creación de modelos y, aunque para este momento no estuviese implementada, entrenamiento de un modelo pueden tener una duración de varios minutos. Mantener una conexión HTTP abierta demasiado tiempo no es aconsejable. Para manejar esta situación fue necesario implementar un administrador de tareas.

Una decisión importante a tomar en este punto fue que estrategia tomar para el administrador de tareas. La primer opción era separar las tareas en procesos. Utilizar procesos separados tenía un gran inconveniente, cada modelo cargado demanda de mucha memoria. Como cada proceso se ejecuta con su propio contexto, sería necesario cargar a memoria un modelo en cada proceso. Esto implicaría una demanda excesiva de memoria. La otra opción fue utilizar hilos. La utilización de hilos introduce una complejidad extra al requerir que se maneje la concurrencia pero, al compartir el mismo contexto de ejecución, un mismo modelo cargado en memoria, podría ser utilizado por varios de ellos a la vez.

Debido a esto es que se optó por administrar las tareas mediante la utilización de hilos. De este modo, cada una de las operaciones antes mencionada es controlada por el administrador de tareas. Como puede verse en el *Anexo 7*, cada operación está representada por un tipo de tarea particular. Para resolver los problemas de concurrencia, este módulo incluye lógica para determinar cuáles tareas pueden ser ejecutadas en forma paralela y cuáles no. Además, lleva cuenta del estado de cada tarea con lo cual es capaz de informar su estado en cada momento.

Si bien el desarrollo de este módulo se llevó a cabo exitosamente, su integración quedó pendiente. Si se observa la arquitectura del sistema, este módulo no interactúa directamente con los demás componentes. En su lugar, lo hace a través del controlador de sistema. Por esta razón, su integración final fue demorada hasta que se implementó el controlador de sistema.

El controlador del sistema

Como puede observarse en el *Anexo 8*, el módulo de control del sistema es el punto de entrada al mismo. Cualquier solicitud que se realice será recibida por el controlador, el cual la resolverá utilizando el módulo necesario. El controlador se implementó de modo que para cada solicitud se procese de la siguiente forma:

- Se evalúa si el controlador está listo para procesar una solicitud, es decir, si todos los módulos se han inicializado correctamente.

- Se verifica si existe alguna tarea activa que pueda resultar bloqueante para la operación solicitada. Por ejemplo, si hay una tarea pendiente de creación de modelos en el instante que se intenta modificar la configuración del módulo de procesamiento de palabras, esta operación será bloqueada.
- Dependiendo de la naturaleza de la operación, la misma será ejecutada o insertada en el administrador de tareas. En todos los casos, cada método devuelve un objeto con los resultados de la ejecución de la operación. Para los casos que son puestos en la cola del administrador de tareas, se devuelve el identificador de la tarea para que más tarde se pueda consultar su estado.
- Finalmente, al estar integrado con el manejador de errores, ante cualquier error que se produzca en la ejecución de la tarea, se podrá retornar el indicador de error junto con su descripción en la respuesta.

Tras finalizar el desarrollo de este módulo, todas las operaciones entrantes al sistema pudieron ser resueltas este controlador. Para completar esta implementación y reforzar la idea único punto de acceso, se estableció a este módulo como el elemento exportado por defecto por el paquete de Python.

El módulo de entrenamiento

El objetivo de este componente fue manejar parte de las operaciones relacionadas con el último requerimiento que faltaba por resolver. Su estructura puede verse en el *Anexo 3*. La principal función del mismo es controlar todas las operaciones relacionadas con la administración de ejemplos de entrenamiento y entidades personalizadas. Este fue un desarrollo sencillo dado que solo se enfoco en las funcionalidades de administración.

La primera parte del desarrollo se enfocó en la administración de entidades personalizadas. El *reconocedor de entidades nombradas (NER)* detecta entidades. El modelo en español por defecto cuenta con algunas entidades tales como PER (persona), ORG (Organización) o LOC (Lugar o localización). Para detectar etiquetas especiales tales como “droga” es necesario contar con otras personalizadas por el usuario. Si bien no es mandatorio que las nuevas entidades se administren de forma externa, se definió que era mejor controlarlo desde el sistema.

En segunda instancia, se organizó la administración de ejemplos de entrenamiento. Se estableció que cada ejemplo de entrenamiento tenga una estructura en la cual se tenga la oración y un arreglo con todas las entidades y sus posiciones dentro de la oración. Al administrar las entidades personalizadas, se puede validar que cada ejemplo agregado solo use entidades conocidas.

La obtención de ejemplos de entrenamiento es una tarea ardua y tediosa. Debido a este hecho, se estableció la administración de ejemplos de modo que los usuarios finales (no administradores) puedan proveer de estos al sistema. Con esta situación, se planteó que los ejemplos puedan ser provistos por cualquier usuario, y luego, aprobados o rechazados por el administrador. Toda anotación provista queda guardada en el sistema, con lo cual en todo momento se puede obtener los listados completos de los datos completos de entrenamientos. En todo momento se puede consultar los ejemplos, por estado o históricos de un modelo, y aprobar o desaprobar aquellos que aún no hayan sido aplicados.

El proceso de entrenamiento

El proceso de entrenamiento fue altamente complejo. El mismo generaba mucha incertidumbre dado que durante el trabajo con el prototipo no se había podido resolver completamente. Hasta ese momento, se contaba solo con la información de que la rutina de entrenamiento era demasiado lenta. Sin embargo, desde que se finalizaron los trabajos con el prototipo hasta que se inició el trabajo con el presente proyecto, se añadió a SpaCy la capacidad de procesos en *mini batches*. Esta mejora permite que los ejemplos de entrenamiento sean procesados en lotes, en lugar de uno en uno.

Todo el proceso de entrenamiento se basó en las funcionalidades provistas por el módulo de entrenamiento. De este modo, los ejemplos se acumulan en el sistema y cuando el administrador lo considera propicio, se los aplica al modelo. Cabe destacar que el modelo de entrenamiento que se planteó es un modelo *supervisado*. Es decir, donde cada ejemplo provisto al sistema debe contar con todas las *anotaciones* necesarias. El motivo de esta decisión se fundó en la madurez del sistema. Se consideró que establecer un esquema de entrenamiento semi-supervisado no es aconsejable hasta tener métricas de la aplicación en producción.

La primer parte del proceso implica obtener todos los ejemplos aprobados para un modelo. SpaCy no reconoce el formato de estos tal y como los guarda el módulo de entrenamiento. Para entrenar un modelo es necesario aplicar “anotaciones”, las cuales cuentan con una estructura específica. Para poder utilizar los ejemplos guardados se implementó una transformación que convierte cada ejemplo al formato requerido.

Dado que pudiesen existir nuevas entidades personalizadas, previo a iniciar el entrenamiento se actualiza el componente NER del *pipeline* del modelo. A continuación se evalúan las entidades personalizadas y se agregan a este componente.

El siguiente paso fue establecer la rutina de entrenamiento. Como ya se explicó anteriormente, el entrenamiento o *aprendizaje de máquina* consiste en hacer que una red neuronal ajuste los pesos de sus conexiones sinápticas. Dado que SpaCy encapsula el algoritmo, al trabajar con esta librería solo es necesario establecer la rutina. Dicha rutina consiste en tomar aleatoriamente elementos del conjunto de *anotaciones*. Para ello se genera un *mini batch* a partir de las anotaciones generadas. Esta funcionalidad genera un conjunto de batches. Cada batch contiene una serie de tuplas texto-anotaciones con las cuales se actualiza el modelo de SpaCy.

La actualización del modelo requiere que se le pase el lote de textos, el lote de anotaciones y la tasa de *drop* o descarte. Como se mencionó en el marco teórico, se debe establecer un valor que haga que el entrenamiento converja a una velocidad adecuada. Un entrenamiento demasiado acelerado produce oscilaciones no deseadas, por otro lado entrenar el modelo demasiado lento hace que la convergencia sea muy lenta. Para la selección de este valor se hicieron varias pruebas. En cada una de ellas se evalúo cómo se comportaba un modelo luego de aplicarle un lote de 1.500 anotaciones y 50 iteraciones para la temática de narcotráfico. Con los modelos entrenados se procesaron los mismos textos y se analizaron cualitativamente sus resultados. Finalmente, se ponderó el tiempo de entrenamiento en conjunto con los resultados obtenidos. El valor que mejor ajustó para la tasa de *drop* fue de 0,5. No obstante, este valor es una constante de sistema que puede ser modificada en cualquier momento si se detectan desviaciones o un valor más óptimo.

Luego de ejecutar la rutina de entrenamiento, el modelo actualizado aún se encuentra en memoria. Para actualizar los datos del modelo, se implementa una rutina que elimina los datos antiguos para luego guardar a disco los datos actualizados. Esta es sin duda una operación riesgosa ya que si ocurre un error se pueden perder tanto el modelo entrenado como el anterior. Sin embargo, se considera que es poco probable que falle la operación a disco y, si por algún motivo fallase, muy probablemente todo el sistema estaría en riesgo, no sólo esta operación.

Finalmente, el sistema actualiza el estado de cada uno de los ejemplos utilizados. Para ello se cambia su estado aplicado, para que de este modo no sean utilizados en procesos de entrenamiento posteriores.

El módulo de aplicación

El módulo de aplicación fue un desarrollo sencillo, su diseño puede verse en el *Anexo 9*. El objetivo de este módulo fue únicamente separar ciertas operaciones que pueden ser ejecutadas por un usuario normal, de aquellas que pueden ser ejecutadas únicamente por

el administrador. El único cambio destacable fue el de mover la funcionalidad de análisis de texto del módulo de administración al de aplicación.

Excepciones del analizador

Un detalle que se observó durante las pruebas fue que, al deformar ciertos términos, existe la posibilidad de que uno deformado pueda ser igual a otro correctamente escrito. Por ejemplo, un modelo orientado a narcotráfico que detecta el sustantivo "faso", podría detectar "caso" asociado al anterior. Esto puede puede dar origen a múltiples *falsos positivos*.

Para corregir esta situación, la última funcionalidad agregada al administrador de modelo fue la posibilidad de crear excepciones al análisis del *tokenizer*. Cada excepción se asocia a un modelo particular. Las mismas pueden ser activadas o desactivadas en cualquier momento por el administrador del sistema. Esta funcionalidad posibilita una mejora incremental de los modelos. Su utilización puede ser vista como una forma de refinar la precisión con la que detecta un modelo.

Para implementar esta funcionalidad, se desarrolló un administrador que asocie una o más excepciones a un modelo. Se dispuso, además, que las excepciones se asocien a los modelos cuando estos son recuperados de la base de datos. De este modo, cuando un modelo se carga al sistema, se cargan también sus excepciones asociadas. Para evaluar las excepciones, se agregó un paso durante la rutina del analizador: al momento de detectar un término positivo, válida que no exista una excepción que lo inhiba.

Creación de un modelo para una temática específica

Para la creación de un modelo de prueba, se trabajó en conjunto con el personal del Info-Lab. Para ello se generó una semilla compuesta por una serie de palabras, reconocidas en la jerga penal, relacionadas con el tráfico, consumo y diferentes formas de nombrar a las drogas. Dicho conjunto se utilizó para generar las excepciones del *tokenizer*.

En segunda instancia, se ingresaron al sistema aproximadamente 1.500 ejemplos de entrenamiento. Los mismos fueron etiquetados por Gastón Mignone, alumno de la carrera Ingeniería en Computación de la Universidad Nacional de Mar del Plata. Este trabajo fue realizado durante la etapa de prototipo del proyecto y, finalmente, fue cedido al Info-Lab.

Con dicho modelo creado, se realizaron algunas pruebas sobre textos, relacionados con drogas, extraídos de diarios online o informes sobre drogas. A partir de dichas pruebas, se generaron las excepciones al analizador necesarios para reducir la cantidad de falsos positivos detectados.

5.2.2. Cerberus

Las libertades concedidas por los clientes al momento de diseñar Cerberus fueron claves, esto es debido a que no hubo restricciones acerca de cómo debía verse el sistema, sino más bien las funcionalidades que debía ofrecer. En este punto fue que, al no ser ninguno de los integrantes diseñadores ni contar con la ayuda de alguien especializado en ese ámbito, la creatividad de los mismo resultó clave.

