Creación eficiente de modelos estadísticos para detección automática y precisa de entidades nombradas

Horacio Miguel Gómez $(L : 50825)$
Juan Pablo Orsay $(L : 49373)$

Proyecto final de carrera

2019-11-21

Índice

1. Introducción

El **Reconocimiento de Entidades Nombradas** (*NER*) es una subtarea del área de *Natural Language Processing* (*NLP*) que busca extraer **entidades nombradas** en textos no estructurados. Estas entidades son luego clasificadas en categorías predefinidas como los nombres de personas, organizaciones, ubicaciones, expresiones de tiempo, cantidades, valores monetarios, porcentajes, entre otras.

Por ejemplo en el siguiente texto:

> Este es el proyecto final de la carrera Ingeniería Informática de los alumnos Gómez y Orsay para el Instituto Tecnológico de Buenos Aires.

Se pueden detectar 3 entidades:

- **Gómez**: Persona
- **Orsay**: Persona
- **Instituto Tecnológico de Buenos Aires**: Organización

El estado-del-arte de los sistemas *NER* producen un rendimiento casi humano(Marsh & Perzanowski, 1998) (cercanos al 95 % de *valor-F*).

A pesar de estos altos valores de rendimiento, la industria tiene dificultades para poder capitalizar la efectividad de dichas sistemas y algoritmos. Es por ello que en este trabajo final hemos tomado la decisión de implementar una plataforma Open Source para revertir esta situación.

2. Definición del problema

2.1. Conceptos básicos

2.1.1. Entidad nombrada

En el contexto de *extracción de información*, una *entidad nombrada* es un objeto del mundo real como lo son personas, ubicaciones, organizaciones, productos, etc; que pueden denotarse con un nombre propio. La entidad puede ser abstracto o tener existencia física. Ejemplos de entidades nombradas son «Mauricio Macri», «Ciudad Autónoma de Buenos Aires», «Apple Macbook». También se suele definir sencillamente como aquellas entidades que se pueden ver como instancias de entidad (por ejemplo, la Ciudad Autónoma de Buenos Aires es una instancia de una ciudad).

2.1.1.1. Definición formal

Formalmente el concepto de «entidad nombrada» se deriva de la definición del filósofo estadounidense **Saul Kripke** de **designador rígido** (Kripke, 1980) que forma parte de la lógica modal y filosofía del lenguaje.

> Un designador rígido designa a una misma entidad en todos los mundos posibles en los que esa entidad existe, y no designa nada en aquellos mundos en los que no existe.

Algunos ejemplos de designadores rígidos son:

- Nombres propios como «Saul Kripke», «Júpiter», «Londres», «4» y «Hércules».
- Descripciones definidas matemáticas como «la raíz cuadrada de 4» y «8 - 2».
- Nombres de clases naturales como «agua» y «bronce».
- Nombres de sensaciones como «dolor» y «alegría».

Por el contrario, los **designadores flácidos** pueden designar diferentes cosas en diferentes mundos posibles y **no** son *entidades nombradas*.

Por ejemplo en la oración: «Mauricio Macri es el presidente de Argentina»:

- «Mauricio Macri» y «Argentina» son *entidades nombradas*, ya que se refieren a objetos específicos.
- «presidente» y «presidente de Argentina» no son *entidades nombradas*, ya que pueden usarse para referirse a muchos objetos diferentes en mundos diferentes:
  - «presidente» puede ser de diferentes países u organizaciones que se refieren a diferentes personas.
  - «presidente de Argentina», si bien hace referencia a un mismo pais, puede ser de diferentes períodos presidenciales que se refieren a diferentes personas.

2.1.1.2. Definición no estricta

Existe un acuerdo general en la comunidad *NER* para considerar como entidades nombradas a otro tipos de entidades que violan el principio de designador rígido. Ejemplos de esto son:

- expresiones temporales como «3 de febrero», «2019».
- expresiones numéricas, como cantidades de dinero y otros tipos de unidades

- expresiones que según contexto denotan una entidad rígida pero no en sí mismas. Por ejemplo «Alfredo Fortabat, empresario argentino, fundador de la compañía cementera Loma Negra.» puede ser considerada una entidad nombrada, sin embargo, el término «Fortabat» por si sólo podría referirse a su viuda «María Amalia Lacroze de Fortabat», al museo de arte «Museo Fortabat» o a la localidad Argentina «Villa Alfredo Fortabat».

En este trabajo hemos priorizado esta definición laxa para tener una mayor expresividad en los tipos de entidades que vamos a detectar. Por lo tanto, en adelante, la definición del término «entidad nombrada» será utilizada bajo una definición no estricta.

2.1.2. Reconocimiento de entidades

El Reconocimiento de entidades nombradas a menudo se divide en dos problemas distintos:

1. Detección de nombres
2. Clasificación de los nombres según el tipo de entidad al que hacen referencia a (persona, organización, ubicación y otro)

En la primera fase los nombres se definen como tramos contiguos de tokens, sin anidamiento, de modo que «Instituto Tecnológico de Buenos Aires» es una entidad única, sin tener en cuenta el hecho de que dentro de esta, la subcadena «Buenos Aires» es en sí otra entidad.

Esta forma de definir el problema, lo reduce a un problema de segmentación.

La segunda fase requiere elegir una ontología para organizar categorías de cosas.

2.1.2.1. Dificultades para encontrar mejores algoritmos

El estado del arte de *NER* desde 2014 con la introducción de Redes Neuronales ha llegado a una meseta (Honnibal, 2017). En los últimos años el diferencial capitalizado por los diferentes grupos de investigación especializados fue muy reducido. Como puede verse en la figura 1.

SYSTEM	TYPE	NER F
spaCy `en_core_web_sm` (2017)	neural	85.67
spaCy `en_core_web_lg` (2017)	neural	86.42
Strubell et al. (2017)	neural	**86.81**
Chiu and Nichols (2016)	neural	86.19
Durrett and Klein (2014)	neural	84.04
Ratinov and Roth (2009)	linear	83.45

Figura 1: Principales avances del estado del arte para NER en los últimos años.

Las razones por las cuales esto ocurre escapa el alcance de este trabajo pero se pueden resumir bajo la ley de los rendimientos decrecientes (Case & Fair, 1999). Se necesita mucho esfuerzo académico para obtener una mejora marginal en el estado del arte actual.

Es por esto que resulta interesante analizar que otro tipo de problemas podemos abordar en este trabajo.

2.1.2.2. Dominios del problema

Existe un hilo conductor en el que todas las investigaciones mencionadas en la figura 1 coinciden; incluso los sistemas *NER* más avanzadas son frágiles, dado que los sistemas NER desarrollados para un dominio no suelen comportarse bien en otros dominios (Poibeau & Kosseim, 2000). La puesta a punto de un sistema *NER* para un nuevo dominio conlleva un esfuerzo considerable. Esto es cierto para modelos basados en reglas y para sistemas estadísticos.

Se entiende por dominio a todos los textos que en su conjunto forman un corpus común. Ejemplos de estos son «Noticias periodísticas», «Textos jurídicos», «Reportes militares», «Papers académicos», etc.

2.2. Los datos son el problema

Por todo lo mencionado, es evidente que el cuello de botella para el avance de esta y muchas areas de la «Inteligencia Artificial» es el la captura de datos, no los algoritmos (Montani, 2016).

En particular para el estado actual del arte de *NER* es necesario tener el cuerpo de textos a analizar (*corpus*) tagueado de tal manera que se conozcan previamente los nombres y tipos de entidades

de un subconjunto de textos para inferir sobre el resto.

Este aprendizaje se carga en lo que se reconoce como un «Modelo estadístico» y es a ese model al que se le pide inferir nuevos resultados. En nuestra experiencia los modelos pre-entrenados de las diferentes plataformas / librerias / frameworks y trabajos académicos resultan siempre insuficientes para el uso en producción de los mismos.

De esta manera queda bien definido el problema que queremos atacar (y el título de este trabajo).

> Creación eficiente de modelos estadísticos para detección automática y precisa de entidades nombradas.

Para este fin se creó un sistema informático que permitirá obtener resultados a la altura de de las soluciones del estado del arte para cualquier corpus de documentos que posea una cantidad suficiente de datos.

3. Estado del arte

El análisis del estado del arte fue basado en la definición del problema.

3.1. Stack de software

Python es el lenguaje más utilizado para resolver problemas de Machine Learning, en especial *NLP* (Elliott, 2019)

Spacy es el *framework* mejor ranqueado para la tarea de *NLP* (Elliott, 2019) y sabemos por la figura 1 que obtiene resultados a-la-par del estado del arte actual.

Además la implementación de *spaCy* es robusta y orientada a la creación de apliciones para producción, a diferencia de muchas otras librerías de *NLP* que sólo se utilizan con fines académicos.

3.2. El pipeline

Todas las operaciones de analisis de lenguaje natural sobre textos no estructurados, tienen como primer paso el de separar el los mismos en tokens. Luego, el documento se procesa en varios pasos diferentes que consisten en el «pipeline de procesamiento». Usualmente los pasos consisten en un etiquetador, un analizador sintáctico y un reconocedor de entidades en el caso de *NER*.

Cada componente del pipeline mostrado en la figura 2 devuelve el un documento *Doc* procesado, que luego se pasa al siguiente componente.

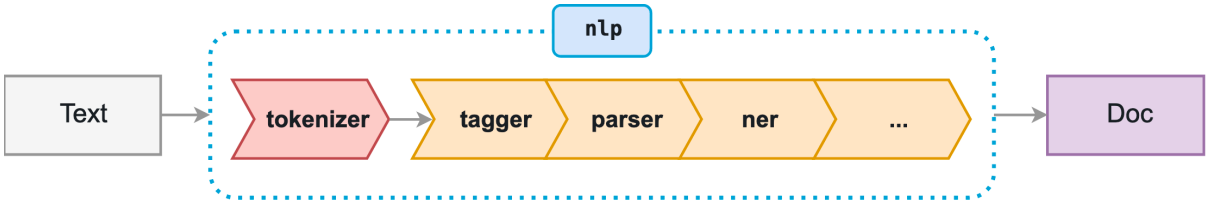


Figura 2: Pipeline standard para los algoritmos de NER.

En este capítulo estudiaremos la morfología de dicho pipeline.

3.3. Algoritmo de tokenización

Para tokenizar un texto de manera correcta no basta con separar el mismo en espacios. Dependiendo el lenguaje que se esté estudiando, existen «excepciones» a esta regla y otros caracteres que representan separaciones entre tokens segun el contexto de los mismos.

En particular, *spaCy* posee algoritmo de tokenización inteligente que puede ser resumido de la siguiente manera:

1. Iterar sobre subcadenas separadas por espacios en blanco.
2. Compruebar si existe una regla definida explícitamente para esta subcadena. Si existe, usarla.

3. De lo contrario, intentar consumir un prefijo. Si consumimos un prefijo, regrese al punto #2, para que los casos especiales siempre tengan prioridad.
4. Si no se puede consumir un prefijo, intente consumir un sufijo y luego regrese al punto #2.
5. Si no se puede consumir un prefijo ni un sufijo, buscar un caso especial.
6. Buscar una coincidencia de token
7. Buscar «infijos» - cosas como guiones, etc. y dividir la subcadena en tokens en todos los infijos.
8. Una vez que no se pueda consumir más de la cadena, tratarla como un token único.

Ejemplo:

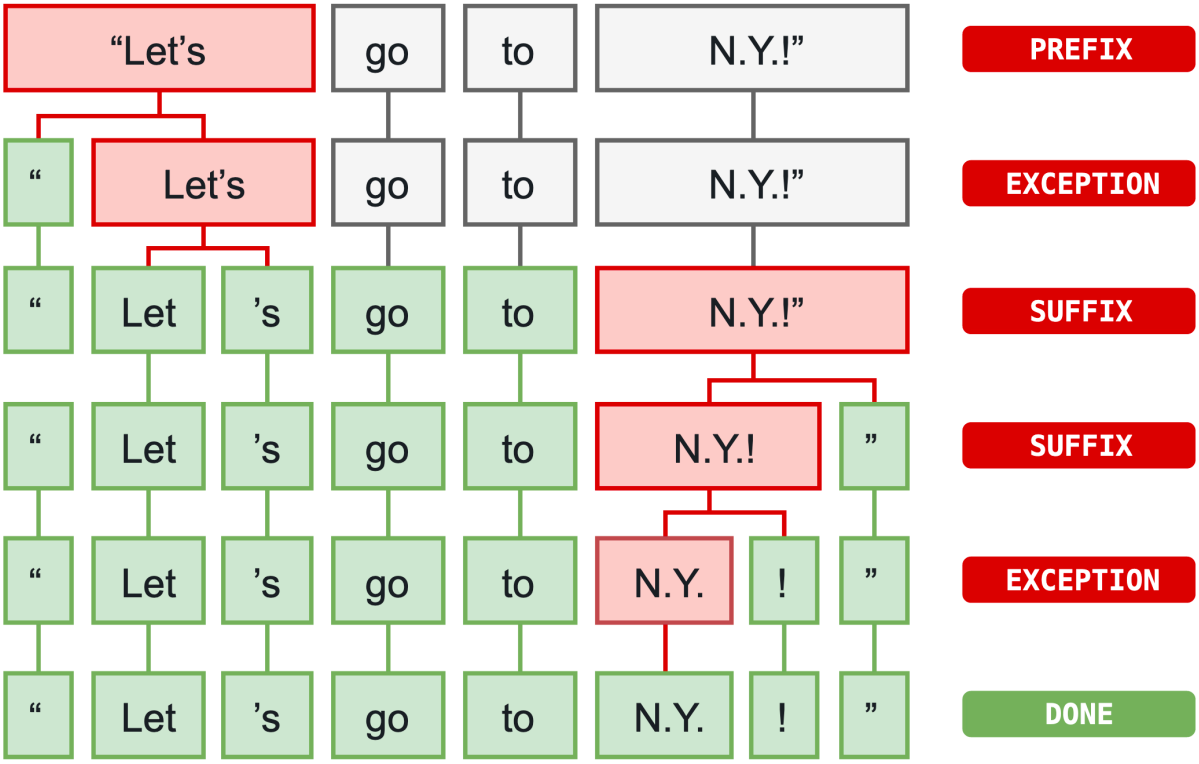


Figura 3: Transiciones del modelo Stack-LSTM indicando la acción aplicada y el estado resultante.

3.4. Modelos basados en reglas

Antes de entrar en detalles de cómo trabaja el modelo estadístico de spacy y entender sus fortalezas es importante esbozar brevemente el grupo de algorítmos más «naive» posible. El de los modelos basados en reglas fijas.

En estos modelos se implementan reglas finitas o expresiones regulares para la detección de las entidades. Las principales limitaciones de este enfoque son:

- **Mucho trabajo manual**: el sistema *Rule Based* exige un profundo conocimiento del dominio, este analisis debe ser realizado por humanos expertos en el dominio.
- **Consumo de tiempo**: la generación de reglas para un sistema complejo es bastante difícil y requiere mucho tiempo.
- **Menor capacidad de aprendizaje**: el sistema generará el resultado según las reglas, por lo que la capacidad de aprendizaje del sistema por sí mismo es baja.

- **Dominios complejos**: si el corpus demasiado complejo, la creación del sistema RB puede llevar mucho tiempo y análisis. La identificación de patrones complejos es una tarea desafiante en el enfoque RB.