Primeros pasos

Mediante la descripción de los requisitos y funcionalidades necesarias para que los usuarios pudieran desenvolverse de manera simple y efectiva con el sistema fue que se comenzó a realizar bosquejos de ciertas pantallas. Resultó clave la retroalimentación obtenida gracias a uno de los clientes. Como los usuarios finales son investigadores judiciales, sus consejos y aportes respecto a que sería mejor en la interfaz fueron muy beneficiosos. Tal es así que, siempre que se pudo, se llevaron a cabo reuniones con la finalidad de presentar avances y permitir la utilización del sistema parcial que se tenía.

Antes de comenzar a codificar la interfaz de usuario se deliberó acerca de cómo sería la estructura de la base de datos que albergaría la información necesaria y generada por el sistema. De esta forma se generó un modelo de entidad-relación (*Anexo 11*) que en esa primera instancia sólo incluía tres entidades muy importantes del sistema “Caso”, “Documento“ y “Usuario”. Estas fueron las bases para empezar a diseñar y codificar prototipos iterativos hasta llegar al producto final que se tiene actualmente.

Gracias a Django muchas funcionalidades fueron muy sencillas de desarrollar. El manejo de usuarios y administración de contraseñas fue uno de estos casos, el framework posee todo un sistema de alta performance que se encarga de esto, liberando así a los desarrolladores de la necesidad de implementar una solución distinta. Sin embargo, una de las mayores complejidades que apareció fue la implementación de asincronía a lo largo del FrontEnd. Esto sucedió debido a que Django funciona sincrónicamente y para lograr la asincronía era necesario utilizar frameworks o librerías de terceros que no siempre resultaban sencillos o 100% compatibles con el mismo. A causa de esto es que se decidió aprovechar las facilidades que brinda Django para trabajar de manera sincrónica.

Luego de tomar estas decisiones y crear una primera instancia de la base de datos de manera muy sencilla gracias al ORM que implementa Django, fue que comenzó la codificación de las primeras pantallas de la interfaz de usuario.

Login

Una de las cualidades que posee Django es la de incluir plantillas pre-construidas para aquellos recursos que aparecen comúnmente en todos los sistemas web. Este es el caso de la pantalla de Login o Inicio de sesión.

Fue necesario únicamente definir el aspecto de la misma sin necesidad de inmiscuirse en la codificación del funcionamiento o del manejo de usuarios.

Administración de casos y documentos

Como se mencionó, en primera instancia el sistema surgió como un pequeño prototipo capaz de crear casos, editarlos y cargar documentos a los mismos. Los casos serían el punto donde volcar toda la información asociada a una investigación específica de un acto delictivo, para luego poder procesarla y generar conclusiones en base a lo obtenido. Esta información en un primer momento sólamente se vería reflejada por la descripción con la que pudiera contar el caso en sí, como también las de los documentos que se hubieran cargado al sistema. Esto cambiaría notablemente conforme avanzará la codificación del sistema.

Si bien se poseían ciertos conocimientos de Django antes de empezar a desarrollar el sistema web, fue necesario mucha más especialización e inmersión en el framework conforme las dificultades iban surgiendo. Igualmente, gracias a la gran cantidad de documentación oficial y extraoficial que existe, esto no implicó más que dedicación y tiempo a tal fin de solucionar las problemáticas. Algunas serían por ejemplo la generación de expresiones regulares específicas para búsquedas determinadas dentro de los archivos, otras el hecho de la carga correcta de archivos al sistema utilizando el FileField de Django debido a las ventajas que presenta a nivel de almacenamiento, la descarga de los mismos, el manejo de las migraciones de la base de datos que requiere el framework, etc.

A este punto, una vez estuvo disponible la posibilidad de crear casos, surgieron las dificultades mencionadas anteriormente respecto a la carga de documentos. Si bien se poseía un modelo en el cual almacenar los documentos adjuntos a un caso, era necesario implementar la parte lógica en la cual se procesaba a estos mismos conforme se los cargaba y finalmente se los guardaba en el servidor.

Muchas decisiones debieron tomarse en esta instancia, acerca de cómo almacenar el texto para que fuera más sencillo su procesamiento posterior con tal de obtener la mayor cantidad de información posible, los tipos de documentos a aceptar, en dónde se guardarían los archivos, como se comportarían los mismos respecto a duplicados, etc.

En base a investigación y deliberaciones fue que se decidió permitir únicamente documentos en formato .docx y .txt, ya que otros formatos como .pdf y .doc resultaban muy difíciles de procesar de manera correcta a tal fin de obtener el texto de manera plana e idéntica. A su vez, en la base de datos sólo se almacenaría el texto plano de los archivos, sin ningún tipo de formato de forma tal de ocupar menos espacio. Los documentos originales se almacenan en el servidor en una carpeta local con el nombre conformado por el hash SHA1 del mismo, concatenado con la fecha en la que se lo subió, esto con el fin de no permitir que exista conflictos por nombre.

Un detalle a destacar es que todos los documentos guardados poseen tanto un hash SHA1 como un hash MD5. Si bien el último no tiene ninguna utilidad actualmente, se lo contempló con la previsión de que el sistema, a futuro, no permita la carga de archivos duplicados dentro y entre los casos. De esta forma, mediante los códigos hash se podría determinar si un documento ya existe en el sistema y sí es así no cargarlo nuevamente, sino compartir el existente. Por todo esto también es que se agregó la posibilidad de que cada documentos tenga un único propietario pero muchos usuarios, siendo que por como funciona actualmente el sistema siempre se corresponderá el propietario del mismo con el usuario, y este será único.

Otro de los dilemas más importantes que surgieron a esta altura del desarrollo fue el deseo de lograr que los documentos puedan ser cargados de manera asincrónica. Sin embargo, la falta de facilidades que se percibió por parte de Django para permitir hacerlo y el hecho de que en reiteradas ocasiones los clientes indicarán que no era para nada necesario que el sistema se comportara de manera asincrónica, conllevo a que se abandonara este deseo y se rediseñaran las funcionalidades de tal manera que el sincronismo de Django no fuera percibido por los usuarios.

Finalmente, cuando se logró que la subida de documentos fuera todo un éxito es que se pudo pasar a desarrollar las partes más importantes del sistema.

Pantalla de admin e historial

Una de las partes más poderosas de Django es la interfaz de administrador automática que posee. Es decir, el administrador de un sistema, mediante el acceso a esta interfaz, ya se encuentra en poder de utilizar todas las herramientas necesarias para administrarlo.

Esta pantalla sólo es viable de ser accedida si se es el administrador del mismo, cosa que se determina mediante la posesión, o no, de privilegios de super-usuario.

A pesar de todos los beneficios que incluye este sitio, no trae incorporada la posibilidad de mantener un historial acerca de todos los ABS que puedan suceder en la

base de datos. En consecuencia y gracias a los consejos aportados por el tutor, se encontró la aplicación "django-simple-history" que permite mantener un registro de todos aquellos modelos que se le indiquen.

Si bien parecía que se tenía todo resuelto para poder llevar un tracking de los ABM de la base de datos del sistema, apareció el inconveniente de que la aplicación antes mencionada sólo permitía mantener el rastro de las altas y modificaciones pero no de las bajas. Esto era debido a que la tupla en sí dejaba de existir en el sistema y por lo tanto se perdía todo rastro histórico asociado a la misma. Para solucionar este dilema fue que se decidió que cada uno de las tuplas de las tablas ya no serían eliminadas de la base de datos cuando esto fuera lo deseado. Sino, más bien, se les incluiría un nuevo campo booleano "eliminado" a las tablas para que indicara el estado de las tuplas.

Gracias a esta nueva modificación fue que se logró que todas las tuplas tuvieran su correcto registro histórico, incluso si esta tupla decidía eliminarse. Ya que este eliminado no sería más físico, sino lógico. Provocando así que la información a mostrar en el FrontEnd siempre deba ser filtrada por "eliminado" para determinar si aún se encuentra en vigencia.

Es de destacar que siempre que se hable acerca de la eliminación del sistema de alguna tupla, se estará refiriendo a la eliminación lógica anteriormente mencionada.

Los buscadores

Las denominadas tres cabezas de Cerberus surgen como el atractivo mayor del sistema y a su vez el facilitador más importante de información específica. Desde un primer momento estos tres buscadores fueron estipulados como la funcionalidad primordial con la debía contar la aplicación. Si bien los nombres no indican en detalle cómo se comporta cada uno de ellos, esto es así ya que los usuarios de los mismos no necesitan saberlo gracias a la simpleza con la que se pueden utilizar.

Cada caso tiene habilitadas todas las búsquedas para poder ser realizadas siempre y cuando continúe el mismo en curso. Estas, funcionan sobre todos los archivos, actuales, que se encuentran adjuntos al mismo y que conforman gran parte o toda la información vinculada al acto delictivo.

Para realizar esto fue necesario un correcto almacenamiento y procesamiento de los archivos subidos. Cada uno de los buscadores procesa la información almacenada de distinta forma:

- El buscador general hace una búsqueda simple dentro de cada uno de los archivos para determinar la existencia de expresiones introducidas por el usuario al sistema.

- El buscador guiado se basa en expresiones regulares para poder buscar y determinar la existencia de ciertos patrones específicos en los documentos.
- El buscador inteligente, el buscador más importante, consume el tokenizer y el NER generado por la librería de procesamiento de lenguaje natural para poder llevar a cabo búsquedas por entidades específicas, de manera muy simple, a lo largo de toda la documentación. De esta forma se permite a los usuarios acceder a información que resulte de gran importancia para ellos dentro de un caso determinado.

Cabe destacar que para que Cerberus brinde respuestas precisas, es necesario entrenar a la red neuronal, tarea encargada para el administrador del mismo. Esto resulta relativamente sencillo debido a que se puede obtener información errónea, para el entrenamiento, desde el mismo buscador inteligente. El mismo, a su vez, permite eliminar o modificar aquellas entidades que para el usuario parecen no coincidir con los resultados deseados. Esto implica que si bien el administrador es el encargado de entrenar la red neuronal para que los modelos funcionen de manera más precisa, los usuarios deben comprometerse a purgar aquellas búsquedas que conscientemente noten se encuentran del todo correctas, para así otorgar información correcta con la cual el administrador pueda realizar su trabajo.

Si bien al comienzo sólo se utilizaba el modelos estándar de SpaCy que no permitía satisfacer los requerimientos de los clientes, posteriormente cuando se finalizó la librería de procesamiento natural y fue viable consumir directamente a través de esta, comenzó a ser posible ubicar apariciones relacionadas con la droga y con delitos económicos, dos de los modelos indispensables para los usuarios del sistema. Al crear un caso es necesario que se indique con cuál de estos modelos (económico/drogas) es que se desea trabajar, para así la búsqueda inteligente saber cómo funcionar en base a esto.

Como se explicó anteriormente, la creación de modelos depende del administrador, es así entonces que la aplicación está construida de tal forma que permita la escalabilidad hacía más de los dos modelos que fueron estipulados de manera inicial como requerimiento.

Notas

En una de las reuniones pautadas con los clientes se presentó la necesidad de que se pudiera, tanto a los casos como a los documentos, anexar notas. Estas notas son claves debido a la potencialidad que agregan al permitir introducir información extra tanto a los casos como a los documentos.

Esta funcionalidad fue agregada en un sprint nuevo, involucrando la necesidad de realizar el diseño correspondiente que aparece visible en el MER (*Anexo 11*).

A este punto no aparecieron grandes dificultades a nivel de BackEnd, sólo se puede mencionar ciertos problemas, que fueron solucionados rápidamente, respecto a cómo llevar adelante la muestra y creación de notas de tal forma que fuera intuitivo y fácil de realizar por parte de los usuarios.

Resultados e informes

Aunque apareciera a colación en unas de las reuniones en las cuáles ya se encontraba avanzado el desarrollo del sistema web, se había estipulado desde el primer momento la necesidad futura de un medio mediante el cual extraer aquella información de interés que se obtuviera mediante el sistema. Provocando así la nula necesidad de modificar código ya existente.

La presentación de informes resulta crucial en todo el ámbito jurídico, ya que la información se transmite y almacena en su gran mayoría de forma impresa.

Si bien, a esta altura del proyecto no se veía como una tarea de grandes complejidades, fue necesario bastante tiempo para poder obtener un diseño y una implementación coherente a lo esperado.

Una de las problemáticas con las que se encontró fue la de lograr que los resultados de una búsqueda determinada puedan ser guardados sólo si el usuario así lo indicaba. Esto implica que toda la información generada por el BackEnd y transmitida hacia el FrontEnd para mostrarla, debía ser reenviada nuevamente hacia el BackEnd si es que el usuario indicaba el deseo de guardarla.

Luego de mucha deliberación en la que se debatieron varias alternativa, se decidió que lo mejor sería enviar desde el BackEnd hacia el FrontEnd un json con todas las tuplas de resultados. Implicando que esta misma información se encuentre duplicada en el FrontEnd pero que el json sea invisible debido a que está ubicado dentro de un campo oculto. Ante la necesidad de guardar los resultados en la base de datos, el mismo. es enviado hacia el BackEnd para ser procesado.

Al momento de procesar el json, se compara cada una de las tuplas del resultado con un array que indica si alguna de estas fue destacada por parte del usuario antes de solicitar el guardado. El concepto bajo el cual destacar una tupla o no queda completamente bajo criterio de los usuarios, ya que no implica nada más que el hecho de resaltar una tupla sobre otras.

Toda esta información procesada se almacena en la base de datos en tablas diseñadas específicamente dependiendo de si el resultado guardado fue obtenido mediante una búsqueda general, guiada o inteligente.

Las pantallas de resultados son las que recuperan toda esta información para mostrarsela a los usuarios y permitirle generar en base a ella tantos informes como desee. Estos pueden mostrar toda la información como solo la destacada.

Para generar los informes fue necesario inmiscuirse en la librería python-docx. La misma permite crear documentos de word en formato .docx desde cero. Por suerte es muy utilizada y existe mucha información acerca de cómo utilizarla. Estos informes muestran en formas resumidas quien es el usuario que lo creó, a que hora, la búsqueda que se llevó a cabo, el parámetro en base al cuál se realizó la búsqueda y el caso específico.

Lograr que el texto ingresado al documento quedase en un formato agradable para los usuarios fue todo un trabajo aparte. Si bien python-docx se encuentra documentada, no es sencillo descifrar cómo es que debe hacerse para que el texto del documento aparezca bajo el formato deseado. Debido a esto es que tomó más tiempo de lo esperado poder generar un informe sencillo y fácil de leer que fuera útil para presentar información importante.

Protección y seguridad

Generalmente, las investigaciones no son llevadas a cabo por una sola persona. Esto se traslada a los casos del sistema, más de una persona debería poder acceder y aportar a la investigación asociada a un caso que se está realizando.

En base a esto es que Cerberus permite la colaboración de varios usuarios en un mismo caso. Sin embargo, que más de un usuario pueda colaborar en el mismo caso no significa que todos tengan los mismos privilegios sobre el mismo.

Cada caso posee un único propietario y muchos usuarios. Al momento de creación obviamente ambos coinciden. No obstante, para que más de un usuario pueda acceder al mismo caso, este debe ser compartido.

Solamente el propietario de un caso es el que tiene el privilegio de poder compartirlo con otros, es decir que un usuario común nunca podrá compartir un caso con cualquier otro usuario a no ser que adquiera los derechos de propietario del mismo.

Al momento de compartir, el propietario puede elegir si cede sus privilegios sobre el mismo o no. Esto afectará no sólo a su posibilidad de seguir siendo el “dueño” del caso y decidir quién puede accederlo y quién no. Cuando no posee privilegios sobre el mismo, un usuario no puede eliminar el caso del sistema, ni finalizarlo o eliminar documentos

asociados al mismo. Es decir que se pierde de muchos beneficios que otorga el hecho de ser propietario del mismo.

Es de destacar el hecho de que a pesar que un usuario no pueda eliminar los documentos asociados a un caso debido a que no posee los derechos sobre el mismo, no quita la posibilidad de que le solicite al propietario su eliminación bajo un justificativo de por qué lo hace. Queda a criterio del propietario considerar la justificación como suficiente o no para borrarlo.

Reestructuración

Ya sobre el final del desarrollo del sistema, se tomó una decisión clave para modificar la perspectiva del usuario sobre los casos.

Si bien este implica un cambio radical en cómo el usuario accede y opera con los casos, gracias a como fue implementado todo el sistema no fue necesario realizar grandes cambios.

La idea básica de poseer una única pantalla en la que se mostrarán todos los casos de un usuario, fue modificada y presentada como dos nuevas pantallas. Cada una de ellas mostraría casos pero, en distintos estados.

- Casos en curso: la primera de las dos pantallas y la que más uso tiene replica la funcionalidad que tenía antes la pantalla general de casos, permitiendo acceder a toda la información asociada a los casos y realizar todas las tareas que se realizaban antes de igual manera. Una modificación notable es que ahora los casos deben ser finalizados para desaparecer de esta pantalla (antes se eliminaban directamente del sistema). La finalización puede ser tanto correcta como incorrecta, permitiendo así que se determine si un caso fue resuelto como si no lo fue pero ya el mismo no va a continuar en curso. Los privilegios sobre los casos afectan al hecho de si el usuario podrá finalizar al mismo o no. Si no se posee los privilegios pero se finaliza un caso igualmente, lo único que sucederá es que el usuario dejará de formar parte de los usuarios del caso.
- Casos finalizados: todos los casos que hayan sido finalizados podrán ser visualizados en esta pantalla. Se diferencia a los correctos de los incorrectos mediante una coloración en la sección izquierda de los mismos. Si bien se puede navegar entre los casos y dentro de toda su información asociada como desde la pantalla anterior, aquí no se podrá realizar nuevas búsquedas ni eliminar información ya creada, a excepción de las notas. En este punto, cuando se decide que ya no es necesario

visualizar más un caso, puede ser eliminado (sólo por el propietario) , implicando que el mismo dejará de aparecer en el sistema.

Estas modificaciones permiten que los usuarios adquieran una apreciación mejor de los estados de los casos y de su administración.

5.3. Métricas y retroalimentación

5.3.1. Métricas

El análisis de las métricas del proyecto se hará en función de dos dimensiones: relativas al producto y relativas a la gestión del proyecto.

Métricas relativas al producto

El producto, más allá de satisfacer los requerimientos, muestra métricas interesantes que son importantes de destacar. Si se compara el método actual para analizar evidencia con la capacidad del producto desarrollado, se puede observar una diferencia realmente importante en cuanto a eficiencia.

Las estadísticas indican que una persona, en condiciones óptimas de concentración y fatiga, lee en promedio unas 200 palabras / minuto. Existen ciertos cursos o entrenamientos que aseguran que un humano puede leer a un ritmo de 1000 palabras / minuto. Si bien se considera que esto último es difícilmente practicable o, como mínimo, un ritmo que no es posible de mantener en el tiempo, igualmente se hicieron comparaciones. De este modo, no solo se comparó al sistema contra un humano promedio, sino que se lo comparó contra condiciones excepcionales. En la figura 13, se puede observar los tiempos, en segundos, que le tomaría a un humano leer un texto de acuerdo a su longitud medida en cantidad de palabras. La serie correspondiente a “humano A” corresponde a una velocidad de 200 palabras / minuto, mientras que “humano B” corresponde a 1000 palabras / minuto. Cabe destacar que se está descartando el factor de la fatiga.

Las pruebas de rendimiento del sistema fueron realizadas en un equipo de alta gama que, de todas maneras, no alcanza las prestaciones de los servidores donde será alojado el sistema. El equipo utilizado cuenta con las siguientes prestaciones:

- Procesador: 2.8 GHz Intel Core i7.
- RAM: 16 GB 2133 MHz LPDDR3.
- Almacenamiento flash, disco de estado sólido.

- Sistema operativo: macOS High Sierra v10.13.6.

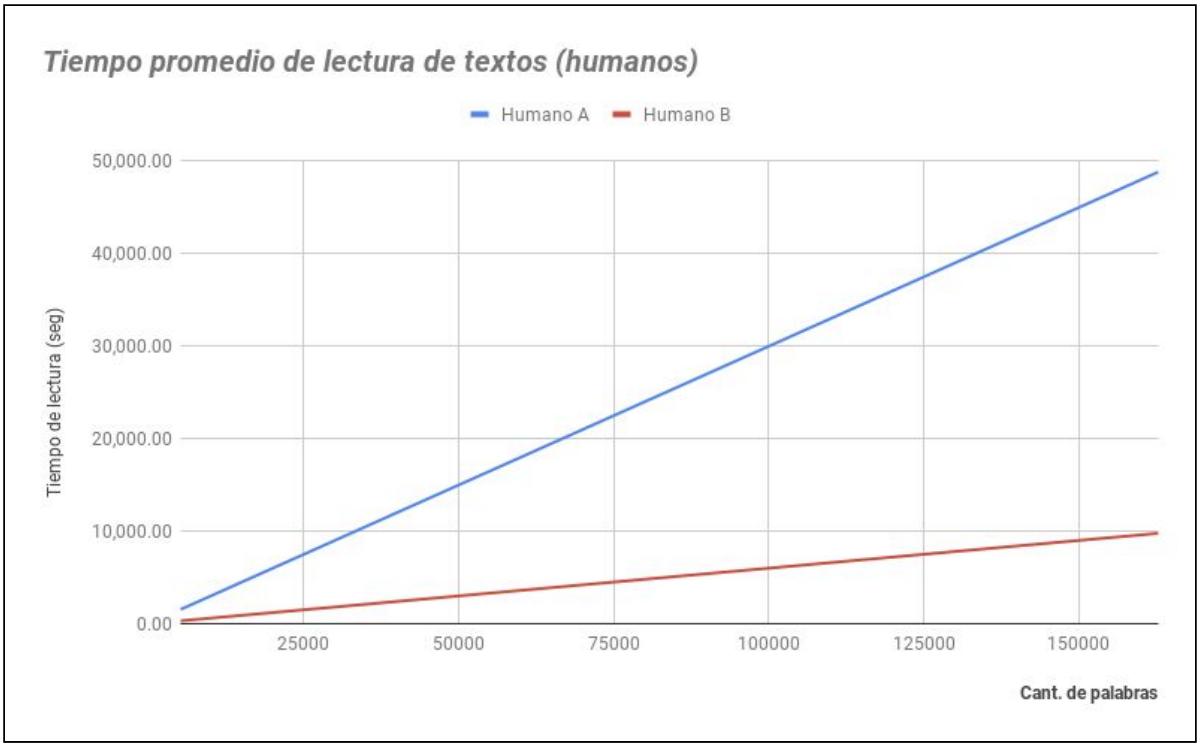

Figura 13. Tiempo promedio de lectura requerido en función de la longitud del texto.

En la figura 14, se puede observar el resultado de las pruebas realizadas. Dado que el sistema requiere una carga inicial cuando se utiliza por primera vez el modelo, se pueden distinguir dos tiempos: cuando se realiza la carga y cuando no. Para obtener los tiempos se ha tomado los timestamp del administrador de modelos. Con el objeto de evitar decimales innecesarios, se ha pasado al segundo siguiente cualquier exceso en milisegundos respecto del tiempo en segundos.

De la comparación entre ambas gráficas, el primer hecho que resalta es que la diferencia entre la capacidad de un humano y el sistema ronda los tres órdenes de magnitud. Es decir, el sistema NLP puede analizar un texto 1.000 veces más rápido que un humano. Ya que la diferencia es demasiado grande como para que su comparación pueda apreciarse en una misma gráfica, se optó por graficar la proporción entre el tiempo demorado por el sistema y el demorado por un humano. En la figura 15, se puede apreciar esta comparación. Como puede observarse la proporción tiende a disminuir conforme más extenso es el texto.