3.5. Modelos de «*deep-learning*»

Cuando se busca mejorar el aprendizaje automático, generalmente se piensa en la eficiencia y la precisión, pero la dimensión más importante es la generalidad. Este es el modelo estadístico que usa *spaCy*.

La mayoría de los problemas de NLP pueden reducirse a problemas de aprendizaje automático que toman uno o más textos como entrada. Si podemos transformar estos textos en vectores, podemos reutilizar soluciones de aprendizaje profundo (*deep-learning*) de propósito general.

Las representaciones de palabras embebidas (*embebed-words*), también conocidas como «vectores de palabras» (*word-vectors*), son una de las tecnologías de procesamiento de lenguaje natural más utilizadas en el estado del arte actual. El modelo de *deep learning* utilizado por *spaCy* puede ser descripto en cuatro pasos.

Los *word-embeddings* permiten tratar a las palabras individuales como «unidades de significado», en lugar de identificaciones completamente distintas. A este proceso se le conoce como ***embed***.

Sin embargo, la mayoría de los problemas de *NLP* requieren la comprensión de tramos de texto más largos, no solo palabras individuales. Al juntar un conjunto de *word-embeddings* en una secuencia de vectores, se usan RNN bidireccionales para codificar los vectores en una matriz de oración. Las filas de esta matriz pueden entenderse como «vectores-de-tokens»: son sensibles al contexto del token dentro de la oración. Este paso se lo llama ***encode***.

Finalmente, el mecanismo de ***attend*** le permite reducir la matriz de oración a un vector de oración, listo para la predicción (***predict***).

De esta manera quedan definidas las piezas que describen al modelo de *spaCy*:

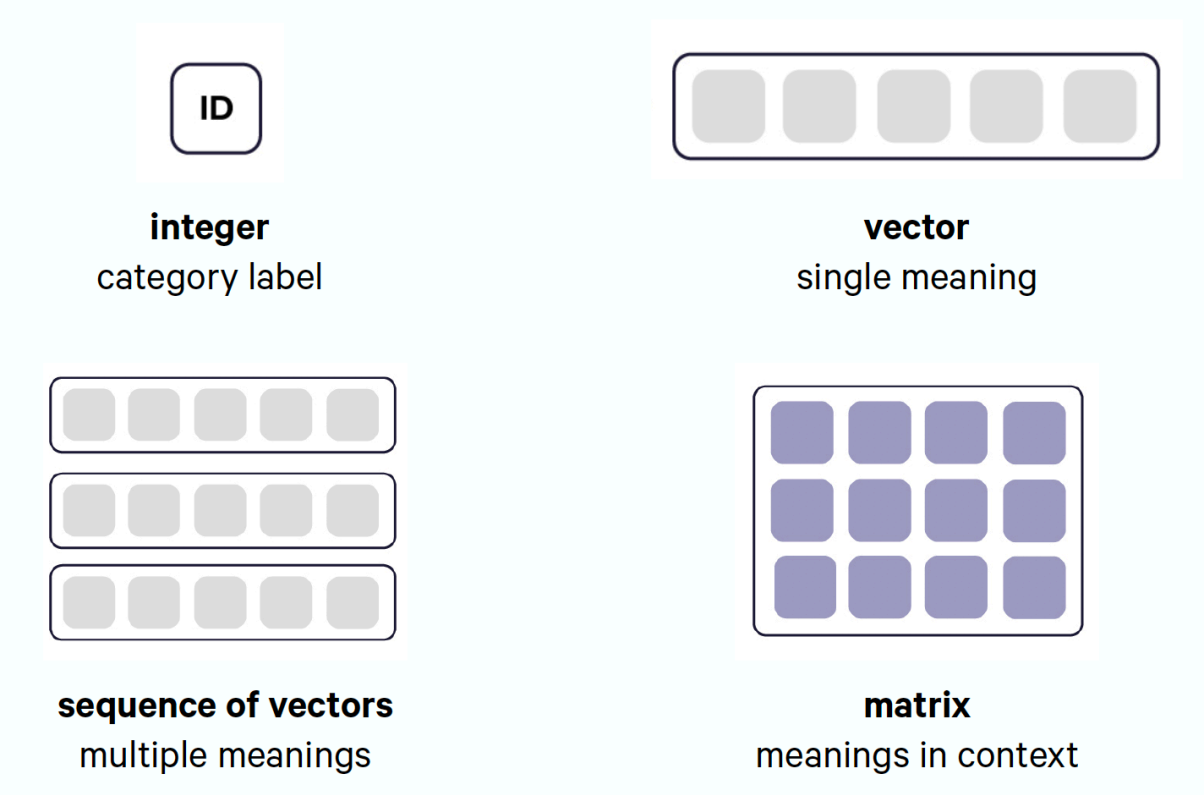


Figura 4: Estados posibles para las diferentes etapas de la CNN.

Estos 4 procesos seran descriptos en detalle en las siguientes secciones.

3.5.1. *Embed*

Resuelve el problema: «todas las palabras se ven iguales para la computadora»

La idea de *word embeddings* es la de «embeber» el conjunto de tokens que componen términos con información adicional. El resultado de esta operación es una estructura abstracta que puede ser descripta como un «vector de significado».

Figura 5: Embed.

Es importante destacar que en la etapa de *embed*, toda la información de significado es principalmente independiente del contexto en el cual esta siendo utilizada y por esta razón es facilmente obtenible del corpus no tagueado de datos (los algoritmos pueden darse cuenta de que palabras están relacionadas entre sí de manera eficiente y no supervisada).

Esto permite al modelo poder inferir significado a partir de la información no etiquetada dentro del problema en particular a resolver (el de *NER*).

Una tabla de *word-embeddings* mapea vectores largos binarios y esparsos en vectores cortos densos y continuos, cargados de significado relevante. Existen varias estrategias para enriquecer los *tokens* con información adicional. Las dos fuentes más grandes de información embebida son las de **información lingüística** y los ***word-vectors***.

3.5.1.1. Información ligüística

El objetivo de ésta etapa es la de encontrar las características intrínsecas de cada palabra. Las principales características lingüísticas detectadas, están resumidas en el siguiente ejemplo (TODO: figura 6).

TEXT	LEMMA	POS	TAG	DEP	SHAPE	ALPHA	STOP
Apple	apple	PROPN	NNP	nsubj	Xxxxx	True	False
is	be	VERB	VBZ	aux	xx	True	True
looking	look	VERB	VBG	ROOT	xxxx	True	False
at	at	ADP	IN	prep	xx	True	True
buying	buy	VERB	VBG	pcomp	xxxx	True	False
U.K.	u.k.	PROPN	NNP	compound	X.X.	False	False
startup	startup	NOUN	NN	dobj	xxxx	True	False
for	for	ADP	IN	prep	xxx	True	True
$	$	SYM	$	quantmod	$	False	False
1	1	NUM	CD	compound	d	False	False
billion	billion	NUM	CD	pobj	xxxx	True	False

Figura 6: Características lingüísticas.

- **Text**: la palabra original texto.
- **Lemma**: La forma básica de la palabra. `Es -> ser`. Esto permite a las siguientes etapas trabajar con una definicón canónica del token.
- **POS**: la etiqueta simple de parte-del-discurso (POS).
- **Tag**: la etiqueta detallada de parte del discurso (POS).
- **Dep**: dependencia sintáctica, es decir, la relación entre tokens.
- **Shape**: la forma de la palabra: mayúsculas, puntuación, dígitos. Muy útil para detectar patrones como números telefónicos, documentos de identidad, CBUs, etc.
- **is alpha**: ¿el token es una secuencia de caracteres alfabéticos?
- **is stop** : ¿es el token parte de una lista de palabras-de-stop, es decir, las palabras más comunes del idioma?

3.5.1.1.1. *Word Vectors*

A diferencia de la información lingüística que es obtenida en el momento, existen otros tipos de *embeddings* más poderosos que son los productos de pre-entrenamientos; como el caso de los *word-vectors*. Los mismos se generan mediante la concatenación de atributos léxicos como *NORM*, *PREFIX*, *SUFFIX* y *SHAPE*. Luego se usa una capa oculta de una red neuronal convolucional (*CNN*) para permitir una combinación no lineal de la información en estos vectores concatenados.

Los *word-vectors* son particularmente útiles para términos que no están bien representados en los datos de entrenamiento etiquetados. En nuestro uso de reconocimiento de entidades, no siempre habrá suficientes ocurrencias. Por ejemplo, en los datos de entrenamiento es posible que existan ocurrencias del término «Coca-Cola», pero ninguna del término «Manaos». Es interesante pensar a las palabras como «vectores de significado». Dentro del espacio vectorial de significados el vector «perro» se encuentra cercano al de «cachorro», «beagle», «bulldog», «poodle». Esto permite al modelo poder inferir nuevas relaciones en base a una cantidad reducida de entradas. Los *word-vectors* ponen ese hecho a disposición del modelo de reconocimiento de entidades. Si bien no existen ejemplos de «Manaos» etiquetados como «Producto»; se verá que «Manaos» tiene un vector de palabras que generalmente corresponde a los términos de un producto, por lo que puede hacer la inferencia. Y si se quiere llegar incluso al detalle de que es una «Gaseosa».

Otra forma interesante de analizar y entender los *word-vectors* en su contexto de espacio vectorial multidimensional de significados es a través del algebra de vectores, como se muestra en el trabajo «*Towards Understanding Linear Word Analogies*» (Ethayarajh, Duvenaud, & Hirst, 2019):

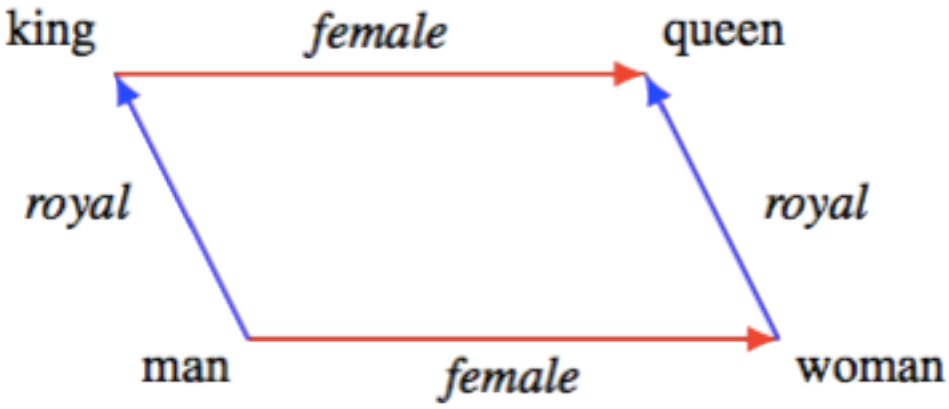


Figura 7: Parallelogram structure in the vector space (by definition).

Es facil de entender viendo la figura 7 que al realizar algebra entre los diferentes vectores de significado se pueden inferir nuevos conceptos:

$$\vec{Rey} - \vec{Hombre} \approx \vec{Realeza}$$
$$\vec{Rey} - \vec{Hombre} + \vec{Mujer} \approx \vec{Reina}$$

3.5.2. *Encode*

Resuelve el problema: el contexto de los significados es relevante y esta siendo descartado,

El resultado de esta etapa es la de codificar vectores **independientes-de-contexto** en matrices de oración **sensibles-al-contexto**.

Figura 8: Encode.

La tecnología utilizada para esta etapa es una Red Neuronal Convolucional en la que la oración es analizando como una *moving window* de tres vectores en la que cada vector analiza su significado en relación con un vector previo y un vector posterior. Es decir cada vector se estudia dentro del contexto de los dos vectores que lo rodean. Luego los vectores subsiguientes se evaluan de igual manera, por lo que se genera naturalmente un «efecto de decaimiento» en el que el contexto de los vectores más lejanos tiene una relevancia cada vez menor.

3.5.3. *Attend*

> Resuelve el problema: tenemos demasiada información para inferir un significado específico al problema a resolver.

En esta etapa toda la información generada en las etapas anteriores es analizada a través de un vector de entrada o tambien conocido como «vector de consulta» o «vector de contexto» representado en la figura 9 como un vector mas oscuro.

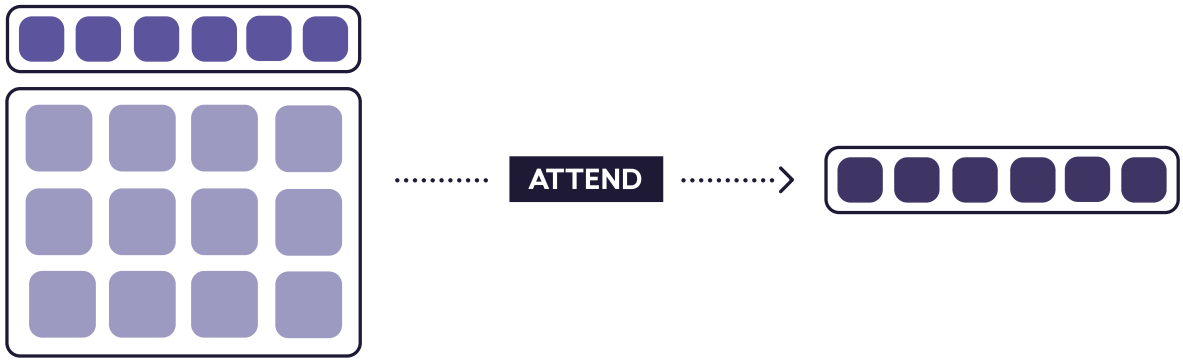


Figura 9: Attend.

Al reducir la matriz a un vector, necesariamente se está perdiendo información. Es por eso que el «vector de contexto» es crucial: Indica que información descartar, de modo que el «vector resumen» se adapte a la red que lo consume.

El análisis de estas estrategias de consulta escapa el alcance de este trabajo pero resulta un tema interesante de investigación en sí mismo. Por ejemplo, investigaciones recientes han demostrado

que el mecanismo de atención es una técnica flexible y que se pueden usar nuevas variaciones para crear soluciones elegantes y poderosas. Por ejemplo, en el estudio de Ankur Parikh et al (Parikh, Täckström, Das, & Uszkoreit, 2016) presentan un mecanismo de atención que toma dos matrices de oraciones y genera un solo vector.

3.5.4. *Predict*

> Resuelve el problema: necesito un valor específico y no una representación genérica abstracta.

Finalmente en esta etapa tenemos un nuevo «vector de significado» que resulta de la consulta a la etapa anterior. Es necesario ahora traducir este vector a un *token* efectivo. En el caso de *NER*, el *token* que interesa obtener es el de la etiqueta de entidad.

Figura 10: Predict.

3.6. Uso del modelo estadístico

Experimentos en inglés, holandés, alemán y español muestran que se pueden obtener resultados a-la-par del estado del arte mediante el uso de un «autómata finito determinístico de pila» en conjunto con una red neuronal (Lample, Ballesteros, Subramanian, Kawakami, & Dyer, 2016).

Este autómata de pila es el nexo entre la Red Neuronal Convolucional (*CNN*) que contiene el modelo estadístico para predecir entidades en el texto completo. En el mismo, se envían cada uno de los estados porl los que se mueve para ir generando entidades con una herística del tipo *greedy*.

La figura 11 muestra las posibles acciones de transición de este autómata.

- SHIFT: consume una token del input y al mueve al stack para generar una nueva entidad.
- REDUCE: mueve el stack actual al output tagueado como entity.
- OUT: consume una token del input y la mueve sl output directamente.