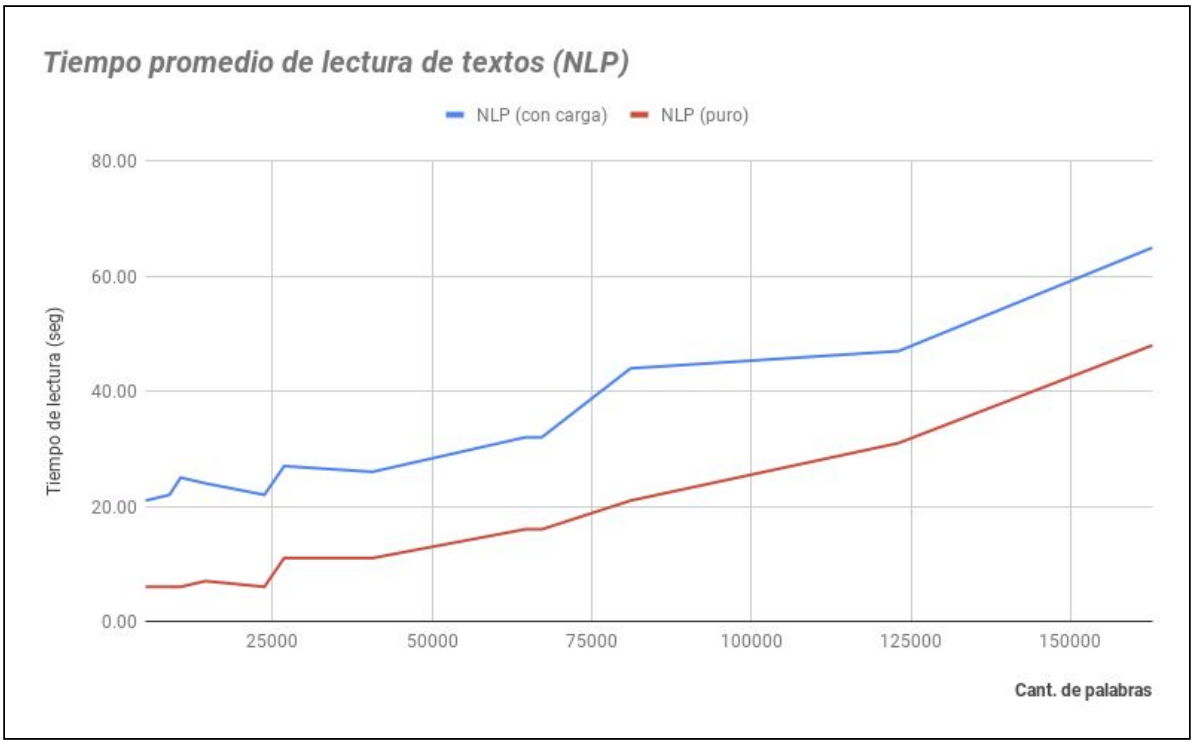

Figura 14. Tiempos de procesamiento registrados por el sistema en función de su longitud.

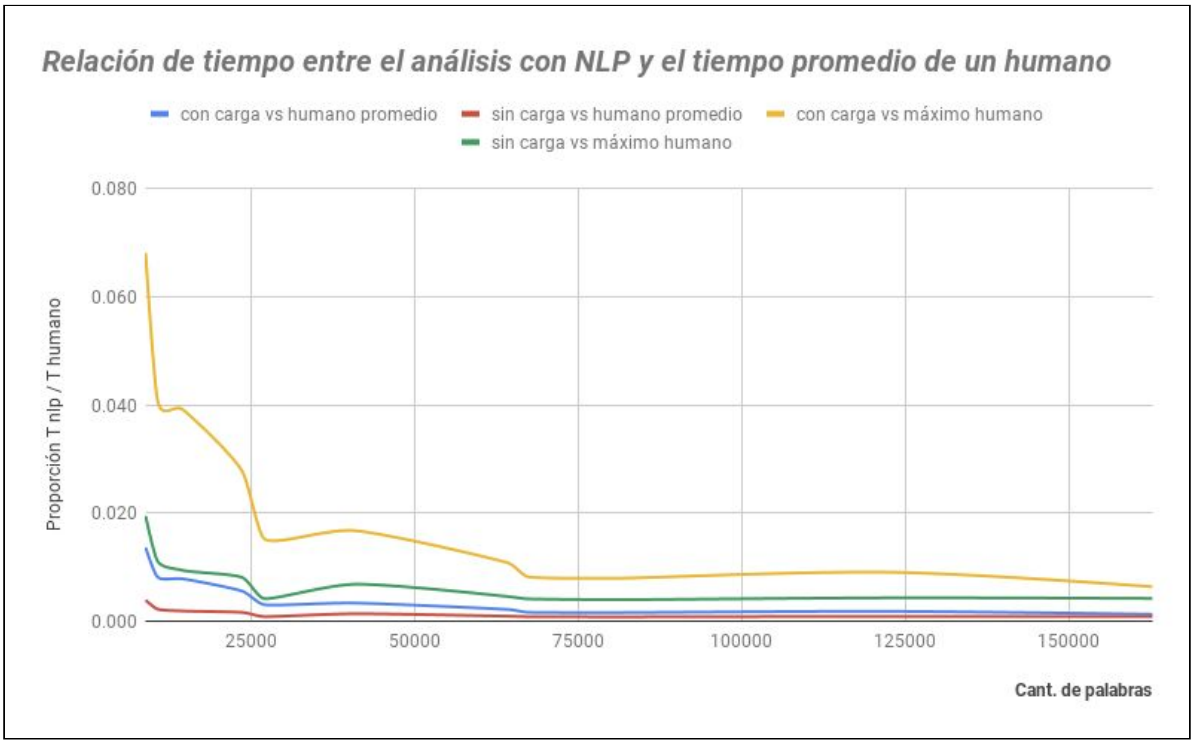

Figura 15. Proporción de tiempo entre el tiempo del sistema respecto del de un humano.

Cuantitativamente el sistema es muy superior a un humano en términos de eficiencia. Además, es destacable que la fatiga y la distracción son factores que condicionan enormemente los resultados. Pero analizar los resultados únicamente en

función de la eficiencia sería un error, para completar el análisis es indispensable que se analice la eficacia.

Como se ha mencionado anteriormente, la intención del sistema no es reemplazar el trabajo del investigador, sino servir como una herramienta que le permita enfocar su atención en determinadas secciones o extractos del texto. Crear un sistema que pueda ser homologado a nivel judicial, es decir, certificar que tiene un determinado porcentaje de de precisión escapa a los objetivos del proyecto. Por este motivo, analizar los resultados del sistema en términos de cuántos términos detectó sobre un documento con una determinada cantidad de ellos carece de sentido. En su lugar, se utilizó un enfoque distinto para verificar su funcionamiento en términos de eficacia. Para ello se utilizó un modelo de prueba entrenado a partir de aproximadamente 1.500 ejemplos de entrenamiento y con un conjunto acotado de excepciones para el analizador. Del mismo modo, se seleccionó un conjunto de textos orientados a temáticas de consumo o narcotráfico. A continuación, se analizaron los mismo utilizando el modelo antes descrito y se evaluaron:

- La cantidad de entidades asociadas a las drogas (etiqueta DRUG) detectadas y cuántas de ellas fueron erróneamente asociadas.
- La cantidad de tokens detectados por el *tokenizer* del modelo y cuántos de ellos no eran correctos.

En la figura 16, se pueden ver los resultados obtenidos para el reconocedor de entidades. Cada barra tiene una componente verde y otra roja: los positivos y los falsos positivos respectivamente. Como puede observarse, para el archivo *test_file_G.txt* el porcentaje de falsos positivos es elevado. Dicho archivo tiene particularidades tanto en la forma en que está escrito como en el contenido. Dicho resultado pone en manifiesto dos cuestiones: dependiendo de la estructura del texto podría ser necesario entrenar modelos específicos para ellas y, además, puede ser necesario un re entrenamiento a partir de los resultados de modo de ajustar el comportamiento erróneo.

En la figura 17, se puede observar los positivos encontrados por el *tokenizer* para el mismo conjunto de datos. En este caso, los resultados muestran un nivel muy bajo de falsos positivos. La precisión puede ser incluso mejorada agregando nuevas excepciones al analizador.

Aunque en el segundo caso los resultados sean mejores, es destacable que su flexibilidad es relativamente menor. Esto se debe a que carece de la posibilidad de inferir resultados, lo cual lo deja limitado a la semilla definida cuando el modelo fue creado.

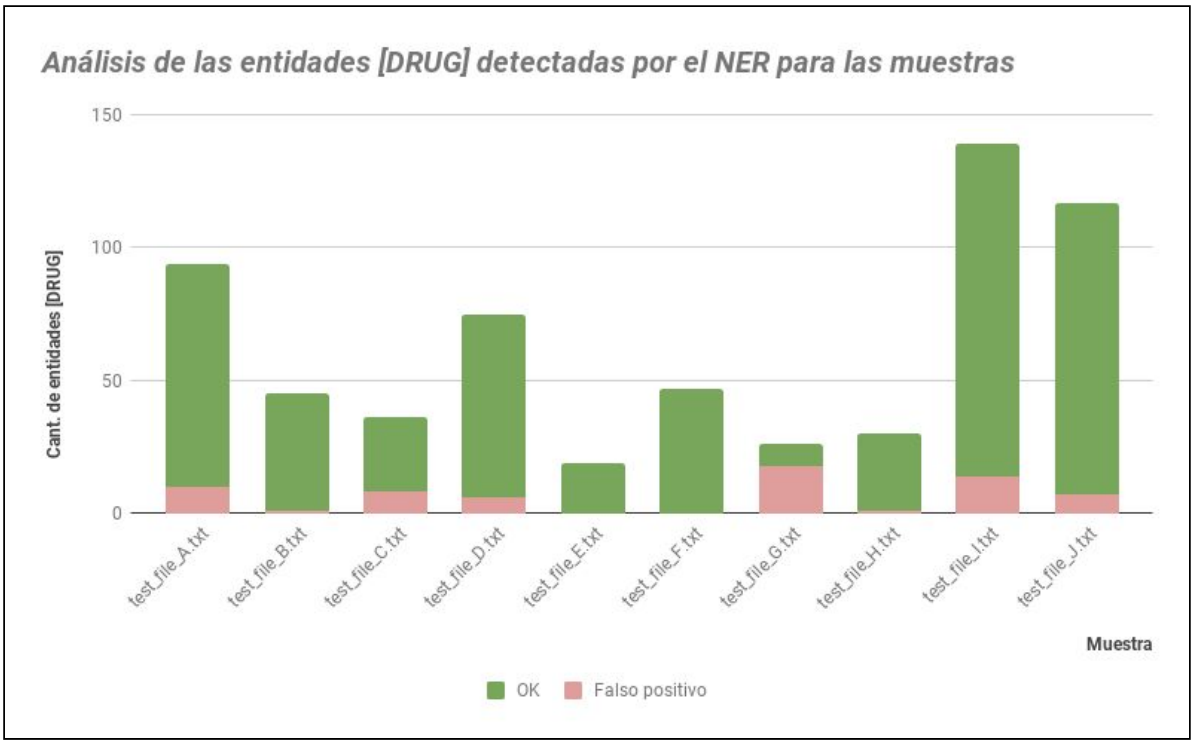

Figura 16. Resultados del análisis realizado con el NER.

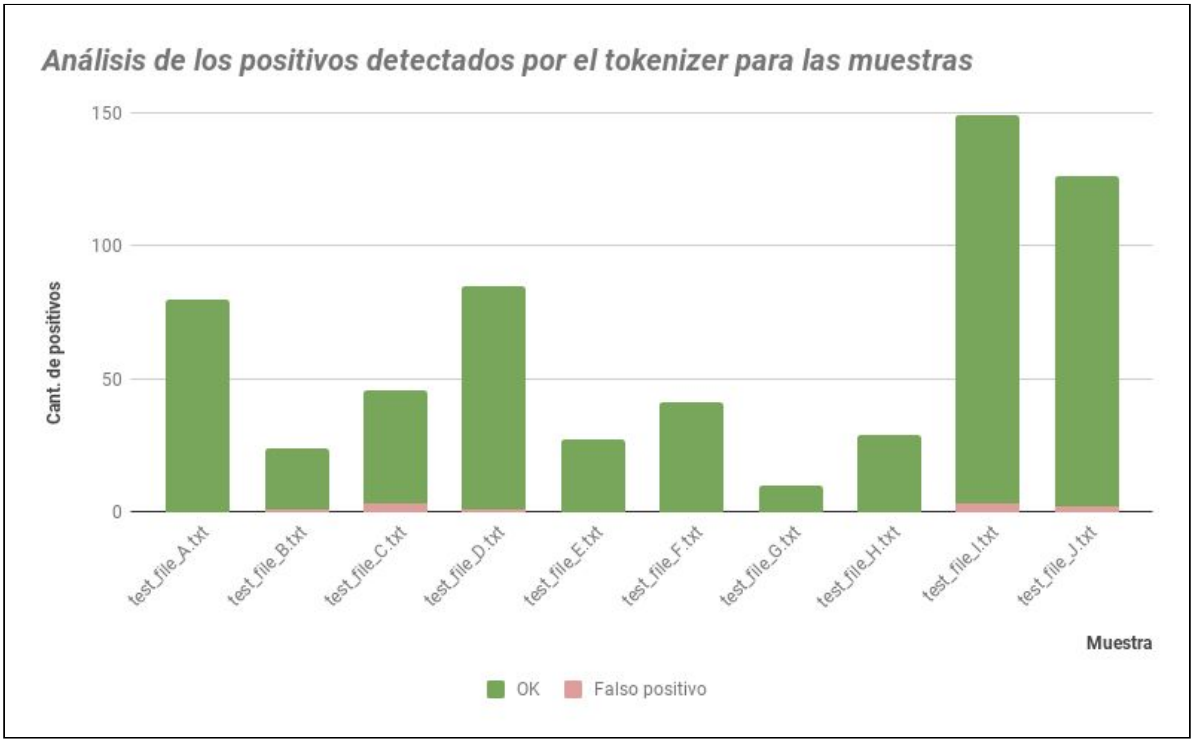

Figura 17. Resultados obtenidos por el análisis del *tokenizer*.

Por último, en la figura 18, se puede observar la comparación del porcentaje de falsos positivos detectado para cada uno de los módulos. Como ya se ha mencionado, si bien

puede parecer más eficaz los resultados del *tokenizer*, ambos análisis son necesarios y sus resultados se complementan entre sí.

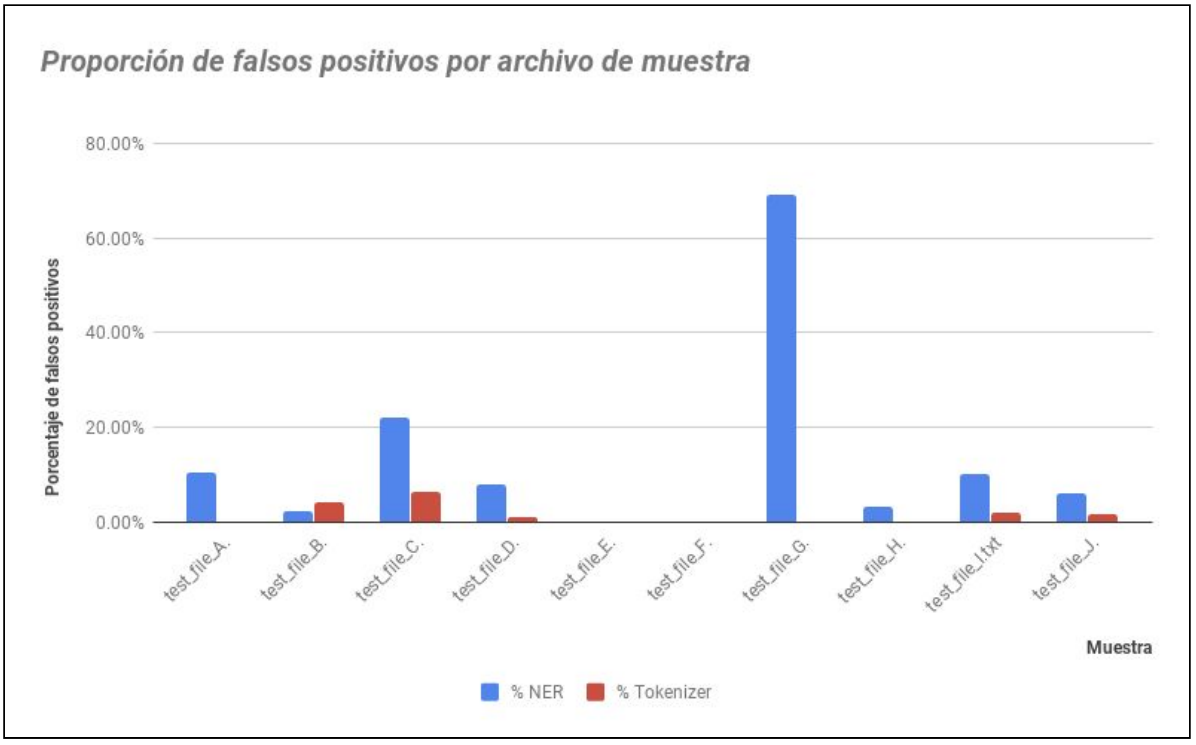

Figura 18. Comparación entre los resultados del *NER* y el *tokenizer*.

Gestión del proyecto

Este proyecto tuvo una particularidad: nació a partir de un prototipo. Esto hizo que la curva de aprendizaje de las tecnologías o herramientas necesarias se amortice durante esos periodos. Al mismo tiempo, dado que las tareas y/o avances realizados durante las prácticas profesionales no tenía como objetivo final el desarrollo del sistema (al momento de realizarlas), todo ese trabajo no fue planificado y controlado. Esto implica que a los tiempos del proyecto se le deben adicionar entre 200 y 250 horas de trabajo. Estas horas corresponden principalmente a investigación, definición de los requerimientos del sistema y diseño. Para simplificar la estadística, los gráficos e información que se expone a continuación omiten esta información.

En la propuesta de proyecto se realizó una planificación que desde el comienzo no pudo ser respetada. Las obligaciones externas hicieron que se re planifique varias veces y que una planificación de mediano / largo plazo sea impracticable. Cada uno de los integrantes tuvo factores tales como el trabajo, la familia o las cursadas que hicieron que

los tiempos se extiendan. Hubo periodos más intensos en cuanto a actividad y hubo algunos en los cuales la misma fue casi nula.

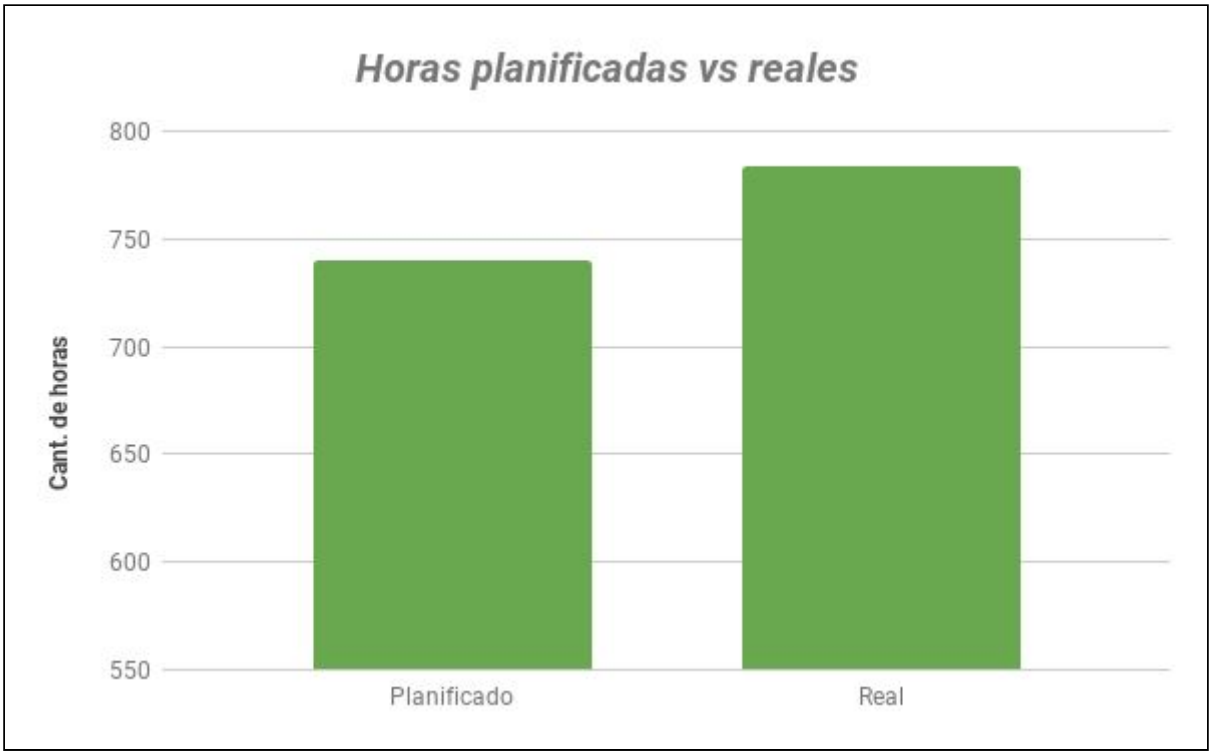

Figura 19. Comparación entre las horas planificadas y reales destinadas al proyecto.

Como se ha dicho anteriormente, todas las tareas realizadas han sido estimadas y planificadas. En la figura 19, se puede observar la comparación entre las horas totales estimadas y las horas reales trabajadas. Se planificaron en total 740 horas, pero se trabajaron 784 realmente. Se puede observar entonces una desviación de 5,95% entre el tiempo planificado y el real. El análisis de las horas reales que ha demandado el proyecto, y las eventuales desviaciones, ofrece información y un aprendizaje muy importante para la estimación de futuros proyectos.

Para medir la gestión del proyecto se eligieron distintos indicadores. Dada la naturaleza del proyecto, se puede observar que la distribución del trabajo tuvo periodos con distintas características. En la figura 20, se puede observar la progresión del trabajo en función de cómo se fueron completando las horas planificadas. Se pueden observar períodos de baja actividad e, incluso, períodos con actividad nula. Dichos periodos se explican por un lado en la cursada del segundo cuatrimestre de 2018 y, por otro, en periodos donde uno de los integrantes tenía una carga de actividades laborales y personales elevada. A partir de comienzos de 2019, se puede observar que, fundado en el

receso de verano, vacaciones y la inexistencia de cursadas, la actividad presenta un pico y se mantiene a un ritmo estable hasta la finalización del proyecto.

Para cada componente principal del sistema se invirtió una determinada cantidad de horas. En la figura 21, se puede apreciar la distribución del trabajo en cada uno de ellos. Como puede verse, del total de horas un 40,80% se destinó a Cerberus y un 59,20% a la librería de administración de modelos.

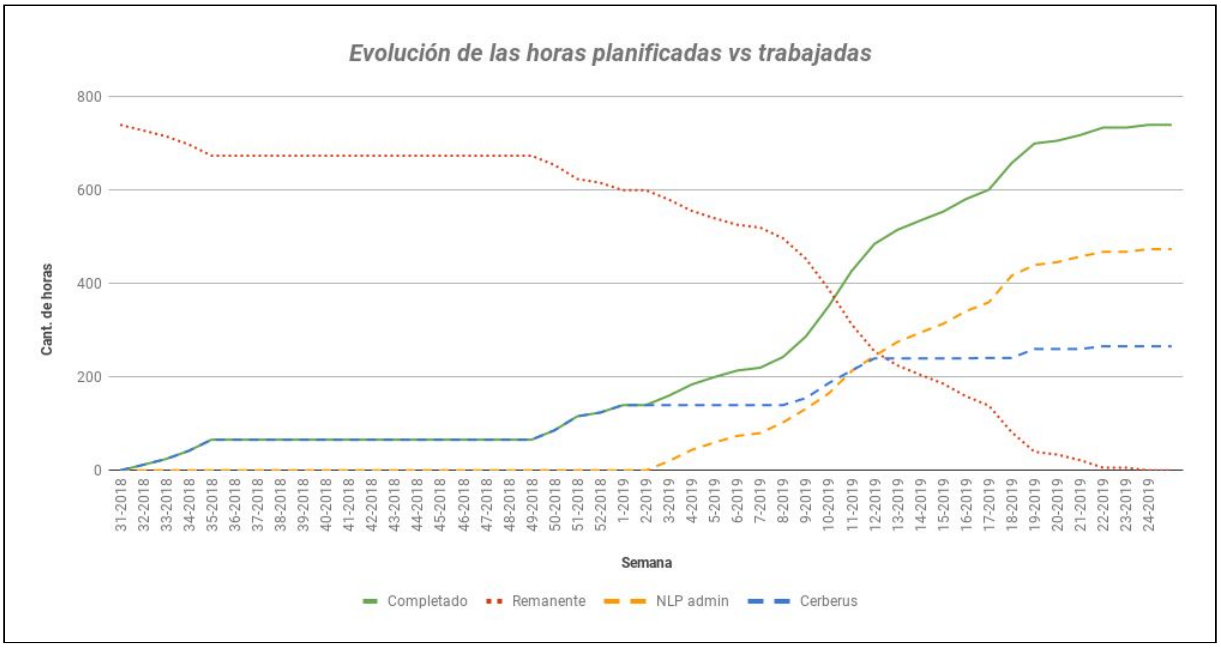

Figura 20. Progreso del avance en el proyecto.