$\textbf{Out}_t$	$\textbf{Stack}_t$	$\textbf{Buffer}_t$	**Action**	$\textbf{Out}_{t+1}$	$\textbf{Stack}_{t+1}$	$\textbf{Buffer}_{t+1}$	**Segments**
O	S	$(\mathbf{u}, u), B$	SHIFT	O	$(\mathbf{u}, u), S$	B	—
O	$(\mathbf{u}, u), \ldots, (\mathbf{v}, v), S$	B	REDUCE(y)	$g(\mathbf{u}, \ldots, \mathbf{v}, \mathbf{r}_y), O$	S	B	$(u \ldots v, y)$
O	S	$(\mathbf{u}, u), B$	OUT	$g(\mathbf{u}, \mathbf{r}_\varnothing), O$	S	B	—

Figura 11: Transiciones del modelo Stack-LSTM indicando la acción aplicada y el estado resultante.

Para saber que acción tomar se consulta el modelo estadístico. En la figura se puede ver un ejemplo de cómo se recorre una oración bajo el stack propuesto:

Transition	Output	Stack	Buffer	Segment
	[]	[]	[Mark, Watney, visited, Mars]	
SHIFT	[]	[Mark]	[Watney, visited, Mars]	
SHIFT	[]	[Mark, Watney]	[visited, Mars]	
REDUCE(PER)	[(Mark Watney)-PER]	[]	[visited, Mars]	(Mark Watney)-PER
OUT	[(Mark Watney)-PER, visited]	[]	[Mars]	
SHIFT	[(Mark Watney)-PER, visited]	[Mars]	[]	
REDUCE(LOC)	[(Mark Watney)-PER, visited, (Mars)-LOC]	[]	[]	(Mars)-LOC

Figura 12: Secuencia de tranciciones para el ejemplo "Mark Watney visited Mars.[en] el modelo de Stack-LSTM.

- Primero se empieza con un stack vacío.
- Se consume «Mark» y la CNN predice que es una posible Persona. Lo envia al stack.
- Se consume «Watney» y la CNN predice que es una posible continuación de Persona. Lo envia al stack.
- Se consume «visited» y la CNN predice que esto no forma parte de una entidad. Por lo tanto antes se REDUCE la entidad «Mark Watney» del stack actual.
- Análogamente se detecta la entidad «Mars»

4. NERd (Implementación)

Definido el problema, queda claro que la creación de un modelo entrenado es de vital importancia para cualquier problema de tagueo de entidades. Es por ello que en el presente proyecto final hemos creado una herramienta para el entrenamiento eficiente de modelos estadísticos así como también una interfaz y API para poder consultar entidades. El nombre de esta herramienta es **NERd**, sigla cuyo significado en inglés es ***N**amed **E**ntity **R**ecognition **D**uh*[1]!

Para organizar este capítulo vamos a realizar una descripción basada en el modelo de vistas de arquitectura 4+1 (Figura 13.

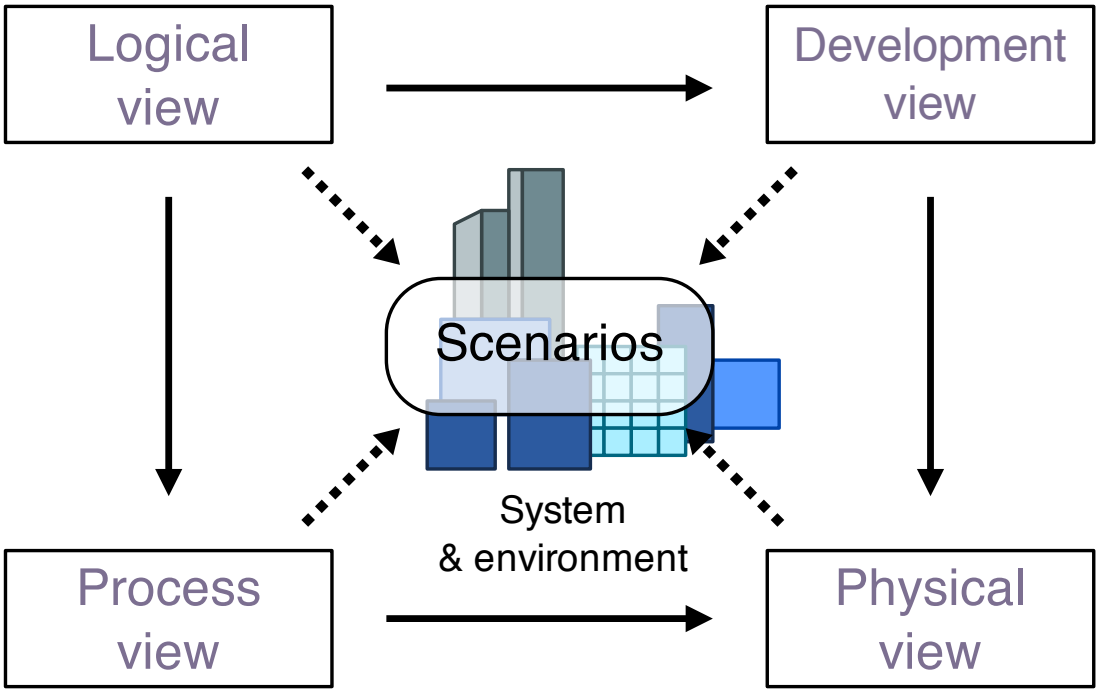


Figura 13: Ilustración de arquitectura 4+1.

Este modelo nos permite describir la aplicación de una manera genérica y ordenada.

> *The «4+1» view model is rather «generic»: other notations and tools can be used, other design methods can be used, especially for the logical and process decompositions, but we have indicated the ones we have used with success.*
>
> — (Kruchten, 1995)

4.1. Vista lógica

La vista lógica se refiere a la funcionalidad que el sistema proporciona a los usuarios finales.

[1] Expresión de obviedad. *Used to express your belief that what was said was extremely obvious* ("Duh definition," 2019)

A continuación detallamos distintas partes del servicio NERd así como también de la interfaz de entrenamiento.

4.1.1. Web

La página web de *NERd* está enfocada en las tareas de mantenimiento de los servicios ofrecidos por el *API* así como también ofrece de interfaces que permiten a usuarios del sistema corregir de manera eficiente el modelo de inferencia.

4.1.1.1. Inicio

Pantalla de inicio donde se encuentran accesos rápidos para entrenar el modelo o para poder buscar entidades en textos. También se encuentra aquí una lista de los 5 usuarios que más contribuyeron a entrenar el modelo. Detrás de esta funcionalidad se busca generar un espíritu competitivo entre los usuarios para que los mismos busquen contribuir más (Figura 14).

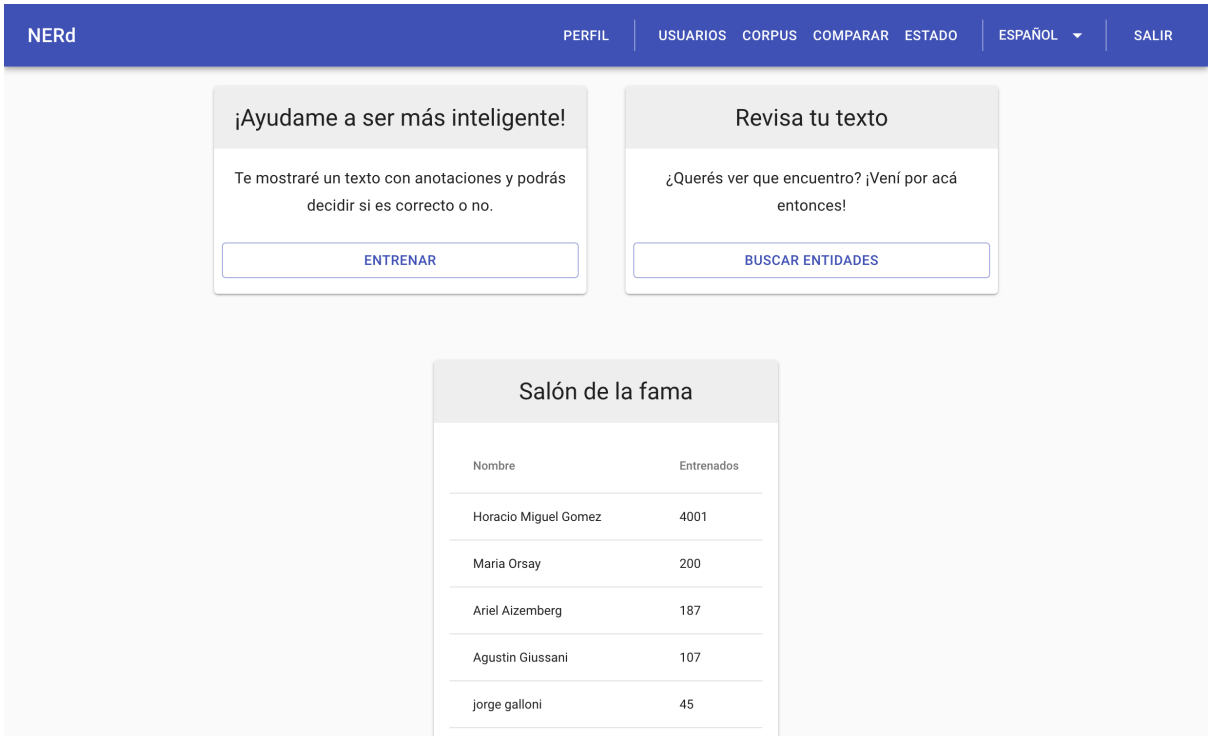


Figura 14: Pantalla de inicio con usuario logueado.

Si la persona no cuenta con permisos de entrenador, se le sugiere que contacte a un administrador para que le otorgue el permiso (Figura **??**).

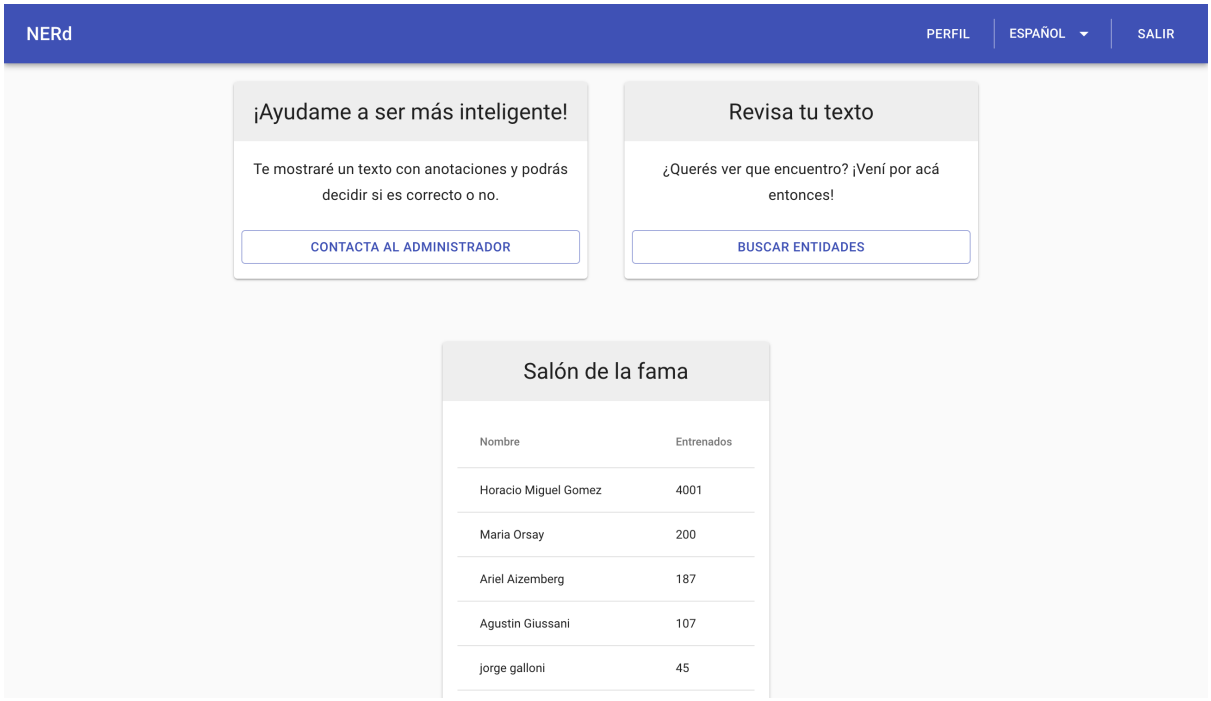


Figura 15: Pantalla de inicio sin rol de entrenador.

Si la persona visitando la página no cuenta con una sesión activa, se le invita a ingresar con una cuenta pre-existente o a registrarse (Figura 16).

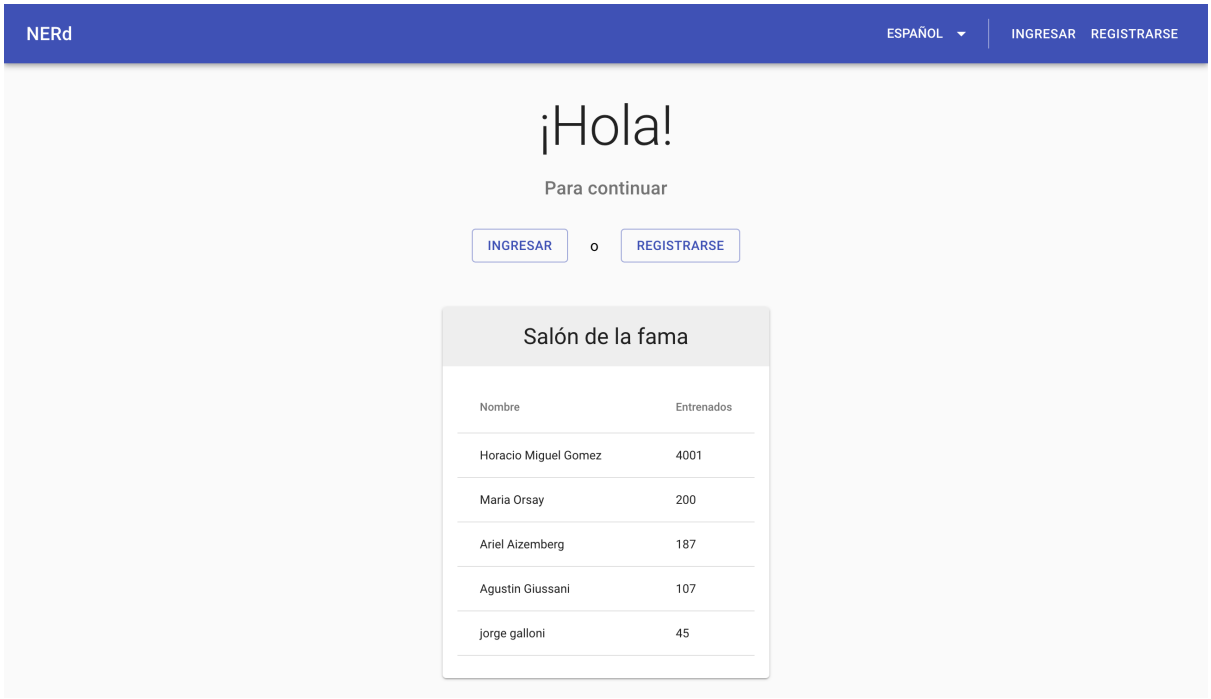


Figura 16: Pantalla de inicio sin sesión.