Las horas trabajadas en el sistema Cerberus se distribuyeron como se indica en la figuras 22 y 23 entre sus componentes y subcomponentes respectivamente. Mientras que las horas trabajadas en la librería de administración de modelos fueron distribuidas como se indica en la figura 24.

Finalmente, en la figura 25 puede observarse la distribución del tiempo en función del tipo de tarea. Cabe destacar nuevamente que una gran cantidad de horas de diseño fueron absorbidas durante la etapa de prototipado del sistema. De este modo, la porción de horas destinada a este tipo de actividades crecería en torno al 30-35% del total del proyecto.

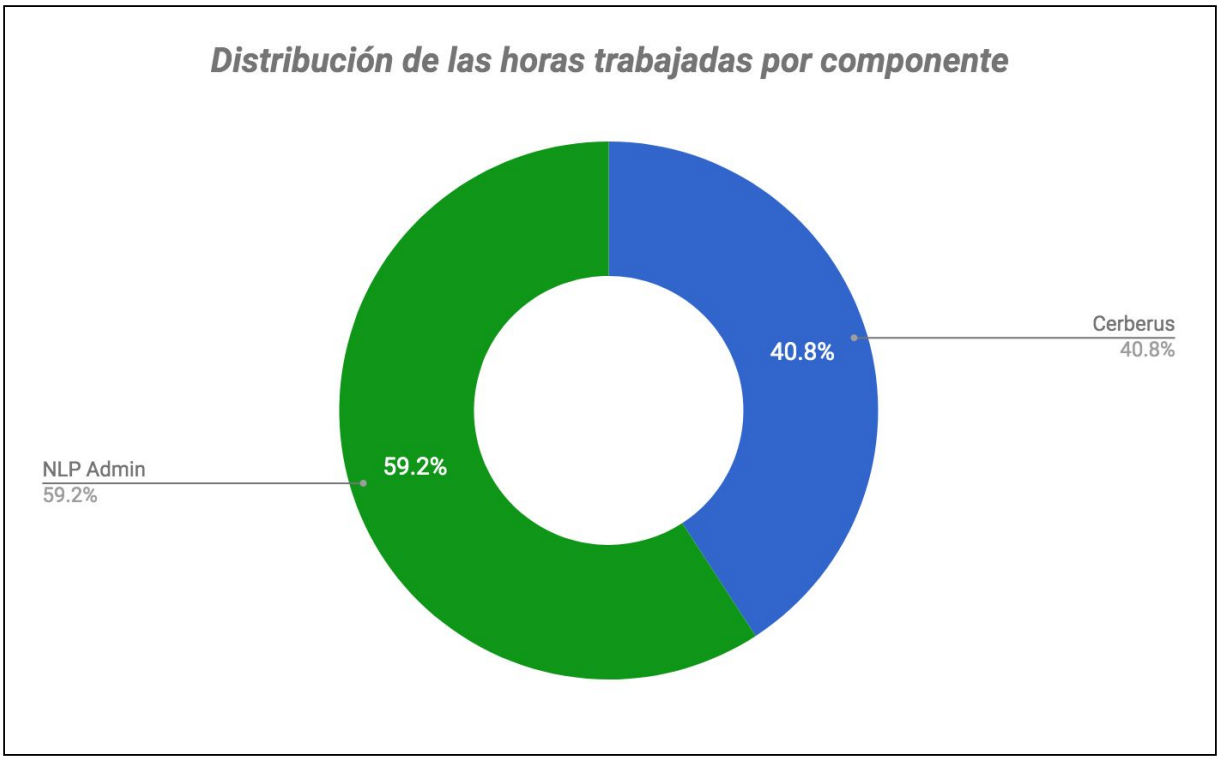

Figura 21. Distribución de las horas trabajadas por componente del proyecto.

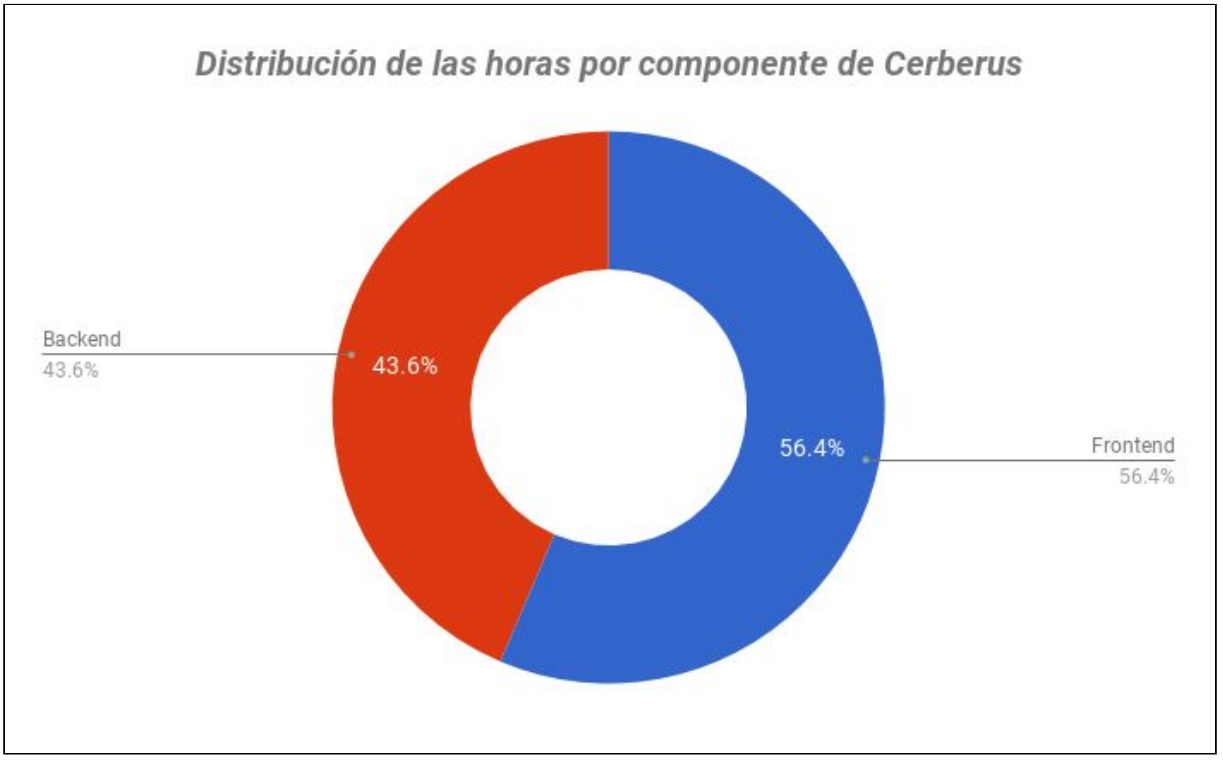

Figura 22. Distribución de horas por componente en el módulo Cerberus.

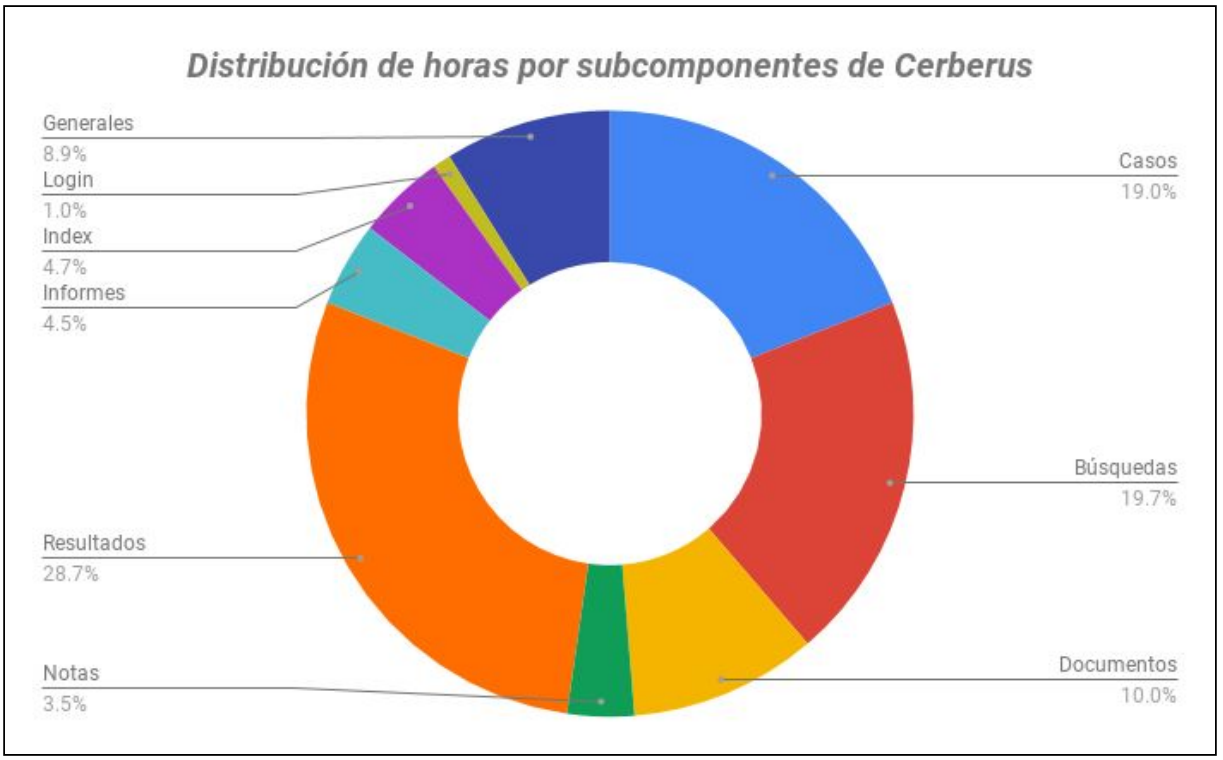

Figura 23. Distribución de horas por subcomponente en el módulo Cerberus.

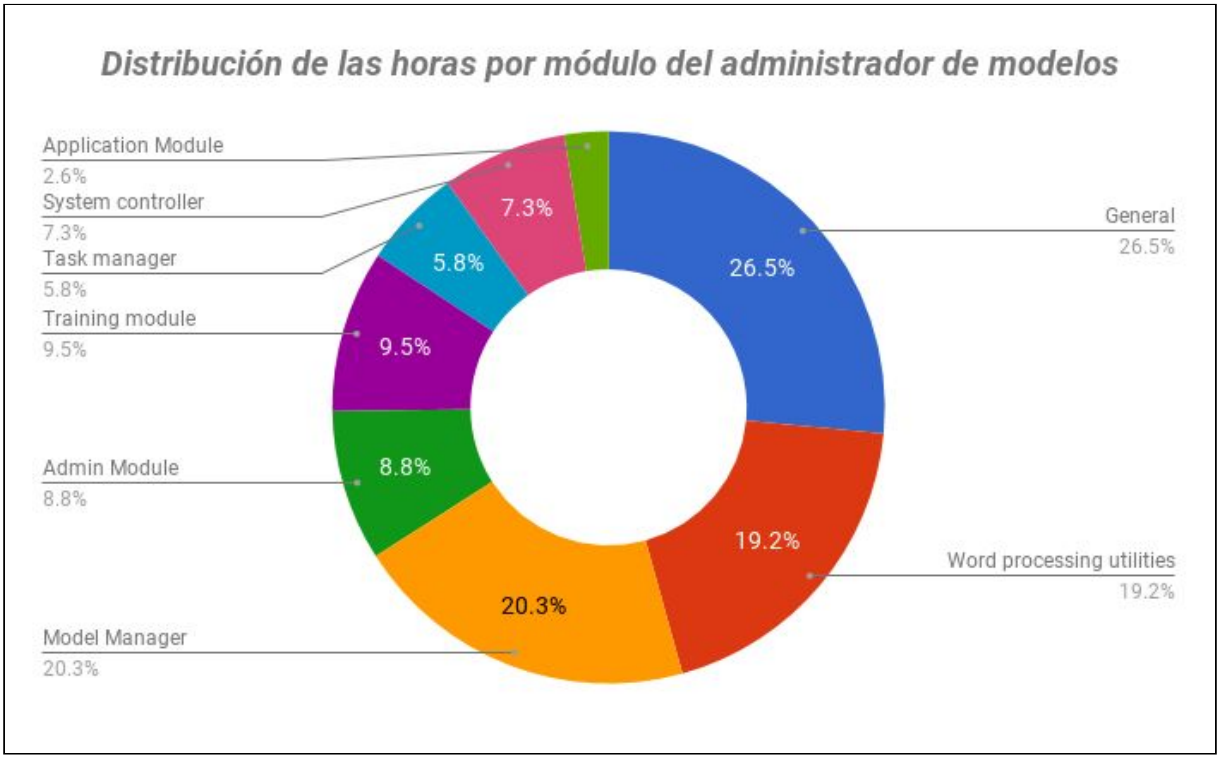

Figura 24. Distribución de horas por subcomponente del módulo de administración.

5.3.2. Retroalimentación

La recepción del trabajo por parte de los clientes, en este caso el Info-Lab, fue satisfactoria por varios aspectos. En principio, es destacable el hecho de que actualmente no cuentan con ninguna herramienta que solucione este tipo de problemas. La realidad muestra que el prototipo, construido en febrero de 2018, está siendo utilizado (con todas sus falencias) en Mar del Plata, mientras que en Bahía Blanca y Necochea se utiliza un modelo en español base de SpaCy. La versatilidad del administrador de modelos en conjunto con la usabilidad añadida por Cerberus a las tareas de análisis suponen un salto de calidad para esta herramienta.

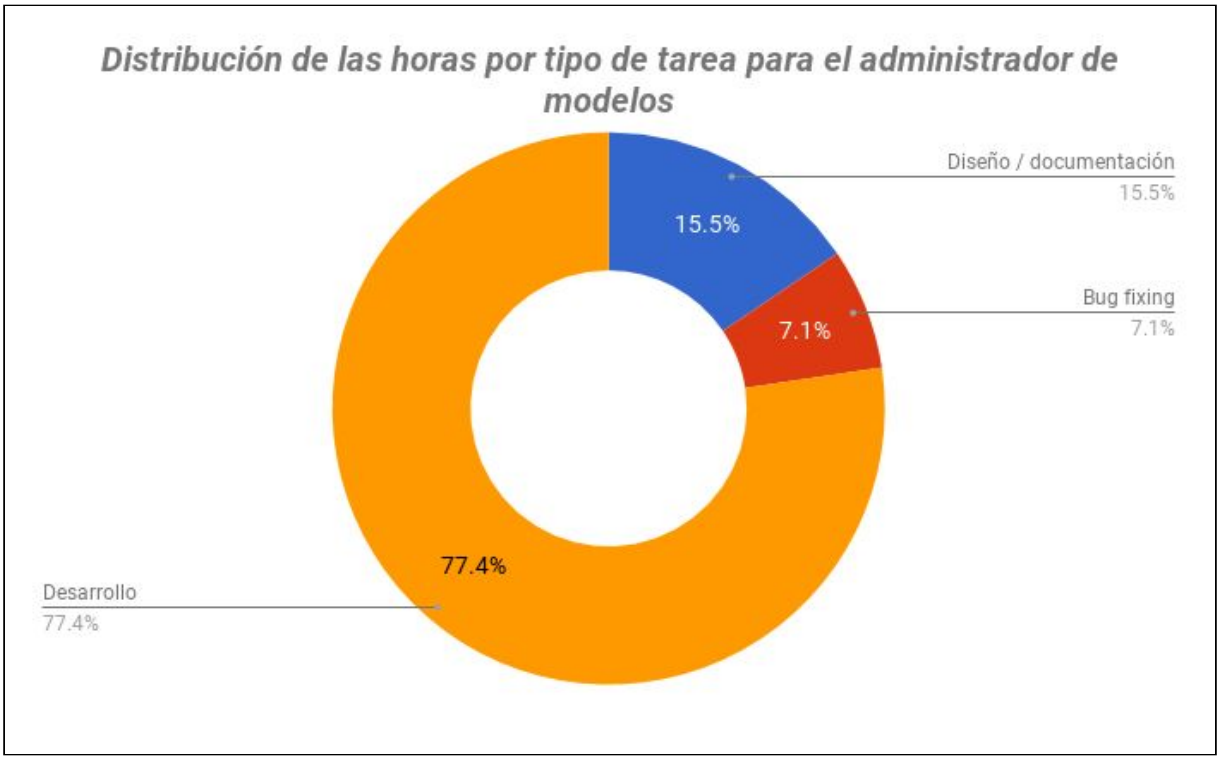

Figura 25. Distribución de horas por tipo de tarea.

La flexibilidad ha sido destacada dado que no se ofrece una herramienta estática y poco configurable. Este aspecto es realmente importante ya que los *argot* cambian constantemente y un modelo puede perder capacidad de detección en el corto o mediano plazo. Contar con una herramienta capaz de ser reformulada y re entrenada en un período corto de tiempo ha sido también altamente valorado.

La interfaz gráfica es otro de los puntos valorados al ofrecer una visualización rápida, dinámica y, además, permitir a los usuario administrar casos y documentos. Del

mismo modo, otro aspecto valorado es que dicha interfaz, fue construida en conjunto con los usuarios finales del sistema, de modo que la mayor parte de sus necesidades y expectativas fueron satisfechas.

Si bien los modelos generados como muestra son más bien demostrativos, han demostrado tener una buena capacidad de detección. De todas maneras, el Info-Lab ha decidido invertir recursos propios para, a través del trabajo de pasantes, generar ejemplos de entrenamiento para enriquecer los modelos que ofrece el sistema. Estos trabajos están programados para comenzar a finales del mes de mayo de 2019. Según directivos del instituto, se espera que con esta actividad se obtengan modelos aún más poderosos.

De las conversaciones mantenidas con personal afectado a las investigaciones se ha destacado, más allá de la mejora de la eficiencia, el impacto en las condiciones de trabajo. Como ya se ha mencionado, este tipo de tareas es fatigante y tediosa para el profesional. Leer cientos de miles de líneas de texto no resulta agradable. Si bien la herramienta desarrollada no es capaz aún de descartar información, permite encontrar en una forma ágil y sencilla los puntos más probables donde se encuentra la evidencia. De este modo, se puede encontrar indicios determinantes sin analizar la totalidad de los datos.

Finalmente, los clientes han destacado el enorme abanico de trabajos futuros que pueden ser desarrollados tomando como base este proyecto. Los mismos van desde mejoras gráficas hasta proyectos ambiciosos como la incorporación de funcionalidades para interpretar audio.

6. Conclusiones

A nivel proyecto se observaron varios puntos buenos pero también muchos aspectos a mejorar. Se destacan varios elementos positivos; la responsabilidad para con el trabajo, el compromiso y apoyo del cliente, las estimaciones han sido bastante acertadas, se trabajó de forma ordenada siguiendo una metodología definida y en equipo.

Por otro lado, se detectaron muchos elementos a mejorar o que podrían haber sido realizados de una mejor manera. Existieron decisiones de diseño erróneas o que no fueron muy acertadas. Por citar algunos ejemplos de los mencionados a lo largo de este trabajo: no haber realizado pruebas exhaustivas en cuanto a la portabilidad de algunos componentes de la aplicación, el orden en cómo se implementaron algunos módulos no resultó el mejor, la planificación no siempre pudo ser respetada por haber obviado factores externos y algunos requerimientos que surgieron en medio del desarrollo generando cambios no previstos.

No obstante, se considera que fueron más los aspectos positivos que los negativos y, además, reconocer los errores permite capitalizarlos para no volver a cometerlos en el futuro.

Se considera que ha sido posible cumplir con cada uno de los objetivos propuestos al comienzo del proyecto. A continuación se detalla una breve reseña de cada uno de ellos:

- ***Analizar los textos y, de este modo, permitir al investigador focalizar su atención sobre los fragmentos con mayor probabilidad de contener evidencia***. La aplicación de modelos personalizados de procesamiento de lenguaje natural permite que los usuarios analicen todo tipo de textos en busca de indicios relevantes a la investigación. Para ello aplica el *tokenizer* y el *reconocedor de entidades nombradas* para encontrar datos relevantes.
- ***Permitir al administrador del sistema administrar los diferentes modelos, orientados a temáticas particulares, con los que cuente el sistema***. El administrador de modelos permite la consulta, edición y borrado de los modelos del sistema. Por decisión del personal del Info-Lab, se definió que no sea necesario implementar una interfaz gráfica del administrador de modelos. No obstante, se documentó completamente la API de modo de facilitar el uso del módulo de administración.
- ***Permitir crear modelos especializados en distintas temáticas***. El producto permite generar modelos de entrenamiento de forma rápida y efectiva. Se pueden crear tantos modelos como sea necesario y utilizar aquel que sea más apropiado para un determinado análisis.
- ***Permitir administrar los datos de entrenamiento de cada modelo y, al mismo tiempo, ofrecer un mecanismo para aplicar los datos de entrenamiento a los modelos***. El módulo de administración permite crear, aprobar y desaprobar ejemplos de entrenamiento para cada modelo particular. Además, permite crear y editar entidades personalizadas que pueden ser utilizadas para etiquetar nuevos datos de entrenamiento. Finalmente, el sistema permite consultar los ejemplos pendientes, aprobados e, incluso, obtener el historial completo para un modelo. Finamente, permite realizar el entrenamiento por medio de una rutina para mejorar incrementalmente la capacidad de detección de cada temática.
- ***Permitir gestionar los documentos vinculados a investigaciones que lleva a cabo el Ministerio Público, en primera instancia, de manera sencilla y útil***. Cerberus permite administrar todos los documentos relacionados con un

determinado caso. De este modo, se puede organizar y obtener datos de cada uno de los documentos de una investigación particular.

- ***Ofrecer al menos un modelo de análisis base orientado a una temática criminal particular***. Gracias al trabajo conjunto con los integrantes del Info-Lab, se pudo crear un modelo orientado a la temática de drogas. Este modelo ha sido entrenado con el set de ejemplos de entrenamiento realizado por el alumno de Ingeniería en Computación Gastón Migone durante su PPS en el InFo-Lab.
- ***Permitir a los usuarios analizar, de forma amigable, los textos asociados a una investigación pertinente***. Gracias a la herramienta gráfica de Cerberus, pueden no solo realizar el análisis sino también visualizar y administrar los resultados de una manera cómoda y sencilla.
- ***Generar informes que muestren información considerada de importancia respecto de una investigación***. Cerberus no sólo permite obtener resultados de gran utilidad, sino que una característica importante es la posibilidad de exportar informes de extensión .docx generados en base a estos. De esta manera, se puede generar rápidamente reportes que sirvan como apoyo a la hora de presentar los resultados de las investigaciones.

El desarrollo del proyecto ha demostrado la viabilidad en la aplicación de las tecnologías de procesamiento a la problemática del análisis forense en el ámbito de las investigaciones judiciales. Se ha demostrado que se pueden registrar mejoras en torno a tres órdenes de magnitud en cuanto a la eficiencia respecto de un humano y despreciar elementos tales como la fatiga. Se constató antes, durante y después que es una herramienta necesaria y que genera expectativa para los profesionales de la investigación.

La eficiencia demostrada, los buenos resultados arrojados y, por sobre todo, la flexibilidad de la aplicación la convierten en un excelente producto. Además, al tener en cuenta que no existe ninguna aplicación similar resulta el punto de partida para construir nuevas herramientas que profundicen, mejoren y extiendan las prestaciones del producto obtenido.