4.1.1.2. Entrenamiento

La pantalla de *Entrenamiento* es el núcleo de la web en la cual es posible entrenar el modelo.

El usuario es presentado con un texto perteneciente al Corpus del servicio con las entidades inferidas por el modelo actual. Con un editor especial, le permitimos al usuario poder corregir las entidades inferidas y enviarle la corrección al servicio. Esa corrección será utilizada posteriormente a la hora de mejorar el modelo actual (Figura 17).

Figura 17: Pantalla de entrenamiento.

4.1.1.2.1. Usabilidad

Tuvimos un foco fuerte en la usabilidad del widget ya que los entrenadores del servicio van a pasar prácticamente todo su tiempo en ésta pantalla, por lo que se tuvieron las siguientes consideraciones en la implementación.

Llamado a acción y ayuda

Dado que lo primero que ve el usuario es un texto con anotaciones, agregamos un título que invita al usuario a realizar acciones sobre el texto. De esta manera, le mostramos las dos acciones principales realizables desde el widget de entrenamiento: Click en alguna palabra o entidad y arrastrar un conjunto de palabras para crear una entidad nueva. Como refuerzo de este llamado a acción, agregamos un botón que al ser clickeado muestra un mensaje de ayuda con instrucciones más detalladas sobre el objetivo del entrenador y las acciones que deben de realizarse sobre el mismo (Figura 18).

Figura 18: Ayuda del entrenador.

Creación y edición de entidades

Para la creación de entidades decidimos ofrecer dos maneras: La primera es arrastrando un conjunto de palabras de manera tal de unirlas todas en una única entidad. La otra es hacer click en una palabra y ahí se ofrecen opciones dependiendo de la ubicación de la palabra dentro del texto:

- Si no existen entidades en el texto actual, se le asigna por defecto el tipo *MISC* y se muestran el resto de los tipos para permitir cambiarlo de ser necesario.
- Si existen entidades antes o después, se ofrece la opción de unir la palabra actual con la entidada más próxima para el lado elegido.

Figura 19: Edición de entidad.

Como podemos ver en la figura 19, para la edición de entidades decidimos permitir únicamente la modificación del tipo de una entidad inferida. Si el modelo infirió una entidad de manera incorrecta, ya sea por que sea una entidad inválida o agregó palabras de más a una entidad inválida, permitimos que el usuario remueva la entidad y que después vuelva a agregar la entidad correcta.

Optimización en tiempos de carga

Dado que es esperado que un usuario entrene más de un texto, al momento de pedir un texto para mostrar, se pide el siguiente. Mediante este mecanismo de pre-carga, podemos eliminar el tiempo de espera entre texto y texto ofreciendo al usuario una experiencia completamente fluida.

4.1.1.3. Administración de usuarios

La pantalla de *Administración de usuarios* permite a los usuarios con el rol de administrador poder modificar los roles de todos los usuarios del sistema, borrarlos o acceder a los detalles del usuario, tal como la lista de textos entrenados (Figura 20).

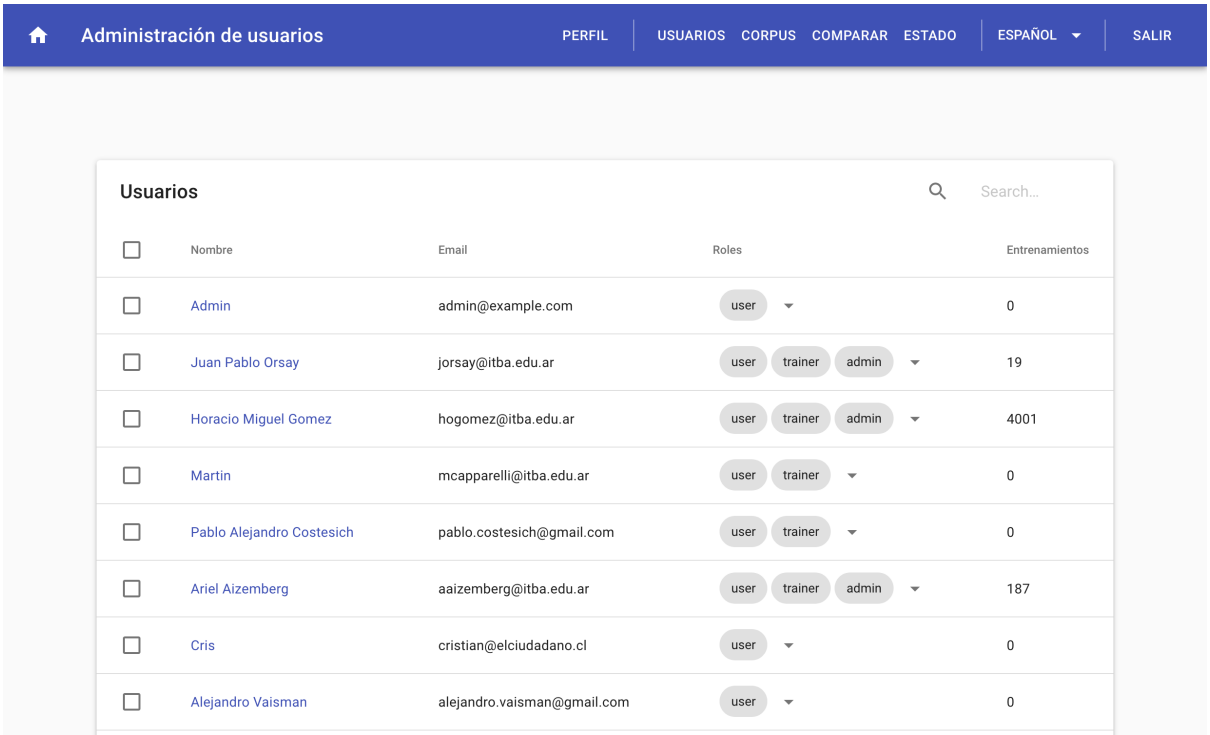


Figura 20: Administración de usuarios.

4.1.1.4. Detalles de usuario

La pantalla de detalle de usuario permite al usuario con sesión activa ver sus entrenamientos y cambiar su contraseña.

Los usuarios con rol administrador pueden realizar las acciones mencionadas previamente pero a otros usuarios (Figura 21).

Figura 21: Perfil de usuario.

4.1.1.5. Corpus

La pantalla de *Corpus* permite a un usuario con el rol de administrador realizar tareas relacionadas con el corpus del sistema (Figura 22).

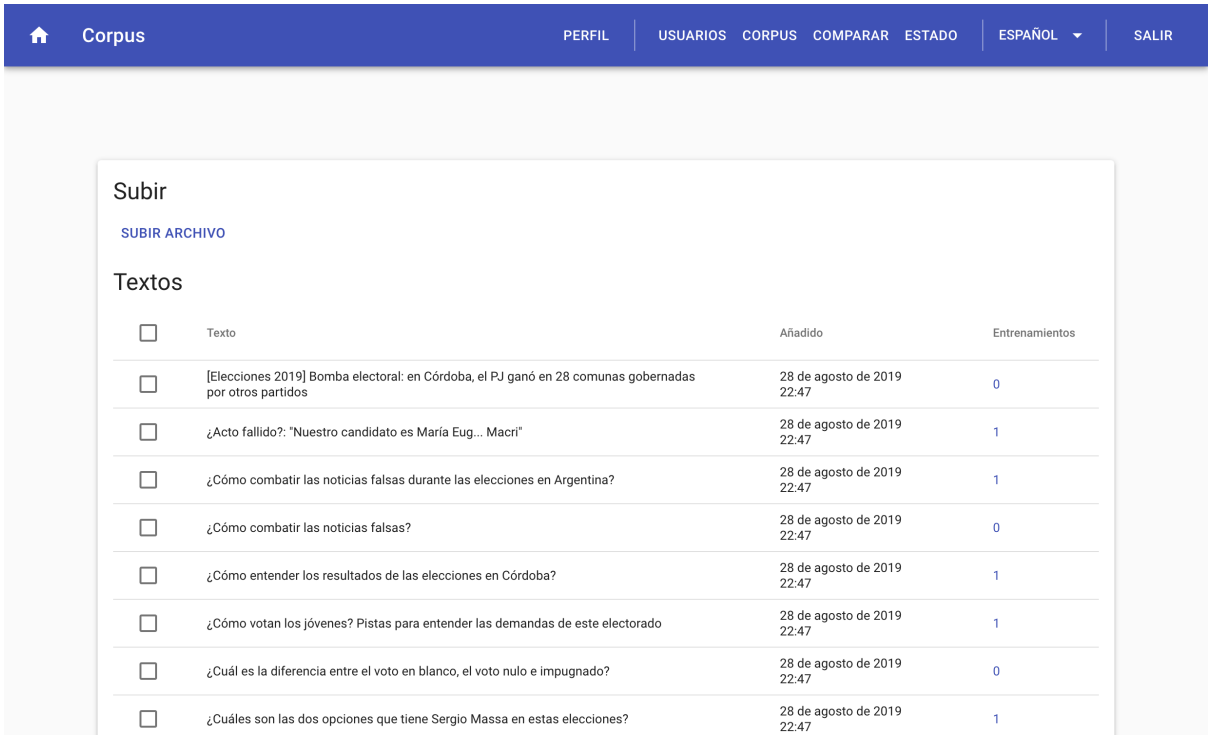


Figura 22: Administración de corpus.

Desde aquí es posible agregar textos al corpus utilizando la funcionalidad de subida de archivos. Los archivos deben ser archivos con extensión *.txt* y cada línea del archivo será agregada al corpus como un texto individual.

También es posible desde aquí ver todos los textos que forman parte del corpus así como también poder ver los entrenamientos para cada uno de los textos. Finalmente, es posible quitar textos del corpus así como también es posible eliminar correcciones a las inferencias de entidades cargados por usuarios.

4.1.1.6. Estado

La pantalla de *Estado* permite a un usuario con el rol de administrador visualizar el estado de entrenamiento del corpus así como también realizar diversas acciones sobre los *workers* (Figura 23).

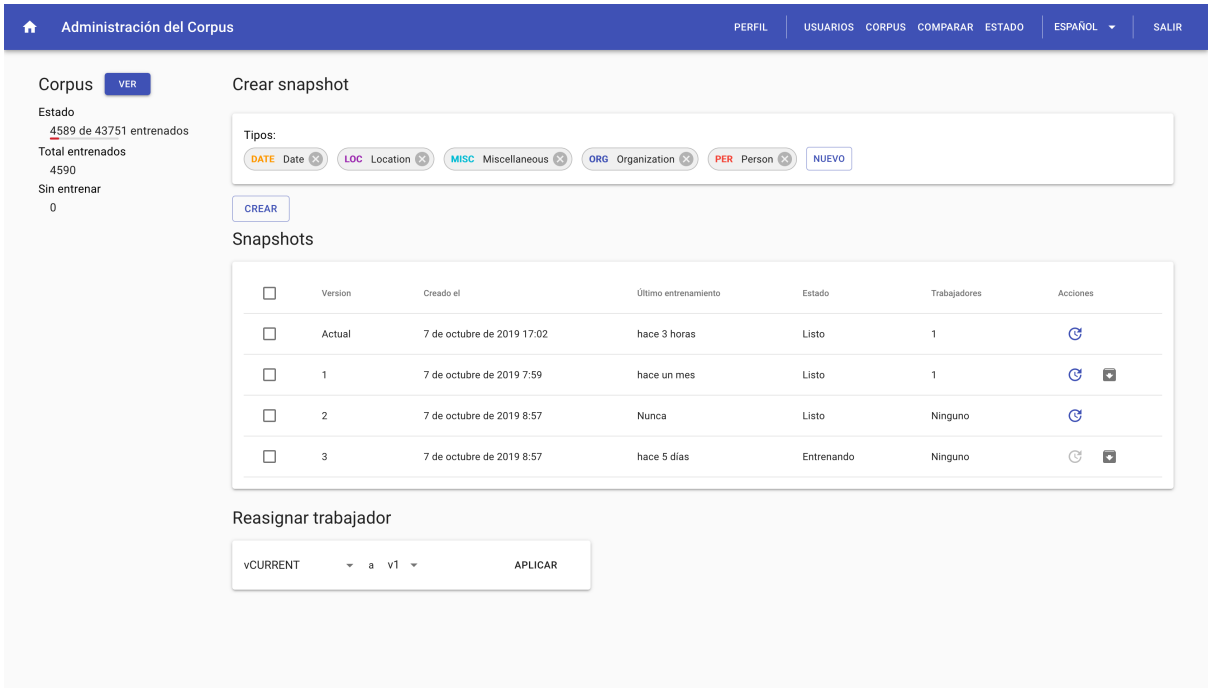


Figura 23: Información de corpus y manejo de workers.

4.1.1.6.1. Secciones

Corpus

Es la columna la izquierda y aquí se puede ver rápidamente que porcentaje de el corpus contiene correcciones por usuarios así como también saber la cantidad total de correcciones del sistema (un texto puede tener más de una corrección por distintos usuarios) y también presenta un botón que permite al administrador ir a la pantalla de *Corpus*.

Crear snapshot

Es la sección en la cual será posible crear, borrar o modificar los tipos de entidades reconocidos por el snapshot actual. La acción de editar las entidades genera un snapshot nuevo.

Si el administrador así lo quisiera, puede utilizar esta sección para crear un snapshot nuevo sin editar entidades.

Snapshots

Sección en la cual podemos ver la lista completa de snapshots. Para cada Snapshot, se muestra cuando fue la última vez que se entrenó así como también cuantos trabajadores tiene asignados. Finalmente es posible desde aquí forzar a entrenar el modelo para ese snapshot en particular y también se presenta la opción para desentrenar, borrando el modelo guardado en el disco.

Reasignar trabajador

Sección que permite reasignar trabajadores para que sirvan un snapshot distinto. De esta manera se pueden servir distintas versiones del modelo de inferencia para poder realizar distintas pruebas sobre los mismos.

4.1.1.7. Sandbox

La pantalla de *Sandbox* permite a los usuarios hacer consultas al servicio NERd para poder obtener entidades nombradas a partir de textos arbitrarios. Adicionalmente, si el usuario tiene el rol de entrenador, podrá corregir las entidades inferidas y agregar el texto con sus correcciones al corpus (Figura 24).

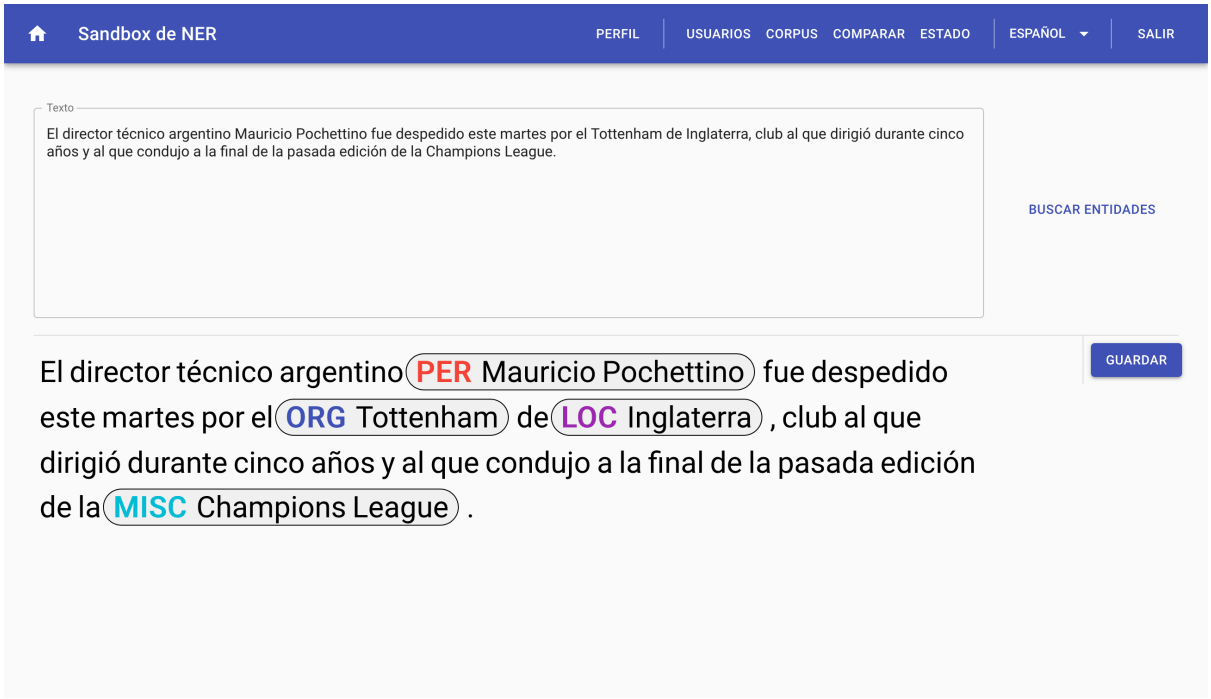


Figura 24: Inferencia de entidades en sandbox.

4.1.1.8. Comparar

Sección accesible únicamente a administradores en la que es posible comparar las entidades inferidas por dos modelos distintos. A su vez, si el usuario logueado tiene el permiso de entrenador, es posible corregir de manera inline los errores en la inferencia del modelo actual (Figura 25).

Figura 25: Comparativa de modelos.

4.1.1.9. Registración

Sección únicamente accesible cuando no hay una sesión activa. Aquí se registran los usuarios con la opción de que el inicio de sesión persista luego de que se cierre la pestaña del navegador (Figura 26).

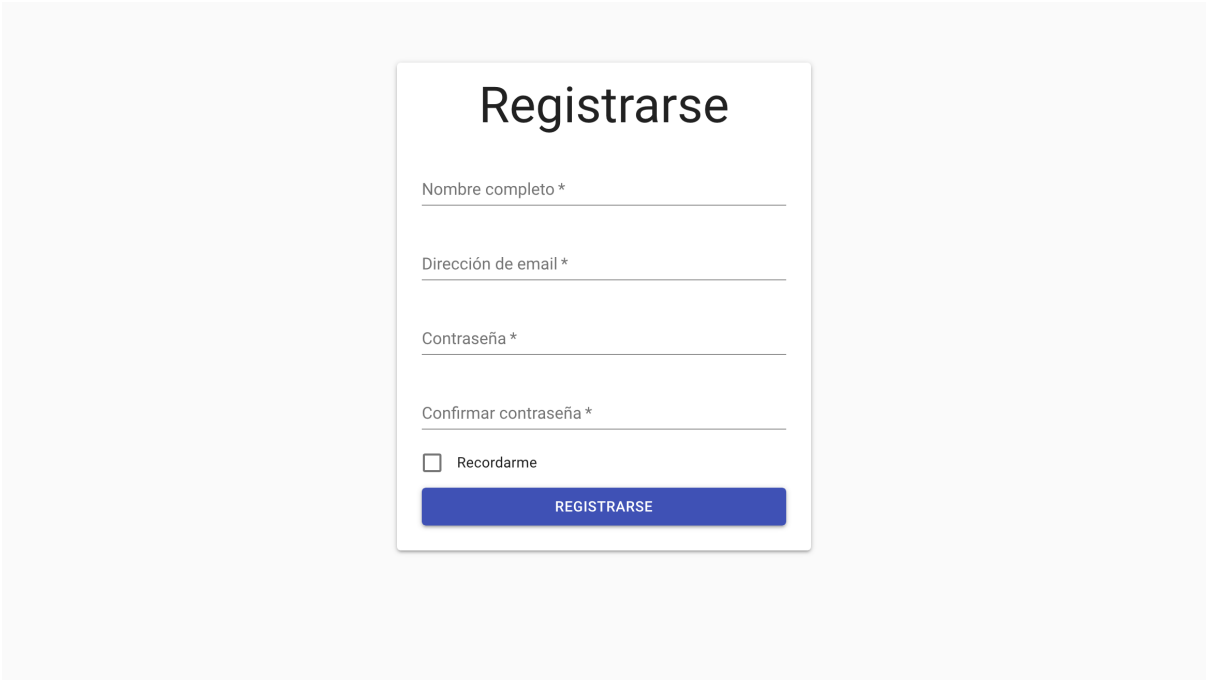


Figura 26: Registración.

4.1.1.10. Login

Sección únicamente accesible cuando no hay una sesión activa. Aquí ingresan los usuarios al sistema con la opción de que el inicio de sesión persista luego de que se cierre la pestaña del navegador (Figura 27).

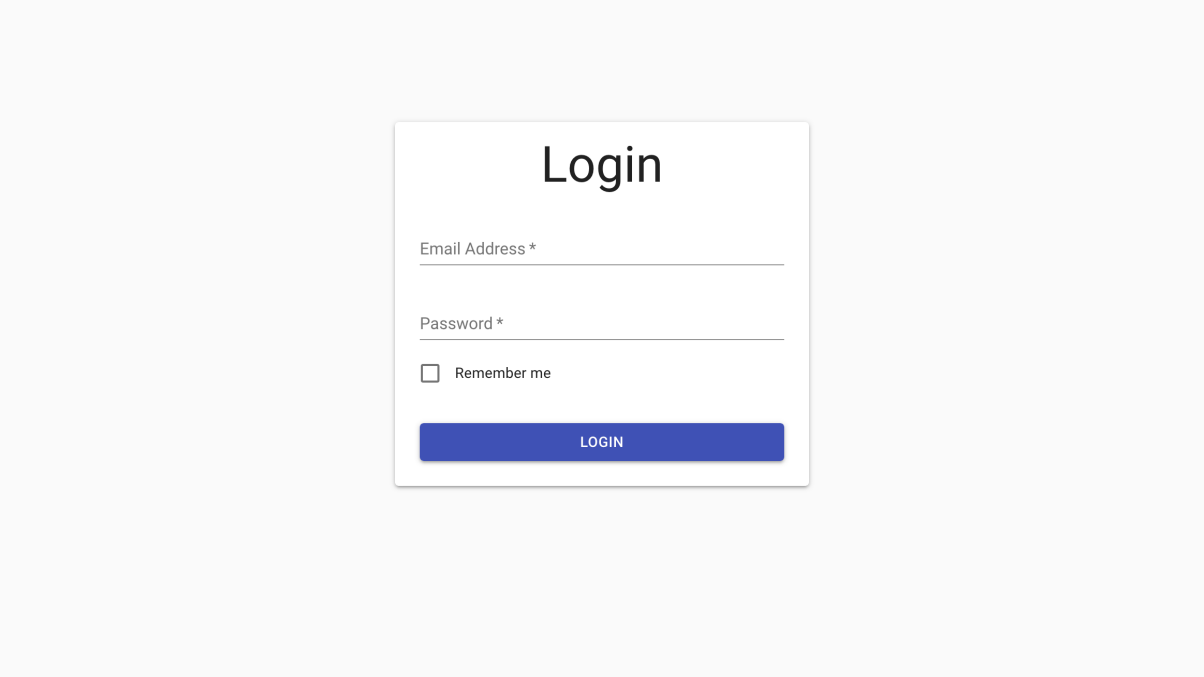


Figura 27: Inicio de sesión.

4.1.1.11. Barra de navegación

La barra de navegación muestra distinta información dependiendo del contexto actual del sistema.

Cuando no hay una sesión activa, se muestran únicamente las opciones para acceder al sistema, ya sea registrándose o ingresando utilizando credenciales (Figura 28).

Figura 28: Usuario anónimo.

Cuando hay una sesión activa el usuario cuenta con el rol de *USER* y/o de *TRAINER* pero **no** con el de *ADMIN*, debería poder acceder únicamente a su perfil (Figura 29).

Figura 29: Usuario regular.

Finalmente, cuando el usuario logueado tiene el rol de *ADMIN*, tiene acceso a todas las secciones de administración del sistema (Figura **??**fig:logic-status-admin)).

Figura 30: Usuario administrador.

4.1.2. Servicio

El acceso al servicio se realiza mediante un API REST que se auto-documenta debido a implementar la especificación de OpenAPI. A continuación detallamos los endpoints.

4.1.2.1. Autenticación

Rutas del API dedicadas a la autenticación de usuarios.

- **POST** */api/auth/register*
  - Registrar un usuario nuevo.
- **POST** */api/auth/token*
  - Genera un nuevo token de acceso y refresco con credenciales.
  - Utilizado para la funcionalidad de *login*
- **POST** */api/auth/refresh*
  - Refresca el token de acceso.
  - Utilizado cuando un token de acceso caducó.

4.1.2.2. Usuarios

Conjunto de operaciones relacionadas con los usuarios del sistema.

- **GET** */api/users*
  - Lista de usuarios existentes
  - Separa los resultados en páginas
- **GET** */api/users/top5*
  - Lista de los 5 usuarios con más entrenamientos
- **GET** */api/users/me*
  - Retorna la información del usuario logueado
- **PATCH** */api/users/me*
  - Actualiza la información del usuario logueado
- **GET** */api/users/me/trainings*
  - Retorna los entrenamientos del usuario logueado
  - Separa los resultados en páginas
- **GET** */api/users/{user_id}*
  - Retorna la información del usuario especificado por *user_id*
- **PATCH** */api/users/{user_id}*
  - Actualiza la información del usuario especificado por *user_id*
- **DELETE** */api/users/{user_id}*
  - Borra al usuario especificado por *user_id*
- **GET** _/api/users/{user_id}/trainings
  - Retorna los entrenamientos del usuario especificado

4.1.2.3. Roles

- **GET** */api/roles*
  - Retorna la lista de todos los roles asignables a usuarios del sistema

4.1.2.4. Corpus

Rutas dedicadas a operaciones con el corpus del sistema.

- **GET** */api/corpus/{text_id}*
  - Retorna los detalles del texto especificado por *text_id*
- **DELETE** */api/corpus/{text_id}*
  - Borra un texto especificado por *text_id* del corpus
- **GET** */api/corpus/{text_id}/trainings*
  - Retorna la lista de entrenamientos proporcionados por los usuarios sobre las entidades en el texto
- **PUT** */api/corpus/{text_id}/trainings*
  - Agrega un entrenamiento para el texto con id *text_id*
- **POST** */api/corpus/upload*
  - Permite agregar textos de manera masiva al sistema
  - Acepta una lista de archivos .txt donde cada línea es un texto a agregar
  - Los archivos deben ser UTF-8
- **GET** */api/corpus*
  - Lista de textos cargados en el sistema para entrenamiento
  - Separa los resultados en páginas
- **POST** */api/corpus*
  - Agrega un texto al sistema para entrenamiento

4.1.2.5. Snapshots

Conjunto de operaciones relacionadas con los snapshots y workers.

- **GET** */api/snapshots*
  - Listado de los snapshots disponibles
  - Separa los resultados en páginas
- **GET** */api/snapshots/{snapshot_id}*
  - Retorna información (tipos de entidades, fecha de creación, fecha de entrenamiento, etc.) sobre un snapshot específico
- **DELETE** */api/snapshots/{snapshot_id}*
  - Borra un snapshot con el id especificado
- **POST** */api/snapshots/{snapshot_id}/force-train*
  - Envía la tarea de entrenamiento a los workers que tienen el snapshot *snapshot_id* cargado.
- **POST** */api/snapshots/{snapshot_id}/force-untrain*
  - Envía la tarea de desentrenar a los workers que tienen el snapshot *snapshot_id* cargado.
- **GET** */api/snapshots/current*
  - Retorna información sobre el snapshot actual
- **PUT** */api/snapshots/current*
  - Crea un nuevo snapshot con la información provista

4.1.2.6. Reconocimiento de Entidades Nombradas

Conjunto de operaciones relacionadas al *Reconocimiento de Entidades Nombradas*

- **GET** */api/ner/train*
  - Retorna un texto para que un usuario del sistema revise si está correctamente inferido
  - Únicamente retorna textos que el usuario logueado no haya corregido ya
- **GET** */api/ner/compare/{first_snapshot}/{second_snapshot}*
  - Compara el *Reconocimiento de Entidades Nombradas* entre dos snapshots distintos
- **POST** */api/ner/current/parse*
  - Retorna un documento *spaCy* para un texto dado utilizando el snapshot actual
- **POST** */api/ner/{snapshot_id}/parse*
  - Retorna un documento *spaCy* para un texto dado utilizando el snapshot especificado
- **POST** */api/ner/current/entities*
  - Retorna la lista de *Entidades Nombradas* para un texto dado utilizando el modelo actual
- **POST** */api/ner/{snapshot_id}/entities*
  - Retorna la lista de *Entidades Nombradas* para un texto dado utilizando el modelo especificado

4.1.2.7. Entrenamientos

- **DELETE** */api/trainings/{training_id}*
  - Borra un entrenamiento

4.1.2.8. Workers

- **GET** */api/workers/*
  - Lista de los workers disponibles
- **POST** */api/workers/reassign*
  - Reasigna un trabajador de un a versión de snapshot a otra

4.2. Vista de proceso

> La vista de proceso trata los aspectos dinámicos del sistema, explica los procesos del sistema y cómo se comunican, y se centra en el comportamiento del sistema en tiempo de ejecución. La vista de proceso aborda concurrencia, distribución, integradores, rendimiento y escalabilidad, etc.

Como podemos ver en la figura 31, existen cinco procesos que juntos forman la totalidad de NERd y su entrenador.

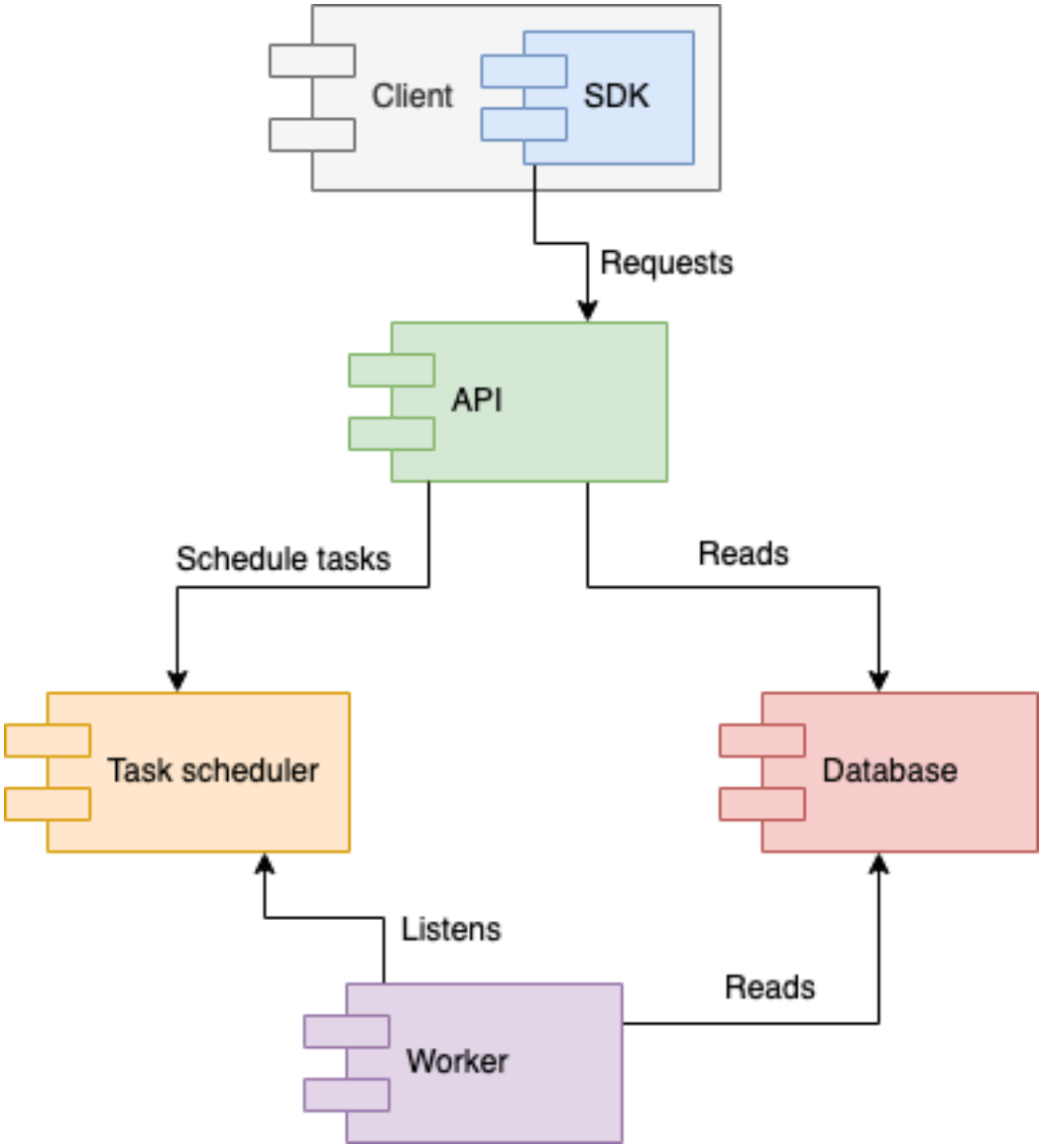


Figura 31: Componentes del sistema.

4.2.1. API

Servicio en el cual los clientes pueden pedir las *Entidades Nombradas* de textos así como también permite la carga de textos con sus respectivas correcciones que luego serán utilizadas para entrenar el modelo de inferencia. El *API* se comunica el servicio de *Bases de Datos* para obtener o modificar información sobre los usuarios, los textos pertenecientes al corpus o información de los distintos *snapshots* y con el *Task Scheduler* para realizar diversas tareas relacionadas con *spaCy*.

4.2.2. Task Scheduler

Recibe tareas desde el *API* tales como reconocer entidades, entrenar y desentrenar un modelo, cambiar el modelo de un worker entre otras. Dado que su tarea es recibir tareas desde el *API* y enviar y recibir mensajes de los *Workers*, debería existir una única instancia del mismo. A su vez, notifica al *API* la finalización de tareas asincrónicas

4.2.3. Worker

Un *Worker* se encarga de realizar el *Reconocimiento de Entidades Nombradas* así como también de entrenar modelos de inferencia. Es un servicio que existe de manera independiente y necesita de dos otros servicios para funcionar:

- *Base de datos* para obtener los datos necesarios para entrenar un modelo.
- *Task scheduler* para poder recibir tareas.

Como el *Worker* es una unidad de trabajo que recibe tareas desde el *Task Scheduler*, es posible tener varios *Workers* donde cada uno utiliza un modelo de inferencia distinto (generados a partir de distintos snapshots).

4.2.4. Database

Servicio que contiene la base de datos con la información necesaria (usuarios, snapshots) para que el servicio de API funcione así como también los datos necesarios por los workers (entrenamientos).

4.2.5. Cliente

Es la interfaz del entrenador que se comunica con el API utilizando un SDK auto-generado a partir de la definición de *OpenAPI*. Implementado en este proyecto es un cliente web el cual permite explotar las funcionalidades de entrenamiento así como de inferencia de entidades

4.3. Vista de desarrollo

> La vista de desarrollo ilustra un sistema desde la perspectiva de un programador y se ocupa de la gestión de software. Esta vista también se conoce como la vista de implementación.

4.3.1. Web

La página web que contiene la interfaz de administración así como también la de entrenamiento. Para implementarla decidimos utilizar el modelo de *Single Page Application* que nos permitió generar un *bundle* que puede ser hosteado en *CDNs* y nos permite tener ciclos de release independientes al los del servicio.

Poder separar las releases de la web y el servicio es importante ya que los clientes del servicio no necesariamente necesitan de la web, ya que pueden acceder al servicio mediante el *API REST* y una baja por cambios a la interfaz de la web no afecta la disponibilidad del servicio.

4.3.1.1. Tecnologías

Para la implementación utilizamos el lenguaje de programación llamado **Typescript**, lenguaje desarrollado por *Microsoft* que compila a *JavaScript* con el agregado de *Tipos* que permiten producir

código de mayor calidad debido al agregado de la posibilidad del chequeo estático de código para validar su correctitud.

Una vez definido el lenguaje, optamos por utilizar la librería llamada **ReactJS** (Facebook, 2019) desarrollada por *Facebook* para implementar la interfaz.

Sumado a **ReactJS**, incluímos diversas librerías para poder resolver varios problemas en el desarrollo de una *SPA*:

4.3.1.1.1. Material UI

Provee de una biblioteca de componentes estilados siguiendo el patrón de diseño propuesto por *Google* llamado *Material Design*.

4.3.1.1.2. React Router

Desarrollada por *React Training*, proveé de mecanimos para la navegación en *SPAs*.

4.3.1.1.3. i18next

Librería que proveé mecanismos de internacionalización. A su vez, nos permitió generar los archivos de traducción mediante su extracción del código fuente utilizando la herramienta `i18next-scanner`.

4.3.1.1.4. OpenAPI Generator

OpenAPI Generator es una herramienta que permite generar código a partir de una definición de *OpenAPI*. Debido a que el servicio implementado expone una definiciión de *OpenAPI*, podemos utilizar ésta librería para auto-generar todo el código necesario para comunicarnos con el servicio usando el generador de *Typescript*.

4.3.1.2. Arquitectura

El código en la Web se encuentra separado por rasgos, ya sean de utilidad, como modelos o funciones, o de funcionalidad, como la vista de la pantalla de inicio (Figura 32).

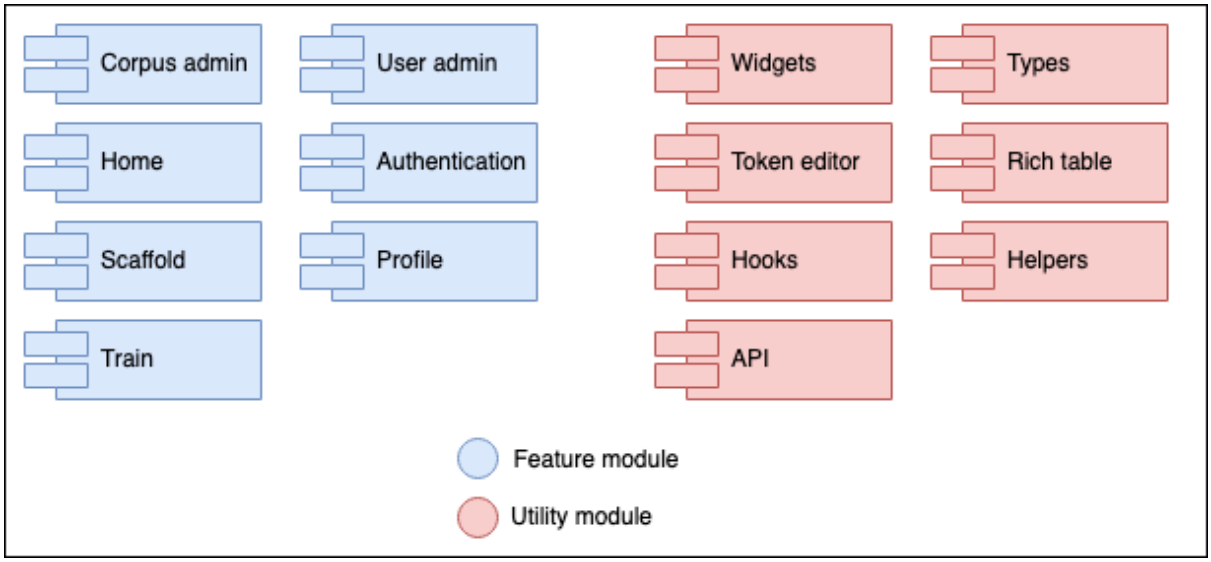


Figura 32: Módulos de la página web.

Los módulos marcados en azul son relacionados al contenido que se muestra cada ruta del sistema.

Los módulos marcados en rojos son de utilidad, ya sea por que contienen todo el código relevante a la comunicación con el *API REST* o a componentes utilizados en varias áreas de los módulos azules.

4.3.1.3. Sesión

El servicio web ofrece la posibilidad de recordar la sesión de un usuario en el momento de realizar el login. De elegirse, las credenciales son guardadas en el *local storage* del cliente web, de manera tal que el usuario pueda utilizar la misma sesión por más de que haya cerrado el cliente web.

Si en algún momento en el futuro el *API* nos retorna algún request indicando que el usuario no tiene permisos para acceder a algún área (código HTTP 401) y el token de refrescado dejó de ser válido, las credenciales se borran del *local storage* y redirigimos al cliente a la pantalla de inicio.

Ofrecemos también la posibilidad de no guardar la sesión, lo cual hará que una vez cerrada la pestaña, el usuario deba volver a ingresar al sistema para poder acceder a los distintos servicios.

4.3.1.4. Internacionalización

Así como las credenciales, las preferencias de internacionalización son guardadas en el *local storage* del cliente web, de manera tal que los usuarios del sistema no deban elegir el idioma de su preferencia cada vez que ingresan a la página.

4.3.2. Servicio

El servicio que provee el *API REST* fue implementado utilizando el lenguaje de programación *Python*. El principal motivador de esta elección fue que las librerías que más se acercan al estado del arte en *Machine Learning* se encuentran implementadas en este lenguaje.

4.3.2.1. Tecnologías

4.3.2.1.1. *Flask*

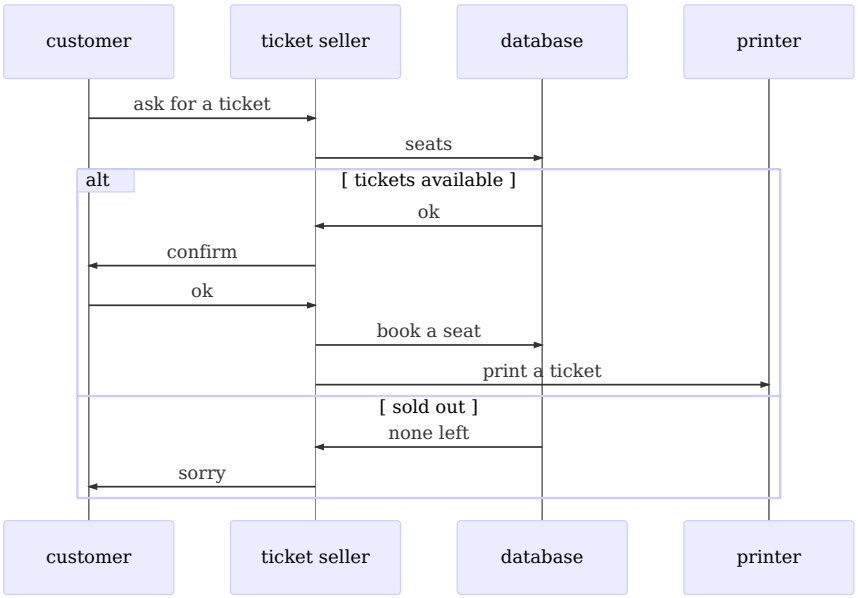
Como base del proyecto utilizamos un *microframework* de aplicaciones web llamado *Flask* (Pallets, 2019) ya que por diseño utiliza pocos recursos. Por defecto no incluye ninguna solución para comunicarse con una base de datos, ni lógica para validar formularios ni una librería para poder tener vistas que utilicen *templates*.

4.4. Vista física

La vista física representa el sistema desde el punto de vista de un ingeniero de sistemas. Se refiere a la topología de los componentes de software en la capa física, así como a las conexiones físicas entre estos componentes. Esta vista también se conoce como la vista de *deployment*.

4.5. Escenarios

La descripción de una arquitectura se ilustra utilizando un pequeño conjunto de casos de uso, o escenarios, que se convierten en una quinta vista. Los escenarios describen secuencias de interacciones entre objetos y entre procesos. Se utilizan para identificar elementos arquitectónicos y para ilustrar y validar el diseño de la arquitectura. También sirven como punto de partida para las pruebas de un prototipo de arquitectura. Esta vista también se conoce como vista de caso de uso.



5. Resultados

5.1. Métrica: precisión y exhaustividad

("Precision and recall," 2019)

> TODO: reescribir
>
> En conferencias académicas como CoNLL, se ha definido una variante del [[valor-F]] de la siguiente manera:
>
> - Precisión es el número de entidades nombradas que coinciden exactamente con conjunto de evaluación. I.e. cuando se predice [Persona Hans] [Persona Blick], pero lo correcto era [Person Hans Blick], la precisión es cero. La precisión es después promediada por cada una de la entidades nombradas.
> - El recobrado es el número de entidades del conjunto de evaluación que aparecen exactamente en la misma posición en las predicciones.
> - El Valor-F es la media armónica de estos dos valores. Se deriva de la anterior definición que cualquier predicción que reconozca erróneamente un token como parte de una entidad nombrada o que deje de detectar un token que sí es una entidad nombrada o lo clasifique erróneamente no contribuirá ni a la precisión ni al recobrado.

=== Evaluación formal === Para evaluar la calidad de la salida de un sistema NER, se han definido varias medidas. Las medidas habituales se llaman [[Precision_and_recall | Precisión, recuperación]] y [[Puntuación F1]]. Sin embargo, quedan varios problemas sobre cómo calcular esos valores.

Estas medidas estadísticas funcionan razonablemente bien para los casos obvios de encontrar o faltar una entidad real exactamente; y para encontrar una no entidad. Sin embargo, NER puede fallar de muchas otras maneras, muchas de las cuales son posiblemente «parcialmente correctas», y no deben considerarse como éxitos o fracasos competitivos. Por ejemplo, identificar una entidad real, pero: * con menos tokens de los deseados (por ejemplo, falta el último token de «John Smith, M.D.») * con más tokens de los deseados (por ejemplo, incluyendo la primera palabra de «The University of MD») * particionando entidades adyacentes de manera diferente (por ejemplo, tratando a «Smith, Jones Robinson» como entidades 2 vs. 3) * asignarle un tipo completamente incorrecto (por ejemplo, llamar a un nombre personal a una organización) * asignándole un tipo relacionado pero inexacto (por ejemplo, «sustancia» vs. «droga», o «escuela» vs. «organización») * identificar correctamente una entidad, cuando lo que el usuario quería era una entidad de menor o mayor alcance (por ejemplo, identificar «James Madison» como un nombre personal, cuando forma parte de la «Universidad James Madison». Algunos sistemas NER imponen la restricción que las entidades nunca se superpongan o aniden, lo que significa que en algunos casos uno debe tomar decisiones arbitrarias o específicas de la tarea.

Un método demasiado simple para medir la precisión es simplemente contar qué fracción de todas las fichas en el texto se identificaron correcta o incorrectamente como parte de referencias de entidad (o como entidades del tipo correcto). Esto tiene al menos dos problemas: en primer lugar, la gran mayoría de los tokens en el texto del mundo real no forman parte de los nombres de las entidades, por lo que la precisión de la línea de base (siempre predice «no una entidad») es extravagantemente alta, típicamente> 90 %; y segundo, predecir erróneamente el lapso completo del

nombre de una entidad no se penaliza adecuadamente (encontrar solo el nombre de una persona cuando le sigue su apellido podría calificarse como ½ precisión).

En conferencias académicas como CoNLL, una variante de la [[puntuación F1]] se ha definido de la siguiente manera: {{r | conll03intro}}

- [[Precisión y recuperación | Precisión]] es el número de intervalos de nombre de entidad pronosticados que se alinean " exactamente " con intervalos en los datos de evaluación [[Verdad fundamental # Estadísticas y aprendizaje automático | estándar de oro]]. Es decir. cuando se predice [Persona Hans] [Persona Blick] pero se requiere [Persona Hans Blick], la precisión del nombre predicho es cero. La precisión se promedia sobre todos los nombres de entidad pronosticados.
- Recordar es igualmente el número de nombres en el estándar de oro que aparecen exactamente en la misma ubicación en las predicciones.
- La puntuación F1 es la [[media armónica]] de estos dos.

De la definición anterior se deduce que cualquier predicción que omita un solo token, incluye un token espurio o tiene clase incorrecta, es un error difícil y no contribuye positivamente ni a la precisión ni a la recuperación. Por lo tanto, se puede decir que esta medida es pesimista: puede darse el caso de que muchos «errores» estén cerca de ser correctos, y podrían ser adecuados para un propósito dado. Por ejemplo, un sistema siempre puede omitir títulos como «Sra.» o «Ph.D.», pero se compara con un sistema o datos de verdad que esperan que se incluyan títulos. En esos casos, cada nombre se trata como un error. Debido a tales problemas, es importante examinar los tipos de errores y decidir qué tan importantes se les dan los objetivos y requisitos.

5.2. Resultados obtenidos

{r fig-train, fig.cap=«TODO figure», out.width=«80 %», fig.asp=.75, fig.align=«center»} ar(mar = c(66.238, 4, .1, .1)) plot(pressure, type = «b», pch = 19)

```
## Parsed with column specification:
## cols(
##   Itn = col_double(),
##   NER_Loss = col_double(),
##   NER_P = col_double(),
##   NER_R = col_double(),
##   NER_F = col_double(),
##   `Token_%` = col_double(),
##   CPU_WPS = col_double(),
##   GPU_WPS = col_double(),
##   T_DOCS = col_double(),
##   V_DOCS = col_double()
## )
```

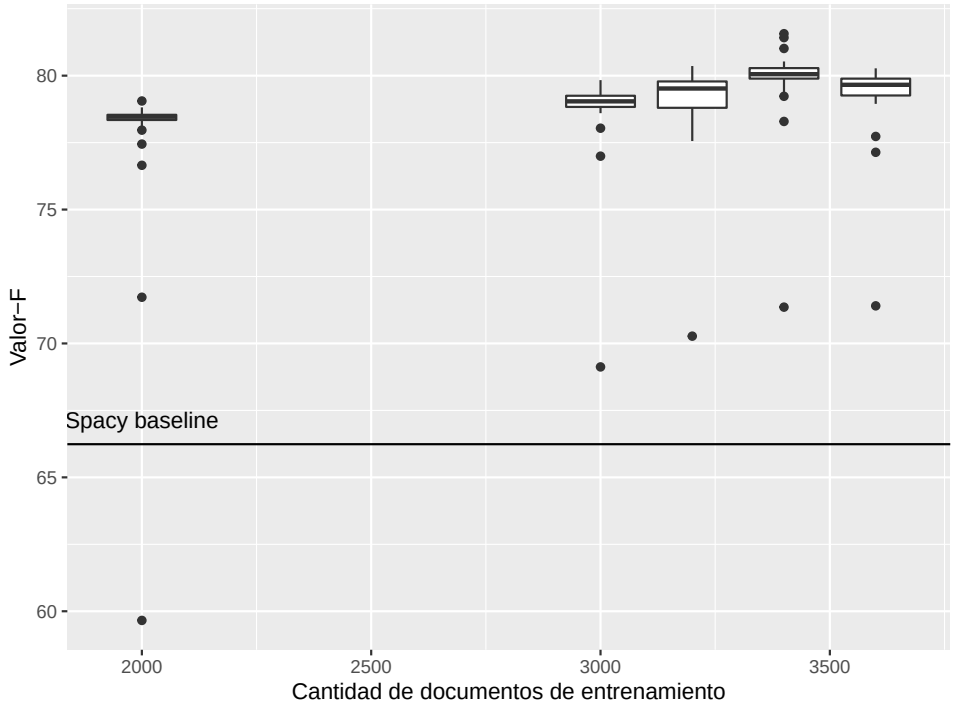


Figura 33: Boxplot del Valor-F vs vantidad de documentos de entrenamiento

6. Discusión

6.1. Tipos de entidades relevantes

tener en cuenta (Brunstein, 2002)

```
# Notas sobre mejora en tipos de entidades
Presidente -> Person Descriptor
NORP -> (Polical) Peronistas, Kirchneristas
Facility Name -> usually location. "Wall Street", "Muralla China"

Organization Name -> Government vs Corporation.
Product Name -> autos "Fiat Toro", celulares "Galaxy S10"
Events -> Superclásico. Superliga. Copa argentina. Elecciones 2019. Las Paso.
Disease ->
Game -> Football, Basket (para "titulos" no tan relevante)
```

6.2. Seed en los types

en especial para los nuevos.

6.3. Linkeo de entidades con Knowledge Base

La tarea de reconocer entidades nombradas en el texto es Reconocimiento de entidades nombradas, mientras que la tarea de determinar la identidad de las entidades nombradas mencionadas en el texto se llama Desambiguación de entidades nombradas. Ambas tareas requieren algoritmos y recursos dedicados para ser abordados. [3]

6.4. Mejora live vs offline

Mejora «Uncertainty sampling» -> buscar entidades que tengan un score ~ 0.5

6.5. Utilidad de la herramienta

Para poder poner a prueba nuestra herramienta **NERd** en un entorno real participamos de la hackaton en MediaParty 2019.

> **("Hackaton," 2019)** es un evento de tres días en Argentina, que reúne a 2500 emprendedores, periodistas, programadores de software y diseñadores de cinco continentes para trabajar juntos para el futuro de los medios de comunicación. Nacido de Hacks/Hackers Buenos Aires, el evento fusiona a grandes empresas como New York Times, The Guardian, Vox, ProPublica, Watchup, Neo4J o DocumentCloud y comunidades regionales de la mayor red de periodistas y desarrolladores del mundo.

Participamos en conjunto con otro proyecto final en el que van a utilizar nuestra API para hacer detección de entidades en documentos PDF.

La experiencia fue muy satisfactoria, recibimos buenas críticas sobre la Usabilidad de nuestra aplicación y la gran utilidad que presta a la comunidad.

Por tal motivo recibimos el primer premio de dicha hackaton ("Mención itba," 2019)

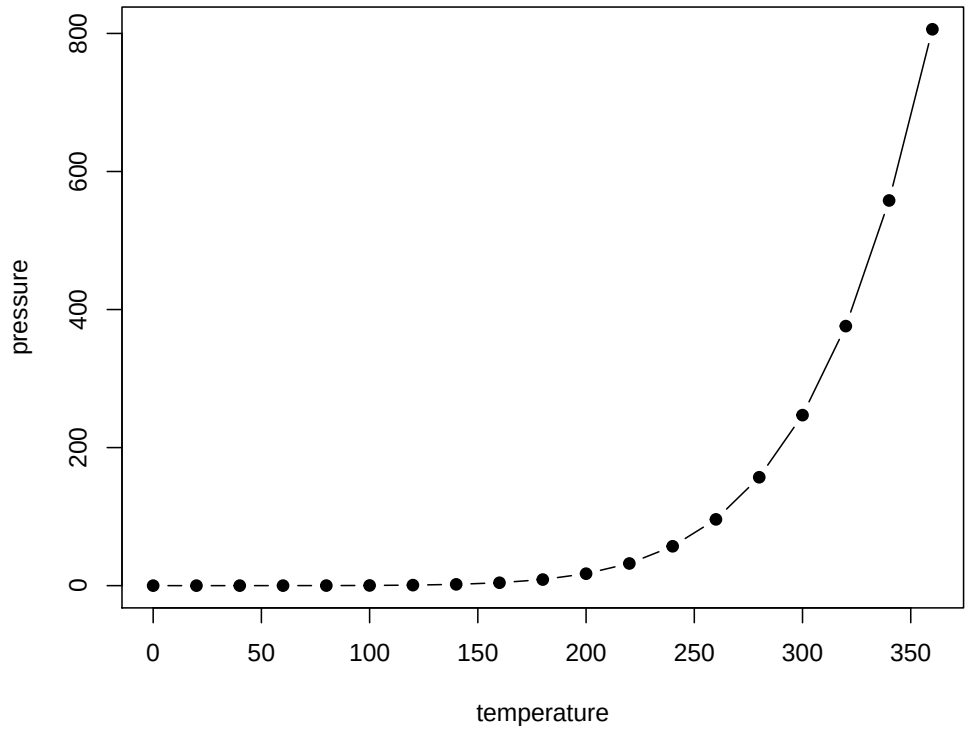


Figura 34: Here is a nice figure!

7. Conclusiones

7.1. Examples

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 1. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter 3.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 34. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2019) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

Cuadro 1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

- **t:** porcentaje de documentos de entrenamiento
- **v:** porcentaje de documentos de validación

.1. t: 0 % - v: 100 %. *Baseline spacy*

```
============================== Training stats ==============================
Training pipeline: ner
Starting with base model 'es_core_news_md'
0 training docs
4000 evaluation docs
  No overlap between training and evaluation data

============================== Vocab & Vectors ==============================
  0 total words in the data (0 unique)
  20000 vectors (533736 unique keys, 50 dimensions)

========================== Named Entity Recognition ==========================
  0 new labels, 0 existing labels
0 missing values (tokens with '-' label)
  Good amount of examples for all labels
  Examples without occurrences available for all labels
  No entities consisting of or starting/ending with whitespace

================================== Summary ==================================
  5 checks passed
Training pipeline: ['ner']
Starting with base model 'es_core_news_md'
Counting training words (limit=0)
```

Itn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	0.000	68.147	64.433	66.238	100.000	25983	26538
2	0.000	68.147	64.433	66.238	100.000	25436	25395
3	0.000	68.147	64.433	66.238	100.000	25752	24471
4	0.000	68.147	64.433	66.238	100.000	25558	29465
5	0.000	68.147	64.433	66.238	100.000	25058	24053
6	0.000	68.147	64.433	66.238	100.000	23921	25149
7	0.000	68.147	64.433	66.238	100.000	24582	25506
8	0.000	68.147	64.433	66.238	100.000	29069	28158
9	0.000	68.147	64.433	66.238	100.000	28226	28367
10	0.000	68.147	64.433	66.238	100.000	23585	23769
11	0.000	68.147	64.433	66.238	100.000	24765	23512
12	0.000	68.147	64.433	66.238	100.000	23319	25613
13	0.000	68.147	64.433	66.238	100.000	23118	24513
14	0.000	68.147	64.433	66.238	100.000	21962	28556
15	0.000	68.147	64.433	66.238	100.000	24425	24294
16	0.000	68.147	64.433	66.238	100.000	27551	23843
17	0.000	68.147	64.433	66.238	100.000	25224	24975

Itn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
18	0.000	68.147	64.433	66.238	100.000	24002	25027
19	0.000	68.147	64.433	66.238	100.000	24367	28260
20	0.000	68.147	64.433	66.238	100.000	25888	28075
21	0.000	68.147	64.433	66.238	100.000	23356	25695
22	0.000	68.147	64.433	66.238	100.000	28023	27458
23	0.000	68.147	64.433	66.238	100.000	27412	25016
24	0.000	68.147	64.433	66.238	100.000	24053	25072
25	0.000	68.147	64.433	66.238	100.000	24194	24086
26	0.000	68.147	64.433	66.238	100.000	24531	24725
27	0.000	68.147	64.433	66.238	100.000	24375	27681
28	0.000	68.147	64.433	66.238	100.000	28897	28069
29	0.000	68.147	64.433	66.238	100.000	24530	23338
30	0.000	68.147	64.433	66.238	100.000	23058	27724

.2. t: 50 % - v: 50 %

```
============================== Training stats ==============================
Training pipeline: ner
Starting with base model 'es_core_news_md'
2000 training docs
2000 evaluation docs
  1 training examples also in evaluation data

============================== Vocab & Vectors ==============================
  27595 total words in the data (6197 unique)
  20000 vectors (533736 unique keys, 50 dimensions)

========================== Named Entity Recognition ==========================
  1 new label, 4 existing labels
0 missing values (tokens with '-' label)
  Good amount of examples for all labels
  Examples without occurrences available for all labels
  No entities consisting of or starting/ending with whitespace

================================== Summary ==================================
  4 checks passed
  1 warning
Training pipeline: ['ner']
Starting with blank model 'es'
Loading vector from model 'es_core_news_md'
Counting training words (limit=0)
```

Itn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	4373.230	61.762	57.692	59.658	100.000	25083	28411
2	2650.090	72.869	70.618	71.726	100.000	21033	22844

Itn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
3	1879.892	77.506	75.820	76.653	100.000	22846	24881
4	1550.256	77.877	77.018	77.445	100.000	26894	22926
5	1331.972	78.330	78.058	78.194	100.000	25107	27973
6	1172.236	78.367	78.689	78.528	100.000	24348	28943
7	1021.166	79.104	79.004	79.054	100.000	22633	24151
8	891.760	78.931	78.657	78.794	100.000	23971	23087
9	825.860	78.731	78.657	78.694	100.000	28811	27209
10	763.561	78.715	78.815	78.765	100.000	23736	21812
11	680.819	78.754	78.878	78.816	100.000	22841	20951
12	579.623	78.504	78.405	78.454	100.000	21688	28305
13	494.455	78.553	78.405	78.479	100.000	23598	22923
14	469.631	78.470	78.247	78.358	100.000	21825	25855
15	421.725	78.592	78.468	78.530	100.000	19715	22731
16	410.767	78.439	78.562	78.501	100.000	24155	22225
17	411.734	78.284	78.531	78.407	100.000	21843	21950
18	347.546	78.220	78.689	78.454	100.000	22409	21145
19	353.033	78.119	78.562	78.340	100.000	21173	24639
20	260.769	78.238	78.657	78.447	100.000	22068	22087
21	306.896	78.332	78.752	78.541	100.000	22042	20555
22	248.995	78.538	78.909	78.723	100.000	20849	23016
23	269.101	78.300	78.720	78.510	100.000	21036	21311
24	237.393	78.228	78.499	78.363	100.000	21676	21239
25	234.766	78.439	78.562	78.501	100.000	20906	22519
26	255.452	78.612	78.562	78.587	100.000	20253	23063
27	226.127	78.432	78.531	78.481	100.000	21402	21935
28	181.469	78.327	78.499	78.413	100.000	22511	19438
29	180.268	77.904	78.026	77.965	100.000	18335	22732
30	179.335	78.023	78.121	78.072	100.000	22433	21633

.3. t: 75 % - v: 25 %

```
============================== Training stats ==============================
Training pipeline: ner
Starting with base model 'es_core_news_md'
3000 training docs
1000 evaluation docs
  No overlap between training and evaluation data

============================== Vocab & Vectors ==============================
  41847 total words in the data (7928 unique)
  20000 vectors (533736 unique keys, 50 dimensions)

========================== Named Entity Recognition ==========================
  1 new label, 4 existing labels
0 missing values (tokens with '-' label)
```

```
  Good amount of examples for all labels
  Examples without occurrences available for all labels
  No entities consisting of or starting/ending with whitespace

================================== Summary ==================================
  5 checks passed
Training pipeline: ['ner']
Starting with blank model 'es'
Loading vector from model 'es_core_news_md'
Counting training words (limit=0)
```

Itn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	5543.695	71.399	66.988	69.124	100.000	28067	25575
2	3114.421	78.567	75.483	76.994	100.000	26176	26036
3	2519.526	78.806	77.284	78.038	100.000	25530	25766
4	2120.064	79.857	78.829	79.339	100.000	25734	25163
5	1922.659	79.624	78.958	79.289	100.000	23645	25299
6	1663.921	79.807	79.858	79.833	100.000	26526	25635
7	1532.299	79.448	79.601	79.524	100.000	26120	26210
8	1311.481	79.625	79.215	79.419	100.000	25331	26163
9	1233.004	79.588	79.537	79.562	100.000	25709	25557
10	1118.481	79.486	79.537	79.511	100.000	25726	25332
11	1031.523	78.968	78.764	78.866	100.000	25399	26109
12	930.418	79.317	79.215	79.266	100.000	23443	24004
13	874.136	79.253	79.151	79.202	100.000	22272	26035
14	828.361	79.199	78.893	79.046	100.000	23957	25490
15	708.060	79.199	78.893	79.046	100.000	25963	28166
16	663.398	78.917	78.764	78.841	100.000	29668	27887
17	659.681	78.737	78.636	78.686	100.000	24656	27843
18	575.416	78.852	78.700	78.776	100.000	24306	28002
19	587.828	78.512	78.764	78.638	100.000	24038	28674
20	527.561	78.498	78.700	78.599	100.000	24642	26366
21	477.470	79.035	79.086	79.061	100.000	24744	24873
22	479.368	78.985	79.086	79.035	100.000	24633	26975
23	440.360	78.829	78.829	78.829	100.000	26582	28826
24	451.145	79.315	78.958	79.136	100.000	26343	28304
25	409.140	79.250	78.893	79.071	100.000	28135	27367
26	434.599	79.226	79.022	79.124	100.000	23172	29557
27	368.677	79.097	78.893	78.995	100.000	29263	25145
28	410.056	79.097	78.893	78.995	100.000	27527	23153
29	329.823	78.792	78.893	78.842	100.000	25775	23087
30	336.049	78.930	78.829	78.880	100.000	25315	24963

.4. t: 80 % - v: 20 %

```
============================== Training stats ==============================
```

```
Training pipeline: ner
Starting with base model 'es_core_news_md'
3200 training docs
800 evaluation docs
 No overlap between training and evaluation data

============================== Vocab & Vectors ==============================
 44669 total words in the data (8237 unique)
 20000 vectors (533736 unique keys, 50 dimensions)

========================== Named Entity Recognition ==========================
 1 new label, 4 existing labels
0 missing values (tokens with '-' label)
 Good amount of examples for all labels
 Examples without occurrences available for all labels
 No entities consisting of or starting/ending with whitespace

================================== Summary ==================================
 5 checks passed
Training pipeline: ['ner']
Starting with blank model 'es'
Loading vector from model 'es_core_news_md'
Counting training words (limit=0)
```

Itn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	5883.516	72.743	67.964	70.273	100.000	26475	26383
2	3402.834	78.329	76.805	77.559	100.000	24611	22944
3	2753.233	78.595	78.021	78.307	100.000	24175	25389
4	2412.231	79.577	79.319	79.448	100.000	25896	23811
5	2068.848	80.130	79.805	79.967	100.000	24106	24838
6	1854.721	80.506	80.049	80.277	100.000	26573	28760
7	1602.726	80.246	79.400	79.821	100.000	25482	25610
8	1400.760	80.874	79.562	80.213	100.000	24446	25178
9	1278.537	80.542	79.562	80.049	100.000	24311	22168
10	1188.572	80.574	79.724	80.147	100.000	23460	23847
11	1057.995	80.921	79.805	80.359	100.000	25690	26429
12	982.713	79.935	79.481	79.707	100.000	24790	24607
13	911.306	79.935	79.481	79.707	100.000	23366	24298
14	872.340	79.805	79.481	79.642	100.000	25026	24863
15	744.991	79.397	79.075	79.236	100.000	23702	22732
16	724.587	79.918	79.400	79.658	100.000	25589	22711
17	703.524	79.577	79.319	79.448	100.000	24541	21817
18	666.757	80.016	79.238	79.625	100.000	24072	23769
19	617.329	80.230	79.319	79.772	100.000	23397	24896
20	538.848	80.180	79.400	79.788	100.000	28608	29527
21	545.278	79.951	79.238	79.593	100.000	27668	28125
22	542.003	79.657	79.075	79.365	100.000	24713	28483
23	515.150	79.085	78.508	78.795	100.000	24505	29683

Itn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
24	510.062	78.694	78.183	78.438	100.000	28378	28694
25	490.366	78.694	78.183	78.438	100.000	29099	25309
26	489.131	79.003	78.427	78.714	100.000	28224	27445
27	481.775	79.038	78.589	78.813	100.000	27452	26523
28	434.892	79.055	78.670	78.862	100.000	29050	23274
29	418.859	79.218	78.832	79.024	100.000	22857	25335
30	384.182	78.763	78.508	78.635	100.000	27555	25168

.5. t: 85 % - v: 15 %

```
============================== Training stats ==============================
Training pipeline: ner
Starting with base model 'es_core_news_md'
3400 training docs
600 evaluation docs
  No overlap between training and evaluation data

============================== Vocab & Vectors ==============================
  47412 total words in the data (8489 unique)
  20000 vectors (533736 unique keys, 50 dimensions)

========================== Named Entity Recognition ==========================
  1 new label, 4 existing labels
0 missing values (tokens with '-' label)
  Good amount of examples for all labels
  Examples without occurrences available for all labels
  No entities consisting of or starting/ending with whitespace

================================== Summary ==================================
  5 checks passed
Training pipeline: ['ner']
Starting with blank model 'es'
Loading vector from model 'es_core_news_md'
Counting training words (limit=0)
```

Itn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	6149.150	72.957	69.824	71.356	100.000	25187	25398
2	3530.670	78.947	77.643	78.290	100.000	28946	30524
3	2782.778	80.537	79.295	79.911	100.000	28908	21788
4	2408.691	82.011	80.837	81.420	100.000	29169	24758
5	2083.651	81.748	81.388	81.567	100.000	28273	21302
6	1874.490	81.195	80.837	81.015	100.000	27418	24743
7	1733.739	80.154	80.066	80.110	100.000	25380	22917
8	1569.485	80.000	79.736	79.868	100.000	22546	23340
9	1404.922	80.132	79.956	80.044	100.000	25653	22216

Itn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
10	1270.519	79.912	79.736	79.824	100.000	23418	23172
11	1213.184	79.868	79.956	79.912	100.000	24245	24331
12	1071.570	80.375	80.286	80.331	100.000	25123	24563
13	968.863	80.353	80.176	80.265	100.000	22790	23443
14	933.663	80.331	80.066	80.199	100.000	23651	24537
15	861.770	80.509	80.066	80.287	100.000	20112	22464
16	774.950	80.442	80.176	80.309	100.000	22588	24903
17	848.828	80.642	80.286	80.464	100.000	23963	25567
18	702.390	80.266	79.736	80.000	100.000	21939	24809
19	666.903	80.577	79.956	80.265	100.000	22971	24589
20	659.185	80.663	80.396	80.530	100.000	23862	23570
21	600.204	80.288	79.846	80.066	100.000	22904	25857
22	579.303	80.221	79.956	80.088	100.000	23899	23143
23	558.429	80.288	79.846	80.066	100.000	23462	24445
24	588.920	80.177	79.736	79.956	100.000	24245	24946
25	529.718	80.266	79.736	80.000	100.000	24724	22070
26	556.021	80.155	79.626	79.890	100.000	23382	23936
27	455.335	80.155	79.626	79.890	100.000	26675	21860
28	473.464	79.535	79.185	79.360	100.000	23786	27999
29	469.124	79.272	79.185	79.229	100.000	23863	22476
30	447.699	79.383	79.295	79.339	100.000	24570	22265

.6. t: 90 % - v: 10 %

```
============================== Training stats ==============================
Training pipeline: ner
Starting with base model 'es_core_news_md'
3600 training docs
400 evaluation docs
 No overlap between training and evaluation data

============================== Vocab & Vectors ==============================
 50207 total words in the data (8769 unique)
 20000 vectors (533736 unique keys, 50 dimensions)

========================== Named Entity Recognition ==========================
 1 new label, 4 existing labels
0 missing values (tokens with '-' label)
 Good amount of examples for all labels
 Examples without occurrences available for all labels
 No entities consisting of or starting/ending with whitespace

================================ Summary =================================
 5 checks passed
Training pipeline: ['ner']
```

```
Starting with blank model 'es'
Loading vector from model 'es_core_news_md'
Counting training words (limit=0)
```

Itn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	6337.608	73.835	69.128	71.404	100.000	24730	26330
2	3733.858	80.364	74.161	77.138	100.000	26793	23166
3	3052.388	79.718	75.839	77.730	100.000	26983	23049
4	2530.293	79.897	78.020	78.947	100.000	28707	23476
5	2252.914	81.034	78.859	79.932	100.000	25129	27980
6	2018.009	81.424	78.691	80.034	100.000	27818	27073
7	1798.934	81.739	78.859	80.273	100.000	29947	27560
8	1650.571	80.763	78.188	79.454	100.000	29033	27491
9	1484.659	81.469	78.188	79.795	100.000	25644	29321
10	1423.551	81.076	78.356	79.693	100.000	29104	28378
11	1211.619	81.109	78.523	79.795	100.000	25347	30059
12	1115.638	80.345	78.188	79.252	100.000	24173	28090
13	1025.299	80.936	78.356	79.625	100.000	27853	20482
14	985.754	81.109	78.523	79.795	100.000	22511	27637
15	998.443	81.501	78.356	79.897	100.000	27517	27543
16	906.401	81.436	78.020	79.692	100.000	25237	29533
17	822.168	80.944	77.685	79.281	100.000	27192	28424
18	796.657	80.977	77.852	79.384	100.000	23619	23841
19	704.390	80.803	77.685	79.213	100.000	23415	24997
20	699.090	80.729	78.020	79.352	100.000	24897	20853
21	691.900	80.763	78.188	79.454	100.000	23200	24635
22	654.682	80.696	77.852	79.249	100.000	23883	23942
23	624.907	81.250	78.523	79.863	100.000	28663	28699
24	640.695	81.391	78.523	79.932	100.000	27891	27498
25	593.737	81.786	78.356	80.034	100.000	23063	22741
26	547.825	81.469	78.188	79.795	100.000	28517	22537
27	538.115	81.786	78.356	80.034	100.000	30210	23101
28	529.483	81.501	78.356	79.897	100.000	22598	21694
29	538.596	80.977	77.852	79.384	100.000	23760	23484
30	495.002	80.522	77.685	79.078	100.000	23161	22751

References

Brunstein, A. (2002). Annotation guidelines for answer types. Retrieved from https://catalog.ldc.upenn.edu/docs/LDC2005T33/BBN-Types-Subtypes.html

Case, K. E., & Fair, R. C. (1999). *Principles of economics*. Retrieved from https://books.google.com.ar/books?id=9vvAlOBfq0kC

Duh definition. (2019). Retrieved October 14, 2019, from https://dictionary.cambridge.org/es/diccionario/ingles/duh

Elliott, T. (2019). The state of the octoverse: Machine learning. Retrieved January 24, 2019, from https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/

Ethayarajh, K., Duvenaud, D., & Hirst, G. (2019). Towards understanding linear word analogies. *Proceedings of the 57th annual meeting of the association for computational linguistics*, 3253–3262. https://doi.org/10.18653/v1/P19-1315

Facebook. (2019). React: A javascript library for building user interfaces. Retrieved November 20, 2019, from https://reactjs.org/

Hackaton. (2019). Retrieved August 31, 2019, from https://mediaparty.info/

Honnibal, M. (2017). Practical and effective neural ner. Retrieved November 2, 2017, from https://github.com/explosion/talks/blob/master/2017-11-02_Practical-and-Effective-Neural-NER.pdf

Kripke, S. (1980). *Naming and necessity*. Retrieved from https://books.google.com.ar/books?id=9vvAlOBfq0kC

Kruchten, P. (1995). The 4+1 view model of architecture. *IEEE Softw.*, *12*(6), 42–50. https://doi.org/10.1109/52.469759

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR*, *abs/1603.01360*. Retrieved from http://arxiv.org/abs/1603.01360

Marsh, E., & Perzanowski, D. (1998). MUC-7 evaluation of IE technology: Overview of results. *Seventh message understanding conference (MUC-7): Proceedings of a conference held in fairfax, virginia, April 29 - may 1, 1998*. Retrieved from https://www.aclweb.org/anthology/M98-1002

Mención itba. (2019). Retrieved October 3, 2019, from https://www.instagram.com/p/B3Koum2peD-/

Montani, I. (2016). Practical and effective neural ner. Retrieved November 28, 2016, from https://github.com/explosion/talks/blob/master/2016-11-28_The-State-of-AI-2016.pdf

Pallets, T. (2019). Flask: Lightweight wsgi web application framework. Retrieved November 20, 2019, from https://palletsprojects.com/p/flask/

Parikh, A., Täckström, O., Das, D., & Uszkoreit, J. (2016). A decomposable attention model for natural language inference. *Proceedings of the 2016 conference on empirical methods in natural language processing*, 2249–2255. https://doi.org/10.18653/v1/D16-1244

Poibeau, T., & Kosseim, L. (2000). Proper name extraction from non-journalistic texts. *CLIN*.

Precision and recall. (2019). Retrieved October 16, 2019, from https://en.wikipedia.org/wiki/Precision_and_recall

Xie, Y. (2015). *Dynamic documents with R and knitr* (2nd ed.). Retrieved from http://yihui.name/knitr/

Xie, Y. (2019). *Bookdown: Authoring books and technical documents with r markdown*. Retrieved from https://github.com/rstudio/bookdown