7. Trabajos futuros

El presente proyecto abre las puertas para la realización de numerosos trabajos orientados a extender las funcionalidades o mejorar las actuales. Se pueden distinguir entre ellas:

- **Implementación de una interfaz gráfica para el administrador de modelos:** el administrador de modelos actual no cuenta con una interfaz gráfica que permita administrar configuraciones y modelos de una manera amigable. Este desarrollo puede mejorar la forma de administrar los modelos y potenciar las herramientas disponibles.
- **Investigación de nuevos componentes y mejoras para el pipeline:** se trata de una categoría amplia, basada principalmente en la investigación de modelos que ofrezcan nuevas funcionalidades. De las sugerencias recibidas a través de la retroalimentación se pueden destacar: clasificación automática de documentos, análisis de sentimiento y establecer relaciones entre diferentes entidades.
- **Mejorar la experiencia de usuario:** desarrollos tendientes a la mejora de la interfaz de usuario. El objetivo general de este trabajo es utilizar la retroalimentación de usuarios reales para mejorar la experiencia de los usuarios.
- **Ampliar el espectro de formatos de archivos a analizar:** implementar mediante desarrollos propios o el uso de librerías de terceros la posibilidad de interpretar nuevos formatos.
- **Agregar nuevos idiomas:** extender la capacidad de la API de procesamiento de lenguaje natural para poder generar modelos que respondan a otros idiomas. Este trabajo implica replantear los módulos de procesamiento de palabras.
- **Mejoras colaborativas:** implementar un sistema de popups en Cerberus que ofrezca a los usuarios ejemplos de entrenamiento para etiquetar. La motivación para esta tarea es hacer de la generación de ejemplos una tarea distribuida. Del mismo modo, ofrecer la posibilidad de informar falsos positivos para que el administrador de sistema pueda refinar paulatinamente los modelos ofrecidos.
- **Entrenamiento semi supervisado:** implementar un proceso que permita el entrenamiento semi supervisado puede ser un salto de calidad para el sistema. El objetivo de esta tarea es desarrollar una manera de realizar una rutina de entrenamiento que consuma en parte los ejemplos generados por los usuarios y administradores pero, al mismo tiempo, genere una cantidad similar de ejemplos por medio del análisis de textos preseleccionados.
- **Procesamiento de audio:** otra fuente importante de información son los audios. Cabe destacar que el método de análisis de audio actual presenta una complejidad adicional al tener un tiempo estricto. Si bien el objetivo de este trabajo es el análisis de textos, incorporar un módulo que transforme audio a texto, que luego pueda ser analizado, sería una herramienta extremadamente útil.

8. Glosario

API: Del inglés Application Programming Interface, se trata de un conjunto de funciones y procedimiento que permiten la creación de aplicaciones que acceden a las características o datos de un sistema operativo, aplicación u otro servicio.

Argot: es el lenguaje específico utilizado por un grupo de personas que comparten unas características en común.

Borrado lógico: El borrado lógico es una forma de eliminar elementos de un medio de almacenamiento. Este método de borrado cambia el estado de cada una de las posiciones en el medio que contiene el archivo a disponible. Sin embargo el contenido de las posiciones sigue siendo el mismo hasta que ese espacio sea asignado a otro archivo.

Chatbot: Software de inteligencia artificial que puede mantener una conversación con un usuario a través de aplicaciones de mensajería.

Distancia de Damerau Levenshtein: Compara dos palabras devolviendo un número entero. Dicho valor queda determinado por la cantidad de cambios que se deben aplicar a una de las palabras para que resulte idéntica a la segunda. Se diferencia de la distancia de Levenshtein simple en cuanto a que no considera las trasposiciones.

File carving: Es un proceso que permite recuperar archivos de un medio de almacenamiento sin la asistencia del sistema de archivos que lo creo.

IoT: Del término en inglés *Internet of things*. Su equivalente en español es “Internet de las cosas” y se trata de un tipo de sistema de distribuido donde los nodos pueden ser sensores y actuadores. El sistema como conjunto tiene un objetivo en común y cada uno de estos componentes cumple una función en el cumplimiento de ese objetivo.

Kanban: Herramienta para mapear y visualizar el flujo de trabajo, dentro del cual las tareas pueden encontrarse en uno de tres estados posibles (“Por hacer”, “En progreso”, “Hecha”).

Lema: En lingüística, el lema o ítem lexical es una serie de caracteres que forman una unidad semántica y que pueden constituir una entrada en un diccionario.

Linting: Se denomina linting al proceso de correr un programa que analiza el código en busca de errores. En general, los programas de linting se utilizan en conjunto con un IDE y requieren de un archivo de configuración para funcionar correctamente.

Meta programming: Se trata de una técnica de programación donde por medio de código se generan nuevas instrucciones, en tiempo de ejecución, para ser evaluadas y ejecutadas posteriormente.

Microsoft Office 365: Suite ofimática de Microsoft. Conjunto de aplicaciones orientados a resolver necesidades generales de oficinas. El programa 365, permite a los usuarios mantener actualizados los aplicativos de Microsoft Office en todos sus dispositivos.

Morfema: Unidad lingüística mínima cuyo significado modifica el significado de otros lexemas.

NLP: Siglas que representan al término Natural Language Processing. Se trata del campo de la informática orientado diseñar e implementar software capaz de interpretar y analizar el lenguaje humano.

NoSQL: Del inglés "not only SQL", se trata de una alternativa al modelo relacional de bases de datos. En su mayoría, no se basan en un esquema predefinido de tablas como el modelo relacional, sino que presentan alternativas más flexibles en cuanto a la estructura de los datos que almacenan.

Open source: El software de código abierto es aquel que está licenciado de manera tal que los usuarios puedan estudiar, modificar y mejorar su diseño mediante la disponibilidad del código fuente.

Patrones Gang of Four: Se trata de patrones diseñados en el ámbito de la programación orientada a objetos. Dentro de los mismos, se pueden encontrar patrones creacionales, estructurales y de comportamiento.

Pip: Se trata de una aplicación que permite gestionar los paquetes locales del lenguaje Python. Su funcionamiento es similar a *apt* en Linux o *npm* para Javascript.

Pipeline: Este término puede verse desde distintos enfoques. Desde el punto de vista de un proceso, se trata de un proceso en el que se ejecutan las diversas tareas que lo componen una después de la otra. Una característica importante es que la salida de la primer tarea sirve como entrada de la siguiente tarea en el pipeline.

Portabilidad: En informática, la portabilidad es un tipo de requerimiento, perteneciente al grupo de requerimientos no funcionales, que hace referencia a la capacidad de un software para ser ejecutado en un dispositivo independientemente de la plataforma.

Positivo: En el ámbito de la investigación se denomina positivo a cada elemento detectado de acuerdo al criterio de búsqueda. De mismo modo, un *falso positivo*, es aquel que es detectado (en general por una rutina automatizada) pero que en realidad no coincide con los criterios buscados.

Práctica profesional supervisada (PPS): Se trata de uno de los requisitos académicos requeridos para completar la carrera de ingeniería en la Universidad Nacional de Mar del Plata. Su premisa es que el alumno trabaje 200 horas en un entorno similar al profesional.

Las mismas pueden ser llevadas a cabo en empresas, institutos de investigación o participando en proyectos de la universidad.

Product Owner: Integrante del equipo de Scrum con un sólido conocimiento de los usuarios, el mercado, la competencia y las tendencias a futuro para el tipo de sistema que se está desarrollando. Es el encargado de controlar el cumplimiento de los requisitos estipulados en el backlog.

Pypi.org: Este sitio administra los paquetes de Python. De manera similar a como lo hace NPM para el lenguaje Javascript, Pypi se compone de una comunidad de desarrolladores que ofrecen diferentes librerías o paquetes de su autoría. El sitio web permite administrar los paquetes propios y, mediante una aplicación, se pueden instalar estos paquetes de forma remota en una forma similar a la que ofrece *apt* para Linux.

Scrum Master: Persona responsable de que se sigan las prácticas y valores descritos en el modelo Scrum.

Servicios en la nube (cloud services): La nube es un concepto muy amplio en el campo de los sistemas distribuidos. Su eje principal es la externalización de recursos de un sistema tales como el almacenamiento, el procesamiento o la plataforma en dominios externos que se encargan de administrarlos.

Stakeholder: Persona, organización o empresa que tiene interés en el sistema en desarrollo.

Suite informática: Paquete de programas y aplicaciones, generalmente orientadas a una temática particular, que se comercializa u ofrece como un todo.

9. Bibliografía

- Kishorjit, N., Vidya Raj RK., Nirmal Y., y Sivaji B. (2012). *Manipuri Morpheme Identification, Proceedings of the 3rd Workshop on South and Southeast Asian Natural Language Processing*. Mumbai: COLING.
- Bain (1873). *Mind and Body: The Theories of Their Relation*. New York: D. Appleton and Company.
- Turing, A. M. (1950). Computing Machinery and Intelligence. Recuperado el 24 de abril de 2019 de https://www.jstor.org/stable/2251299?origin=JSTOR-pdf&seq=1#page_scan_tab_contents.
- Blanco Garcia-Moreno*, J. L.* (9 de mayo de 2019). *TensorFlow, machine learning I: Perceptrón multicapa*. Recuperado el 24 de abril de 2019 en

http://miradadelgolem.blogspot.com/2018/05/tensorflow-machine-learning-i.html .

- Larrañaga, P., Inza, I., y Moujahid, A. (s.f.). *Tema 8: Redes neuronales*. Recuperado el 24 de abril de 2019 en http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.
- Matich, D. J. (Marzo de 2001). *Redes Neuronales: Conceptos Básicos y Aplicaciones*. Recuperado el 24 de abril de 2019 en https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf.
- Forensic Toolkit (FTK) Digital Investigations. (2019). *AccessData*. Recuperado el 25 de abril de 2019 en https://accessdata.com/products-services/forensic-toolkit-ftk.
- EnCase Forensic. (2019). *Guidance Software*. Recuperado el 25 de abril de 2019 en https://www.guidancesoftware.com/encase-forensic.
- Carrier, B. (2003-2019). [Kit de aplicaciones]. *Autopsy*. Recuperado el 25 de abril de 2019 en https://www.sleuthkit.org/autopsy/.
- Banerveld, M., Le-Khac, N., y Kechadi, M. (2011). *Performance Evaluation of a Natural Language Processing approach applied in White Collar crime investigation*. *Recuperado* el 25 de abril de 2019 en https://www.researchgate.net/publication/281640675_A_Natural_Language_Processing_Tool_for_White_Collar_Crime_Investigation.
- Microsoft. (2019). LUIS. Recuperado el 2 de junio de 2019 en https://www.luis.ai/home.
- G2 Crowd, Inc. (2019). *Best Natural Language Processing (NLP) Software*. Recuperado el 25 de abril de 2019 en https://www.g2.com/categories/natural-language-processing-nlp.
- Di Iorio, A. (2016). *GUÍA INTEGRAL DE EMPLEO DE LA INFORMÁTICA FORENSE EN EL PROCESO PENAL*. *Recuperado* el 25 de abril de 2019 https://info-lab.org.ar/images/pdf/PAIF.pdf.
- Fedak, V. (29 de junio de 2018). *5 Heroic Tools for Natural Language Processing*. Recuperado el 25 de abril de 2019 en https://towardsdatascience.com/5-heroic-tools-for-natural-language-processing-7f3c1f8fc9f0.
- Perez, J. M. (22 de octubre de 2013). *¿Cuántos mensajes enviamos y recibimos por WhatsApp?. Hijos digitales*. Recuperado el 25 de abril de 2019 en

https://www.hijosdigitales.es/es/2013/10/cuantos-mensajes-enviamos-y-recibimos-por-whatsapp/.

- InfoBae. (16 de enero de 2018). *Analizan las llamadas y más de 100 mil mensajes entre Nahir Galarza y Fernando Pastorizzo*. Recuperado el 25 de abril de 2019 en https://www.infobae.com/sociedad/2018/01/16/analizan-las-llamadas-y-mas-de-100-mil-mensajes-entre-nahir-galarza-y-fernando-pastorizzo/.
- Parikh, M. (12 de octubre de 2018). *Advantages Of Python Over Other Programming Languages*. Recuperado el 25 de abril de 2019 en https://elearningindustry.com/advantages-of-python-programming-languages.
- Chacón Sartori, C. (12 de agosto de 2018). *¿Por qué es Python más preferible para la inteligencia artificial que Java?*. *Quora*. Recuperado el 25 de abril de 2019 en https://es.quora.com/Por-qu%C3%A9-es-Python-m%C3%A1s-preferible-para-la-inteligencia-artificial-que-Java.
- Dhanendran, A. (26 de septiembre de 2014). *Qué tan rápido puede uno llegar a leer*. *BBC*. Recuperado el 1 de mayo de 2019 en https://www.bbc.com/mundo/noticias/2014/09/140925_vert_fut_leer_super_velocidad_np.
- Haykin, S.(2009). *Neural Networks and Learning Machines Third Edition*. Recuperado el 1 de mayo de 2019 en http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf.