INGENIERÍA EN
Sistemas de Información
PROYECTO FINAL DE CARRERA.

PROYECTO FINAL DE CARRERA

Informe de Proyecto Final

“Diseño e implementación de un agente inteligente capaz de clasificar situaciones de inseguridad mediante técnicas de Machine Learning y de Procesamiento del Lenguaje Natural”

AUTORES

CHAMORRO, Mateo Jesús – LU: 23183 - mateochamorro@hotmail.com

NUDEL, Lautaro – LU: 23215 - lautaronudel@gmail.com

SAINT MARTIN, Jean Pierre – LU: 23388 - jeansaintmartin@hotmail.com

DIRECTORES DE PROYECTO

Roa, Jorge– Director de Proyecto

Rubiolo, Mariano – Codirector de Proyecto

Universidad Tecnológica Nacional - Facultad Regional Santa Fe – Año 2020

Índice de contenidos

1. Introducción

En los tiempos que corren, la seguridad ciudadana se ha vuelto un factor fundamental para el bienestar de hogares y comunidades. Los hechos delictivos, la violencia doméstica y de género, las violaciones, los secuestros, son ejemplos de situaciones de inseguridad que producen consecuencias negativas parciales o permanentes hacia los ciudadanos.

Los costos asociados a las situaciones de inseguridad van desde lo irreparable de la vida misma hasta daños físicos, psicológicos y de integridad social para el individuo o comunidad que las padece. Para el gobierno, implican reducciones enormes tanto en cantidad como en magnitud de inversiones y de producción de las empresas, pues nadie querrá invertir en una ciudad en la que los hechos de inseguridad son moneda corriente.

Se han llevado a cabo múltiples esfuerzos tecnológicos con fines de hacer frente a estas situaciones, pero la mayoría han fracasado o no se les ha dado el provecho que realmente podrían alcanzar. Entre ellos se encuentra la implementación del botón de pánico, el cual requiere de tenerlo al alcance y de tener el tiempo suficiente para accionarlo y, así mismo, la existencia de cámaras de seguridad que permitan un monitoreo continuo de los ciudadanos y su situación. Pero la realidad no es tan sencilla. Para su correcto funcionamiento, es necesario el patrullaje de móviles y/o el control continuo de dichas cámaras (revisión manual), lo cual demanda tiempo, esfuerzo y dinero que, en la gran mayoría de los casos, no se dispone. Y, en el hipotético caso de que estos recursos si existieran, se corre un riesgo alto de irresponsabilidad por parte de los operadores, así como fallos operacionales o de rutina, que no son deseables en absoluto.

Según datos oficiales del Ministerio de Seguridad de la Nación [1], en el transcurso del año 2018 hubo 1.552.285 hechos delictivos, lo que, en comparación con años anteriores, supone un aumento en la tasa de crímenes.

Por tanto, se puede decir que es imperiosa la necesidad de mejorar la seguridad de los ciudadanos. Enfrentar esta carencia con un proyecto de índole tecnológica es un buen camino. Actualmente existen tecnologías tales como el Aprendizaje Automático (abreviado como ML, del inglés Machine Learning) y el Procesamiento del Lenguaje Natural (PLN, en adelante), que pueden ser utilizadas para detectar hechos de inseguridad en tiempo real.

El desafío entonces es poder crear una herramienta capaz de detectar situaciones de inseguridad, que la misma pueda aprender en base a un entrenamiento previo y que no comprometa a la privacidad de las personas, evitando el registro de cualquier conversación o diálogo en el cual se haya detectado o no un hecho de inseguridad.

Es de público conocimiento que la manipulación de conversaciones entre personas es un tema controversial y delicado. Naturalmente, dichas conversaciones no deben ser invadidas por ningún tipo de software. Por tanto, el prototipo desarrollado en este proyecto debe respetar esta premisa. Sin embargo, desde un punto de vista legal, sería de utilidad guardar un registro de las interacciones que, en efecto, resultan conflictivas debido a que presentan cierto nivel de inseguridad en ellas. Estos registros podrían ser utilizados como evidencia en procesos judiciales. Aun así, para los límites de realización del presente proyecto, la línea divisoria será particularmente clara: no existen intereses en invadir la privacidad de las personas.

1.1. Objetivos

1.1.1. Objetivos Generales

El presente proyecto tiene como objetivo general desarrollar e implementar una herramienta de software prototipo, basada en técnicas de ML y PLN, que permita detectar y categorizar hechos de inseguridad. El principal propósito del agente es brindar a la ciudadanía una herramienta que dé soporte a la detección en tiempo real de hechos de inseguridad y que permita decidir las acciones a tomar para mitigar el impacto de estos sucesos en la población.

1.1.2. Objetivos Específicos

1. Investigar, analizar y probar técnicas de ML y de PLN.
2. Diseñar y definir un dataset con textos de diálogos que ocurren durante potenciales hechos de inseguridad.
3. Diseñar e implementar un algoritmo computacional basado en ML.
4. Entrenar el modelo resultante mediante el dataset definido previamente, para así detectar y categorizar hechos de inseguridad, a partir de diálogos representados en formato de texto.

5. Validar y optimizar el modelo entrenado.
6. Desarrollar una herramienta de software prototipo que utilice el algoritmo desarrollado y que permita detectar en tiempo real hechos de inseguridad.

1.2. Temporalidad

El presente proyecto final de carrera se desarrolló entre los meses de Noviembre del año 2019 y Abril del año 2021.

1.3. Abreviaturas

Durante la redacción del presente informe, se hace uso de diversas abreviaturas, a saber:

- ML: Machine Learning (Aprendizaje Automático).
- PLN: Procesamiento del Lenguaje Natural.
- IA: Inteligencia Artificial.
- SVM: Support Vector Machines (Máquinas de soporte vectorial).
- NER: Named Entity Recognition (Reconocimiento de entidades nombradas).
- TEXTCAT: Text Categorizer (Clasificador de texto).
- PFC: Proyecto Final de Carrera.
- PC: Personal Computer (Computadora Personal).

1.4. Organización del informe

El corriente informe se encuentra organizado en 11 (once) secciones, a saber:

1. **Introducción**. Sección destinada a describir brevemente el tópico principal que aborda el proyecto, y la problemática que busca resolver. Además, se detallan los objetivos generales y específicos del proyecto. Finalmente, se establece un contexto espaciotemporal para el proyecto, y se ofrece una lista de las abreviaturas empleadas en el informe, con su correspondiente significado.

2. **Marco Teórico**. Desarrollo de conceptos teóricos necesarios para proveer un contexto, tal que se permita acceder a un total entendimiento del presente informe.

3. **Metodología utilizada**. Descripción de las iteraciones realizadas; objetivos, tareas y retroalimentación de cada iteración.

4. **Tecnologías utilizadas**. Explicación de las distintas herramientas y tecnologías utilizadas a lo largo del proyecto: lenguajes de programación, librerías esenciales y de soporte, entornos de desarrollo.

5. **Conformación de los datasets.** Sección empleada para describir y detallar el proceso de creación de los datasets utilizados

6. **Entrenamiento de módulos.** Explicación de los parámetros utilizados para cada uno de los módulos, el proceso bajo el cual fueron entrenados y, finalmente, la precisión obtenida en cada uno.

7. **Herramienta de software para la detección de hechos de inseguridad.** Definición de requerimientos funcionales y no funcionales para el prototipo desarrollado, exposición de la arquitectura del mismo y de su interfaz. Se ofrece además un hipervínculo al repositorio donde reside una copia del código fuente del proyecto.

8. **Casos de prueba**. Presentación de escenarios concretos para probar el funcionamiento de la herramienta en ambientes específicos.

9. **Conclusiones y trabajos futuros**. Descripción de los aportes del proyecto desde un punto de vista académico, profesional y personal. Además, se detallan aspectos que no fueron considerados en este proyecto pero que podrían ser el punto de partida de otros.

10. **Referencias**. Listado de las fuentes de información recuperadas durante el transcurso del proyecto.

11. **Anexos**. Documentación y material complementario y más específico para el sistema.

2. Marco Teórico

Se desarrolla a continuación un detallado marco teórico que sentará las bases sobre las cuales se construyó el producto ingenieril que da sustento al proyecto.

En primera instancia, se otorga una definición conceptual de la Inteligencia Artificial. Luego, se pone énfasis en el Aprendizaje Automático: concepto, clasificaciones, algoritmos y aplicaciones. Finalmente, se ofrece una explicación detallada del Procesamiento del Lenguaje Natural: concepto, algoritmos más importantes, implementaciones mediante ML.

2.1. La Inteligencia Artificial

La inteligencia artificial (IA, en adelante) es aquella inteligencia demostrada por las máquinas para accionar de manera similar al humano. Puede ser dividida en dos enfoques diferentes: el humano y el racional.

Dentro del **enfoque humano**, podemos definir la IA como:

- Sistemas que piensan como humanos: *"La automatización de actividades que se asocian con el pensamiento humano, actividades como la toma de decisiones, resolver problemas, aprender…" (Bellman, 1978) [2].*
- Sistemas que actúan como humanos: "El arte de crear máquinas que realizan funciones que requieren inteligencia al ser realizadas por personas" *(Kurzweil, 1990) [2].*

Por otro lado, dentro del **enfoque racional**, podemos definir la IA como:

- Sistemas que piensan racionalmente: *"El estudio de las computadoras que permiten percibir, razonar y actuar" (Winston, 1992) [2] .*
- Sistemas que actúan racionalmente: *"Es el estudio del diseño de los agentes inteligentes" (Poole, 1998) [2].*

En resumen, el enfoque centrado en el comportamiento humano mide el éxito en términos de la **fidelidad** en la forma de actuar de los humanos, mientras que un sistema racional hace lo correcto en función de sus **conocimientos**. El enfoque humano no debe ser una ciencia empírica, mientras que el racional implica una combinación de matemática e ingeniería.

2.2. Aprendizaje Automático

Con el fin de poder imitar el comportamiento humano, surge el aprendizaje automático**.** El ML es un subconjunto de la Inteligencia Artificial que se concentra principalmente en el diseño de sistemas, permitiendo a los mismos **aprender** y **realizar predicciones** basadas en cierta experiencia previa (modelos). Ésta rama de la IA es sobre la que se basa casi enteramente el presente Proyecto Final de Carrera (PFC, en adelante).

Existen diferentes formas de aprendizaje automático, en función del componente a mejorar, el conocimiento previo que se disponga, la representación de los datos y la disponibilidad de los mismos. Aunque un modelo de aprendizaje automático puede aplicar una combinación de diferentes técnicas, los métodos de aprendizaje generalmente se pueden clasificar en cuatro categorías:

- Aprendizaje supervisado (o Supervised Learning). El algoritmo de aprendizaje recibe datos etiquetados y la salida deseada. Por ejemplo, imágenes de perros con la etiqueta "perro" ayudarán al algoritmo a identificar las reglas para clasificar imágenes de perros.

- Aprendizaje no supervisado (o Unsupervised Learning). Los datos proporcionados al algoritmo de aprendizaje no están etiquetados, y se le pide al algoritmo que identifique patrones en los datos de entrada. Por ejemplo, el sistema de recomendación de un sitio web de comercio electrónico donde el algoritmo de aprendizaje descubre artículos similares que a menudo se compran juntos.

- Aprendizaje semi-supervisado (o Semi-supervised Learning). Es una combinación de las dos categorías anteriores. En este caso, solo algunos datos se encuentran etiquetados, mientras que el resto son pseudo-etiquetados por el modelo entrenado.

- Aprendizaje reforzado (o Reinforcement Learning). El algoritmo interactúa con un entorno dinámico que proporciona comentarios en términos de recompensas y castigos. Por ejemplo, los autos sin conductor son recompensados por permanecer en la carretera.

Con el fin de lograr una mayor comprensión acerca de las categorías de aprendizaje mencionadas anteriormente, resulta de suma importancia establecer la diferenciación más importante entre lo que se conoce comúnmente como “programación tradicional”, y ML.

Inicialmente, los puntos de partida para la ingeniería de software tradicional y el aprendizaje automático son bastante similares. Ambos tienen como objetivo resolver problemas y comienzan familiarizándose con el dominio del problema: discutir con la gente, explorar el software y las bases de datos existentes. Las diferencias están en la ejecución de su desarrollo.

Ahora bien, por un lado, los desarrolladores tradicionales utilizan su intelecto para encontrar una solución y formularla como un programa que es factible de ejecutarse en un ordenador. Por otro lado, las personas que implementan sistemas de ML no intentan escribir un programa por sí mismos. En cambio, recopilan datos de entrada y los valores objetivo deseados. Luego, dan instrucciones a un ordenador para que encuentre un programa que calcule una salida para cada valor de entrada.

En definitiva, tradicionalmente los desarrolladores automatizan tareas escribiendo programas, mientras que en ML, un ordenador encuentra un programa que se ajusta a los datos (Figura 1).

El desarrollo de un sistema de ML es un proceso aún más iterativo y exploratorio que la ingeniería de software. ML se aplica a problemas que son demasiado complicados para que los humanos los resuelvan. Por tanto, un científico de datos debe adoptar una actitud experimental y estar preparado para probar algunos enfoques antes de decidirse por uno satisfactorio.

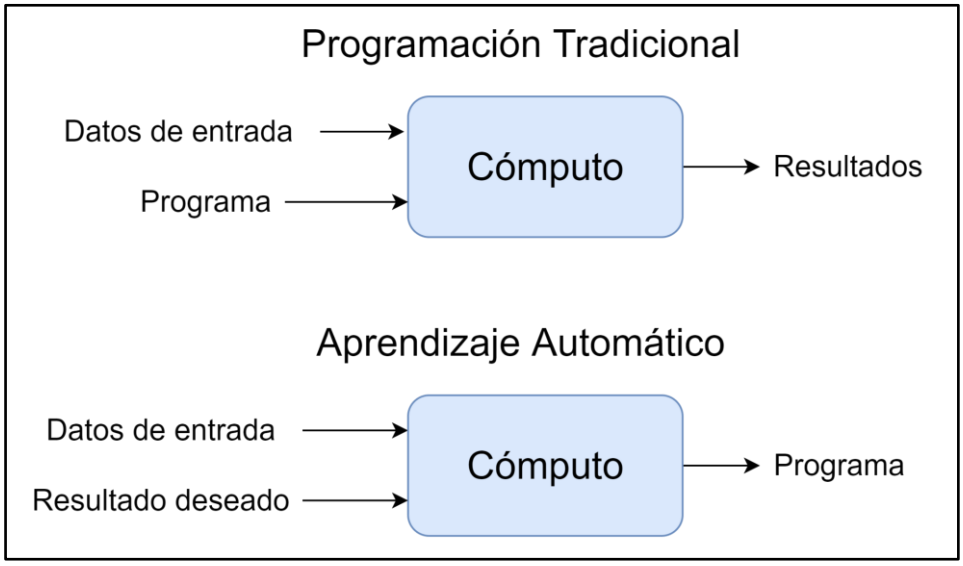


Figura 1. Comparación entre la programación tradicional y el aprendizaje automático

Ahora, una vez establecidas las diferencias entre la programación tradicional y el ML, se profundizarán los detalles relacionados con el Aprendizaje supervisado (que corresponde, justamente, a la categoría utilizada en el presente PFC).

2.3. Aprendizaje Supervisado

El aprendizaje supervisado es la rama más común de ML en la actualidad. Los algoritmos de aprendizaje automático supervisados están diseñados para aprender con el ejemplo. El nombre de aprendizaje "supervisado" se origina en la idea de que entrenar este tipo de algoritmos es como tener un profesor supervisando todo el proceso.

Al entrenar un algoritmo de aprendizaje supervisado, los datos de entrenamiento consistirán en entradas emparejadas con las salidas correctas. Un dataset, como su nombre lo indica, no es otra cosa que un conjunto de datos asociados a alguna temática en particular. Cuando se crea o utiliza un dataset, es recomendable que el mismo esté balanceado. Esto es, que el número de observaciones sea similar para cada una de las clases utilizadas en el dataset. Si no se logra este balance, puede ocurrir que el algoritmo tienda a favorecer la clase con la mayor proporción de observaciones, lo que puede llevar a métricas de exactitud sesgadas.

Durante el entrenamiento, el algoritmo buscará patrones en los datos que se correlacionen con los resultados deseados. Después del entrenamiento, un algoritmo de aprendizaje supervisado tomará nuevas entradas, y determinará en qué *label* (o etiqueta) se clasificarán las nuevas entradas según los datos de entrenamiento anteriores. El objetivo de un modelo de aprendizaje supervisado es predecir la etiqueta correcta para los datos de entrada que van ingresando. En su forma más básica, un algoritmo de aprendizaje supervisado se puede escribir simplemente como:

$$Y = f(x)$$

Donde Y es la salida predicha, que está determinada por una función de mapeo tal que asigna una clase a un valor de entrada x. La función utilizada para conectar los inputs al output predecido es creada por el modelo de ML durante su entrenamiento.

El aprendizaje supervisado puede ser dividido en dos subcategorías, a saber:

- Clasificación.

- Regresión.

A continuación, se explica más detalladamente estas dos subcategorías.

2.3.1. Clasificación

Durante su entrenamiento, un algoritmo de clasificación recibirá datos con una categoría asignada. El objetivo último de un algoritmo de clasificación es tomar un valor de entrada y asignarle una clase o categoría que se ajuste en función de los datos de entrenamiento proporcionados.

Un ejemplo simple de clasificación es el de determinar si un correo electrónico es spam o no (Figura 2). Debido a que, en este caso, existen solo dos posibles etiquetas para elegir ("spam" o "no-spam"), se trata de un problema de clasificación binaria. El algoritmo recibirá datos de entrenamiento con correos electrónicos que son spam y no spam. El modelo encontrará las características dentro de los datos que se correlacionan con cualquiera de las clases y creará la función de mapeo mencionada anteriormente: Y = f (x). Luego, cuando se le proporcione un correo electrónico no visto, el modelo utilizará esta función para determinar si el correo electrónico es spam o no lo es.

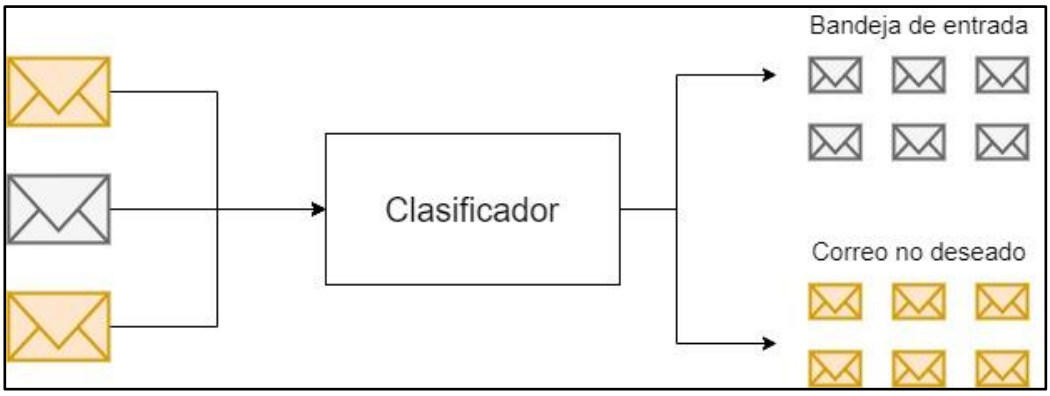


Figura 2. Ejemplo práctico de una clasificación binaria

Los problemas de clasificación se pueden resolver con una gran cantidad de algoritmos. El algoritmo que se decida utilizar depende tanto de los datos como de la situación particular.

Uno de los algoritmos más comunes de clasificación son las Máquinas de soporte vectorial (o Support Vector Machines -SVM, por sus siglas en inglés-). Este algoritmo se basa en una primera fase de entrenamiento y una segunda fase para la resolución de problemas. En ella, las

SVM se convierten en una “caja negra” que proporciona una respuesta (salida) a un problema dado (entrada).

Además, existen otros algoritmos de clasificación tales como:

- clasificadores lineales,
- árboles de decisión,
- K vecinos más próximos,
- etc.

2.3.2. Regresión

La regresión es un proceso estadístico predictivo, donde el modelo intenta encontrar la relación primordial entre las variables dependientes e independientes. El objetivo de un algoritmo de regresión es predecir un número continuo, como ventas, ingresos y puntajes de pruebas. La ecuación para la regresión lineal básica se puede escribir de la siguiente manera:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \ldots + w[i] * x[i] + b$$

Donde $x[i]$ es la característica de los datos y $w[i]$ y b son parámetros que se desarrollan durante el entrenamiento.

Hay diferentes tipos de algoritmos de regresión. Los tres más comunes son los siguientes:

- regresión lineal
- regresión logística
- regresión polinomial

En el caso de que se esté utilizando una sola variable independiente (x), se necesitarán dos parámetros para establecer una regresión de tipo lineal. Entonces, la fórmula de regresión lineal con una única variable x es de la forma:

$$y = wx + b$$

Luego, el aprendizaje consiste en encontrar los mejores parámetros posibles, a partir de los datos que se disponen. La estimación de los parámetros *w* y *b* resultará en encontrar, finalmente, una función lineal capaz de predecir nuevos datos a partir de los ya obtenidos (Figura 3).

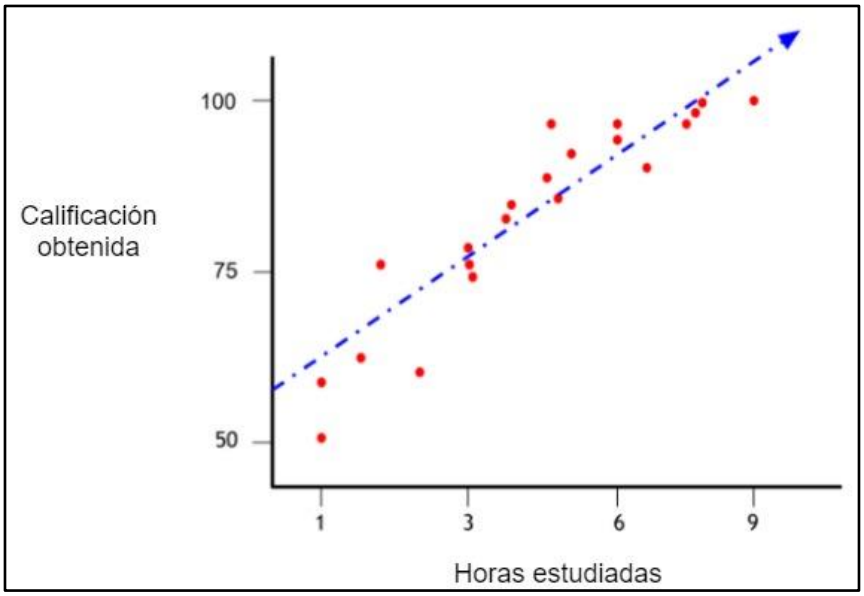


Figura 3. Ejemplo práctico de una regresión lineal

2.4. Procesamiento del Lenguaje Natural

Todo lo que expresamos, ya sea verbalmente o por escrito, contiene una gran cantidad de información. En principio, es posible entender e incluso predecir el comportamiento humano haciendo uso de esta información.

Ahora bien, una persona puede generar cientos o miles de palabras en una declaración, donde cada oración tiene su complejidad correspondiente. Así, los datos generados de conversaciones son ejemplos de **datos no estructurados**. Estos datos son complicados y difíciles de manipular. Sin embargo, gracias a los avances en disciplinas como el ML, se está produciendo una gran revolución en este tema. Hoy en día ya no se trata de intentar interpretar un texto o discurso en base a sus palabras clave (la antigua forma mecánica), sino de comprender el significado detrás de esas palabras (la **forma cognitiva**). Así, es posible detectar figuras retóricas tales como la ironía, o incluso realizar análisis de sentimientos.

El PLN es un campo de la IA que se ocupa de investigar y formular técnicas que permitan la comunicación entre humanos y máquinas utilizando lenguajes naturales, como lo son el español o el inglés. En otras palabras, el PLN otorga a las máquinas la capacidad de leer, comprender y derivar el significado de los lenguajes humanos.

Se trata de una disciplina que se centra en la interacción entre la Ciencia de datos y el lenguaje humano. Hoy en día, el PLN está en auge gracias a las enormes mejoras en el acceso

a los datos y al aumento de la potencia computacional, que están permitiendo a los profesionales lograr resultados significativos en áreas como salud, medios, finanzas y recursos humanos, entre otras.

En el campo del PLN, se analiza la estructura del lenguaje en 4 niveles:

- Análisis léxico. Consiste en un análisis interno de las palabras para extraer rasgos flexivos, unidades léxicas compuestas y otros fenómenos.
- Análisis sintáctico. Según el modelo gramatical empleado (lógico o estadístico) se busca analizar la estructura de las oraciones.
- Análisis semántico. Se basa en entender el significado o significados de cada frase.
- Análisis pragmático. Se busca analizar cada oración más allá de sus límites, tomando en cuenta su contexto.

Si bien se ha avanzado notoriamente en el campo del PLN, existen problemas que son inherentes a los lenguajes y que generan ambigüedad. Por ejemplo, muchas veces una frase no quiere decir lo que expresa en su forma más literal, como es el caso de las ironías.

Por esta razón, no es tan sencillo tratar computacionalmente una lengua, e implica un proceso de modelización matemática, ya que las computadoras sólo entienden bytes y dígitos.

Los principales inconvenientes que enfrentamos en estos días con el PLN se relacionan con el hecho de que el lenguaje es en extremo complejo. Es por ello que se utilizan diferentes técnicas para lidiar con los desafíos más comunes que presentan la comprensión y la manipulación del lenguaje

Debido a la complejidad inherente del lenguaje y, sumado a esto, el hecho de que los ordenadores sólo entienden bytes y dígitos, se han desarrollado diversos algoritmos para paliar esta complejidad. Los algoritmos para el procesamiento del lenguaje natural se pueden resumir en los siguientes:

- Eliminación de *stop words* (palabras de parada).
- Bolsa de palabras.
- Tokenización.
- Stemming.
- Lematización.

- Modelado de temas.
- Named Entity Recognition (NER).
- Text classification.

A continuación, se explica con detalle cada uno de estos algoritmos.

2.4.1. Eliminación de stop words

Las palabras de parada son aquellas palabras que no agregan demasiado significado a una oración. Esto es, pueden ser ignoradas sin sacrificar el significado de la oración.

Se trata de un proceso que elimina artículos comunes del idioma, pronombres y preposiciones. Se filtran y excluyen del texto palabras comunes que, en principio, aportan poco o ningún valor al fin último del algoritmo, eliminando así términos generalizados y frecuentes que no son informativos.

No existe una lista universal de palabras de parada. Éstas se pueden preseleccionar o construir desde cero. Un posible enfoque es el de comenzar adoptando palabras de parada predefinidas, y agregar palabras a la lista más adelante.

2.4.2. Bolsa de palabras

Este algoritmo consiste simplemente en contar todas las palabras en un fragmento de texto. Básicamente, se crea una matriz de ocurrencias para la oración, sin tener en cuenta la gramática y el orden de las palabras. Esta frecuencia de apariciones se utiliza luego para entrenar a un clasificador.

Algunas de las desventajas de esta técnica es la ausencia de significado semántico y de contexto, y el hecho de que las palabras de parada agregan ruido al análisis y algunas palabras no se ponderan correctamente.

Para resolver este problema, un enfoque consiste en cambiar la escala de la frecuencia de las palabras según la frecuencia con la que aparecen en todos los textos (no solo en el que se está analizando) para que las puntuaciones de palabras frecuentes como "el", que también son frecuentes en otros textos, sean penalizadas. Este método de puntuación se denomina “Frecuencia de términos - Frecuencia de documentos inversa” y mejora el algoritmo de la Bolsa

de palabras haciendo uso de pesos. A través de este método, los términos frecuentes en el texto son “recompensados”, pero también son “castigados” si esos términos son frecuentes en otros textos que también se incluyen en el algoritmo. Por el contrario, este método resalta y “recompensa” términos únicos o raros considerando todos los textos. Aun así, este enfoque aún no tiene contexto ni semántica.

2.4.3. Tokenización

Es el proceso de segmentar el texto en oraciones y palabras. Básicamente, es la tarea de cortar un texto en piezas llamadas *tokens* y, al mismo tiempo, desechar ciertos caracteres, como los de puntuación.

2.4.4. Stemming

Se refiere al proceso de cortar el final o el comienzo de las palabras con la intención de eliminar los afijos (adiciones léxicas a la raíz de la palabra) (Figura 4).

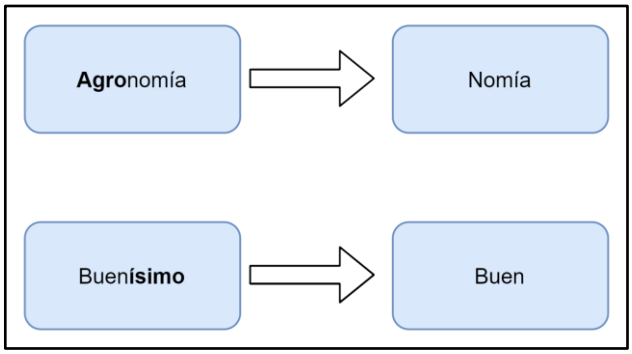


Figura 4. Ejemplo del proceso de Stemming

El problema con este método es que los afijos pueden crear o expandir nuevas formas de la misma palabra (llamados afijos flexivos), o incluso crear nuevas palabras ellos mismos (llamados afijos derivativos).

Un posible enfoque para solucionar esto es utilizar una lista de afijos y reglas comunes, y realizar derivaciones basadas en ellos. Sin embargo, este enfoque presenta limitaciones. Debido a que los *stemmers* utilizan enfoques algorítmicos, el resultado del proceso de *stemming* puede no ser una palabra real o incluso cambiar el significado de la palabra (y la oración). Para compensar este efecto, se pueden editar los métodos predefinidos agregando o eliminando afijos

y reglas. Aun así, se puede estar mejorando el rendimiento en un área mientras se produce una degradación en otra.

2.4.5. Lematización

La lematización tiene el objetivo de reducir una palabra a su forma básica y agrupar diferentes formas de una misma palabra. Por ejemplo, los verbos en tiempo pasado se cambian a presente (por ejemplo, "fue" se cambia a "va"), y los sinónimos se unifican. Por lo tanto, se estandarizan las palabras con un significado similar a su raíz. Aunque parece estar relacionado con el proceso de *stemming*, la lematización utiliza un enfoque diferente para llegar a las formas raíz de las palabras.

2.4.6. Modelado de temas

Este método se basa en agrupar textos para descubrir temas en función de sus contenidos, procesando palabras individuales y asignándoles valor en función de su distribución. Esta técnica se basa en los supuestos de que cada documento consta de una mezcla de temas y que cada tema consta de un conjunto de palabras, lo que significa que si podemos detectar estos temas ocultos podemos desbloquear el significado de los textos.

2.4.7. Named Entity Recognition (NER)

El reconocimiento de entidades nombradas, o NER (por sus siglas en inglés, Named Entity Recognition), es el proceso de encontrar y clasificar un conjunto de entidades en un texto. Una entidad puede ser cualquier palabra o conjunto de palabras que refieren a la misma cosa. Cada entidad detectada es clasificada dentro de una categoría predeterminada.

Cualquier modelo de NER debe seguir dos pasos, a saber:

1. Identificación de entidades. Corresponde a la detección de entidades en un texto, las cuales pueden ser desde una simple palabra a un conjunto de ellas. Cada palabra es conocida como *token*, y, por lo tanto, una entidad puede estar conformada por uno o más tokens.
2. Categorización de entidades. A cada entidad identificada en el paso anterior se le asigna una categoría. Estas categorías dependen de cada modelo en particular.

Es importante destacar que los modelos NER pueden ser implementados con técnicas de ML (como es el caso del presente proyecto), donde cada resultado obtenido conforma una predicción. Para que estas predicciones sean lo más precisas posible, cada modelo debe entrenarse con datos en donde se indiquen las categorías y las entidades a detectar. Cuanto mayor sean los datos de entrenamiento, mayor será la precisión del modelo.

NER se adapta a cualquier situación en la que se requiera una descripción general de alto nivel de una gran cantidad de texto. Algunos casos de uso notables de NER son los siguientes:

- Recursos humanos. Es posible acelerar los procesos de contratación resumiendo los CVs de los solicitantes, así como mejorar los flujos de trabajo interno categorizando las quejas y preguntas de los empleados.
- Atención al cliente. Se puede mejorar el tiempo de respuesta al cliente categorizando las solicitudes, quejas y preguntas de los usuarios, y filtrando por palabras claves.
- Motores de búsqueda y recomendación.
- Clasificación de contenido.
- Cuidado de la salud. También es posible mejorar los estándares de atención al paciente y reducir las cargas de trabajo extrayendo información esencial de los informes de laboratorio.
- Academia. NER permite, por ejemplo, que estudiantes e investigadores encuentren material relevante más rápidamente al resumir artículos y resaltar términos, tópicos y palabras clave.

2.4.8. Text Classification

La clasificación de textos (o Text Classification en inglés) es el proceso de categorizar textos en grupos predefinidos.

Los modelos de este estilo utilizan PLN para analizar grandes volúmenes de texto y asignarles una categoría en base al contenido de los mismos (Figura 5). Estos modelos son especialmente útiles para:

- Sentiment Analysis. Consiste en identificar si un texto habla de manera positiva o negativa sobre un determinado aspecto. Por ejemplo: reseñas de clientes sobre un determinado producto o servicio.

- Topic Detection. Consiste en poder detectar el tema sobre el que trata un documento. Por ejemplo: saber si un correo electrónico entrante pertenece a la carpeta de SPAM o bien a la bandeja de entrada.
- Language Detection. Consiste en poder conocer en qué idioma está escrito un texto.

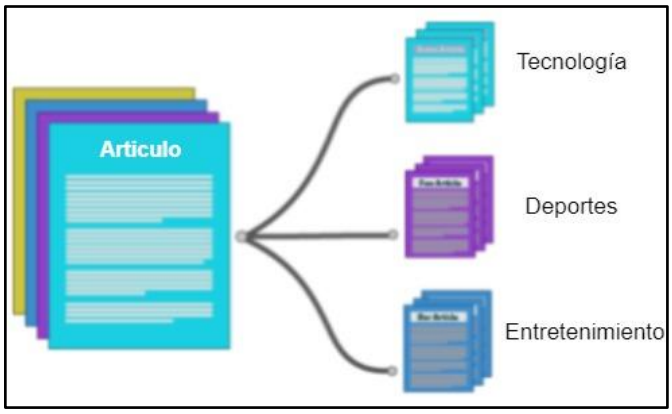


Figura 5. Clasificación de textos

Existen diversas formas de implementar un modelo de Text Classification, pero el que resulta de interés para el proyecto es aquel que utiliza ML para lograrlo. Estos modelos utilizan ejemplos previamente etiquetados para aprender a categorizar las entradas que recibe.

Los datos de entrenamiento se dividen en dos partes: por un lado, porciones de texto y, por otro, una etiqueta que los diferencia del resto. Luego de que el modelo es entrenado con suficientes datos, es posible obtener una buena precisión en el mismo y poder realizar la mencionada clasificación.

2.4.8.1. Text Classification mediante Machine Learning

La clasificación de textos mediante ML aprende a realizar clasificaciones basadas en observaciones pasadas. Al utilizar ejemplos pre-etiquetados como datos de entrenamiento, los algoritmos de ML pueden aprender las diferentes asociaciones entre piezas de texto, y que se espera una salida particular (es decir, etiquetas) para una entrada particular (es decir, texto).

El primer paso para entrenar a un clasificador de PNL de ML es la extracción de características: se utiliza un método para transformar cada texto en una representación numérica en forma de vector. Uno de los enfoques más utilizados es la bolsa de palabras, donde un vector representa la frecuencia de una palabra en un diccionario de palabras predefinido (véase sección 2.3.1. Bolsa de palabras).

Luego, el algoritmo de ML se alimenta con datos de entrenamiento que consisten en pares de conjuntos de características y etiquetas (por ejemplo, deportes, entretenimiento) para producir un modelo de clasificación (Figura 6).

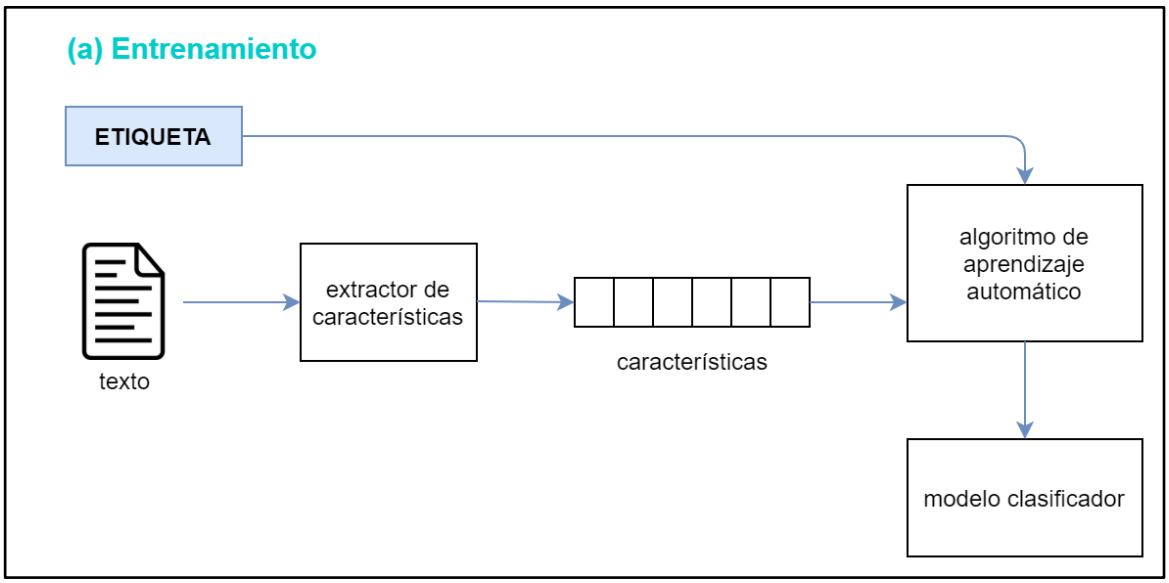


Figura 6. Proceso de entrenamiento de un clasificador de textos mediante ML

Una vez que se entrena con suficientes muestras, el modelo de ML puede comenzar a realizar predicciones precisas (Figura 7). El mismo extractor de características se utiliza para transformar texto en conjuntos de características. Luego, se introducen estas características en el modelo de clasificación para obtener predicciones en etiquetas (por ejemplo, deportes, entretenimiento).

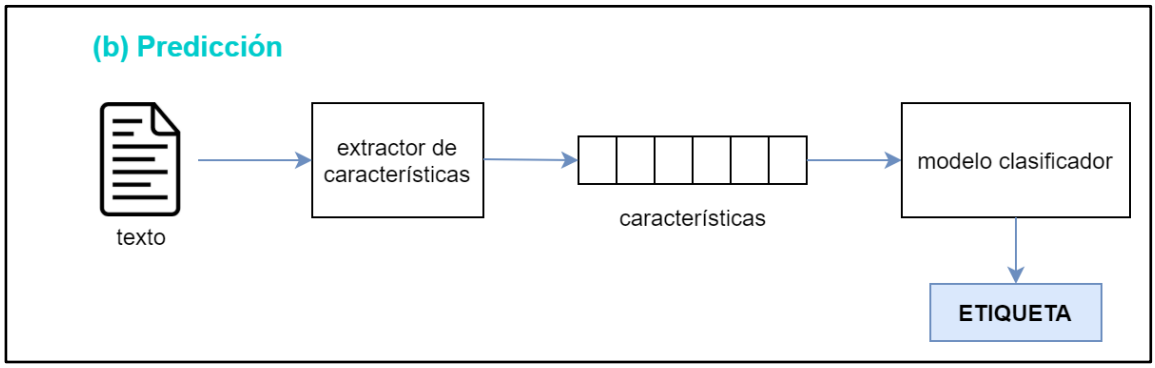


Figura 7. Proceso de predicción de un clasificador de textos mediante ML

3. Metodología Utilizada

La metodología que se decidió utilizar durante la elaboración del proyecto fue una de tipo iterativa experimental, comprendida por cuatro iteraciones. Al comienzo de cada una de ellas, se fijaron objetivos y modificaciones de acuerdo a los avances de la iteración anterior. Al final de cada una (exceptuando la iteración preliminar), se obtuvo un sistema ejecutable, a través del cual se lograron observar resultados parciales.

Este desarrollo por ciclos pequeños permitió dividir el problema final en etapas más sencillas de alcanzar, así como también poder evaluar y monitorear lo elaborado en cada iteración. De esta manera, se lograron identificar problemas y potenciales riesgos.

Para lograr el fin último del proyecto, se recurrió a una iteración preliminar (que consistió, en esencia, en capacitar al equipo de trabajo con las herramientas, tecnologías y conceptos teóricos necesarios para abordar el proyecto) y tres iteraciones incrementales y retroalimentadas entre sí, que permitieron la implementación práctica del proyecto. Como se mencionó anteriormente, cada iteración incremental culminó con un prototipo ejecutable del proyecto (Figura 8).

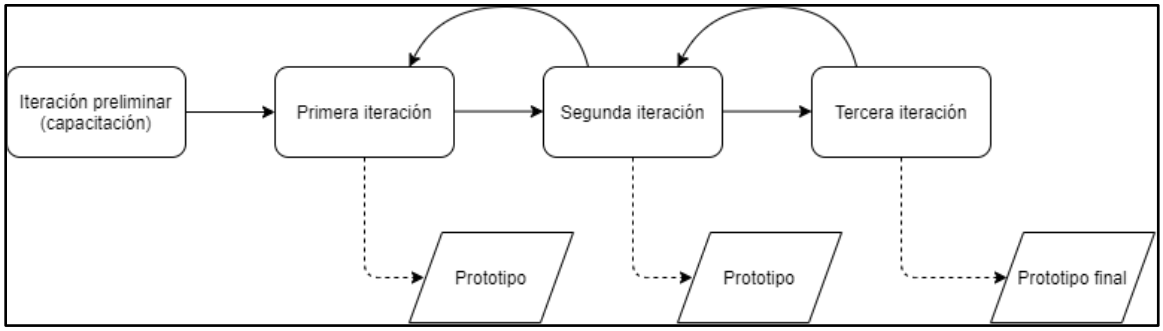


Figura 8. Diagrama de iteraciones realizadas durante la ejecución del proyecto

A continuación, se explica con detalle cada una de estas iteraciones.

3.1. Iteración preliminar

Antes de comenzar con cualquier desarrollo de software, fue necesario poder comprender todo lo relacionado a ML, como así también a Python, el lenguaje de programación más utilizado para estos tipos de proyectos.

Para lograr dicho propósito, se recopiló información y conocimiento a través de varios cursos, a saber:

Machine-learning con Python: una introducción práctica - por edX

Curso introductorio al mundo de ML, en donde se aprendieron las cuestiones más básicas asociadas a los siguientes tópicos: aprendizaje supervisado y no supervisado, regresión lineal y no lineal, y métodos de clasificación (árboles de decisión, regresión logística, máquinas de soporte vectorial).

Además, este curso ofreció información sobre otros temas como son el clustering y los sistemas recomendadores. Sin embargo, como no era de utilidad para el presente proyecto, no fueron tomados en cuenta.

El conocimiento obtenido aquí fue el puntapié inicial que permitió un gran acercamiento al aprendizaje automático.

Deep-learning - por Coursera

Curso especializado en ML, dictado por Andrew Ng. A diferencia del anterior, este curso presenta un nivel de detalle muy superior. Del mismo se logró comprender cómo se desarrolla una red neuronal y cómo funcionan sus distintas capas. Además, se obtuvo una interiorización importante acerca de cómo relacionar una red neuronal con PLN.

Just enough Python - por Udemy

Se trata de una introducción a los principales conceptos de Python. Los principales temas abordados son los siguientes: tipos de datos, definición de variables, estructura de los condicionales, funciones. En definitiva, se trata de un curso que permitió comenzar a entender el lenguaje Python desde sus bases, sin entrar en demasiados detalles avanzados.

3.2. Primera iteración

El objetivo de esta primera iteración fue poder crear un algoritmo simple basado en ML y PLN que pueda, dada una entrada de texto, poder decir si la misma es segura o insegura.

Para poder lograrlo, en primer lugar, se recolectaron distintos ejemplos encontrados en la web cuyo funcionamiento era similar al buscado, siempre teniendo en cuenta que los mismos estén desarrollados en Python y que apliquen técnicas de ML. Luego, el siguiente paso fue ejecutarlos de manera local y analizar cada línea de código para poder comprender su funcionamiento.

En este punto, cuando ya se incorporó cierto nivel de conocimiento acerca de cómo era la estructura general de un algoritmo de este estilo, se propuso comenzar a implementar el propio, que dé sustento inicial al proyecto. Para ello, se recopilaron secciones de código de los ejemplos encontrados con anterioridad, recuperando aquellas porciones de cada algoritmo que más se ajustaban a la implementación buscada.

Paralelamente, se fue realizando un dataset para poder entrenar el modelo del algoritmo desarrollado. Dicho dataset estuvo constituido por los subtitulos de la película “Menace Society”, ya que la misma contiene bastante violencia y escenas de inseguridad. Inicialmente, se decidió etiquetar las frases con las categorías “Seguro” e “Inseguro”, por una cuestión de simplicidad. Es de destacar que, de las 1300 (mil trescientas) frases que contiene la película, sólo se conservaron las primeras 800 (ochocientas), para que, de esta manera, el dataset resulte correcto y balanceado.

Como resultado de esta iteración, se obtuvo un algoritmo que utiliza SVM como método de clasificación, entrenado con un dataset de origen propio y que, ante una entrada de frase o texto, indica si la misma es segura o no.

Esta iteración fue en extremo importante ya que, gracias a ella, se comprendió de forma completamente práctica todo el conocimiento incorporado en la iteración anterior.

3.3. Segunda iteración

El algoritmo desarrollado en la etapa anterior presentó un contratiempo clave: las frases eran analizadas de forma aislada, es decir, el texto ingresado era clasificado como “Seguro” o “Inseguro”, pero no se relacionaba de ninguna manera las interacciones entre frases consecutivas. De tal manera, si se deseaba conocer el nivel de seguridad de una conversación entre dos personas, era necesario conocer la misma de manera completa de antemano. En otras

palabras, se debía esperar a que dos personas terminen de hablar por completo para luego analizar dicha conversación. Esto, naturalmente, no es deseable.

El prototipo de software debía, necesariamente, ir analizando la conversación a medida que iba ocurriendo, en tiempo real. Así, al detectar una situación de inseguridad, el software tenía que tomar las medidas correspondientes.

Por tanto, el objetivo de esta segunda iteración fue el de solucionar el problema previamente descripto. Para lograr dicho cometido, se propuso dividir el problema en dos módulos, tal que resulte en la implementación de dos algoritmos en lugar de uno, donde el segundo toma la salida del primero. Sin entrar en demasiados detalles, ya que el funcionamiento de dichos módulos será explicado en la sección "7. Herramienta de software para la detección de hechos de inseguridad", se puede decir que el mecanismo básico es el siguiente:

1. El primer módulo obtiene las entidades del texto que se ingrese. Dichas entidades, definidas por el equipo de trabajo, se dividen en dos categorías: aquellas relacionadas a situaciones cotidianas, y aquellas asociadas a hechos de inseguridad.
2. El segundo módulo analiza las entidades que obtiene del primero e indica si dicho conjunto representa una situación de peligro o no.

Para la realización de los algoritmos se recurrió a una herramienta de PLN, que brinda las facilidades necesarias para llevar a cabo este propósito. Además, para una correcta implementación de los mismos, se recurrió a documentación proporcionada por las librerías utilizadas, como así también a ejemplos encontrados en la internet.

Al mismo tiempo, se fueron creando dos datasets para el entrenamiento de ambos módulos. Para el primero, se etiquetaron por completo cuatro capítulos de la serie "El Marginal", identificando en cada frase las entidades que nuestro algoritmo debía detectar. Para el dataset del segundo algoritmo, fue necesario crear un pequeño programa que genera frases con las entidades que el primer módulo detecta, para luego etiquetar dichas frases con las categorías "Seguro" o "Inseguro", según corresponda.

Finalmente, cuando ambos módulos fueron entrenados, se procedió a crear un tercer algoritmo que se encarga de la conexión entre ellos, permitiendo al usuario introducir una o

más frases para luego analizarlas y poder determinar si el conjunto de las mismas representa un hecho de inseguridad o no.

Como resultado de esta iteración, se obtuvo un prototipo inicial de software capaz de analizar una conversación entera, y no solo frases aisladas.

3.4. Tercera iteración

Para esta iteración, se propuso como meta mejorar lo realizado en la iteración anterior, teniendo como principal objetivo incrementar la precisión de ambos módulos. La principal influencia en la precisión de los algoritmos está dada por la composición de sus datasets, por ello, lo que se hizo fue incrementar el tamaño de ambos.

Para el algoritmo encargado de detectar entidades, como primera medida, se hizo una revisión de la lista de entidades que el sistema debía detectar. En este sentido, se quitaron ciertas entidades y se agregaron otras, como resultado de un exhaustivo análisis. Con la lista de entidades ya actualizada, se procedió a etiquetar subtítulos de series y películas donde la violencia es en extremo predominante.

Por otra parte, para el algoritmo encargado de recibir una lista de entidades y de clasificarlas como segura o insegura, se mejoró el dataset en dos aspectos: por un lado, se incrementó su tamaño, pasando de 100 (cien) frases a 1000 (mil); por otro, se definió que no todas las frases tengan la misma longitud, para que, de esta manera, el algoritmo se adapte mejor a la salida del primer módulo. Por último, y para que el sistema pueda detectar distintos niveles de inseguridad, las etiquetas pasaron de ser dos (“Seguro” e “Inseguro”) a cuatro (“Muy inseguro”, “Inseguro”, “Seguro” y “Muy seguro”).

Es importante mencionar que, hasta el momento, todo entrenamiento realizado sobre los módulos se realizaba de manera local, consumiendo mucho tiempo y memoria, lo cual no es del todo deseable. Para solucionar esto, se recurrió a servicios en la nube, más específicamente a Google Colab, los cuales son mucho más veloces y permitieron aumentar el número de iteraciones en cada entrenamiento. De esta forma, se mejoró aún más la precisión de cada módulo.

Además, se modificó también el sistema que hace de conexión entre ambos módulos. Se agregó una interfaz gráfica, similar a la de un chat, para mejorar la interacción con el usuario.

4. Tecnologías utilizadas

En esta sección se presentan las distintas tecnologías que fueron utilizadas para el desarrollo del presente PFC. En primer lugar, se detalla el lenguaje de programación principal empleado durante el desarrollo del software. Más adelante, se hace un análisis detallado de las herramientas esenciales que se utilizaron durante el desarrollo del proyecto, justificando luego la elección de una de ellas. Luego, se detallan las librerías y herramientas de soporte que han sido empleadas en conjunto con las esenciales. Finalmente, se listan los entornos de desarrollo sobre los cuales se escribió el código fuente del proyecto en cuestión.

4.1. Lenguajes de programación

4.1.1 Python

Python es uno de los lenguajes de programación más utilizados para proyectos relacionados con ML. Esto se debe, entre otras cosas, a que es posible encontrar una gran cantidad de librerías y frameworks que resultan de utilidad para proyectos vinculados al aprendizaje automático. Además, los algoritmos escritos en Python poseen un nivel de performance muy elevado. Por último, es posible encontrar una gran cantidad de ejemplos y soporte en la web ya que es uno de los lenguajes más populares y con una comunidad muy activa.

4.2. Bibliotecas y herramientas esenciales

Esta sub-sección está destinada a justificar la herramienta esencial sobre la que basamos los esfuerzos de nuestro proyecto: **spaCy**. Para ello, inicialmente se enumeran y detallan los puntos claves que debe cumplir la herramienta para que pueda ser utilizada en el desarrollo del proyecto. Luego, se lista el abanico de opciones que se dispone, con una breve descripción de cada herramienta. Finalmente, se establece una comparativa entre las distintas opciones disponibles, y se justifica la elección realizada.

4.2.1. Características deseables

A continuación, se presenta una lista de características deseables sobre las cuales se basa nuestro análisis y comparación posterior:

1. Velocidad de respuesta. Mide qué tan rápido se obtiene la respuesta deseada por parte de la herramienta. Este aspecto es en extremo importante, ya que ante situaciones de peligro se debe actuar lo más rápido posible.
2. Soporte al español. De acuerdo al contexto espacial del PFC (fue realizado enteramente en Argentina), es necesario que la herramienta pueda dar soporte al español.
3. Compatibilidad con Python. Python es el lenguaje mayormente utilizado para proyectos de ML, principalmente debido a su performance. Por esta razón, es necesario que la herramienta pueda dar soporte al mismo.
4. Comunidad. Cuanto mejor y más rica sea la comunidad, mejores beneficios se tendrá en cuanto a dudas sobre cómo utilizar la herramienta.
5. Módulos. Son requeridos los módulos de NER y TEXTCAT, ya que son el núcleo central del presente proyecto.
6. Libre y de código abierto. Es importante que la herramienta sea gratuita en cualquier circunstancia y que se pueda utilizar bajo cualquier propósito.
7. Forma de ejecución. Se pueden diferenciar entre ejecución local o en la nube. En este sentido, tendrán prioridad aquellas que puedan ejecutarse totalmente de forma local para evitar comprometer la privacidad de las personas que utilicen el sistema desarrollado.

4.2.2. Herramientas a analizar

La lista de opciones disponibles de herramientas es la siguiente:

- spaCy. Es una librería para el procesamiento de lenguaje natural de código abierto, desarrollada por Explosion AI.
- NLTK. Por sus siglas en inglés, Natural Language Toolkit, es otra plataforma para construir programas que trabajen con PLN.
- Stanford CoreNLP. Herramienta que contiene varios modelos lingüísticos para trabajar en diversos lenguajes de programación.
- Apache OpenNLP. Librería basada en ML para PLN, desarrollada por Apache Software Fundation.

- Apache UIMA. Por sus siglas en inglés, Unstructured Information Management Applications, son aplicaciones que pueden analizar grandes volúmenes de datos no estructurados. El lenguaje humano pertenece a este grupo.
- MonkeyLearn. Es una herramienta basada en IA, capaz de procesar grandes cantidades de texto.

En la figura 9 se pueden observar los logotipos de cada una de las herramientas a ser analizadas.

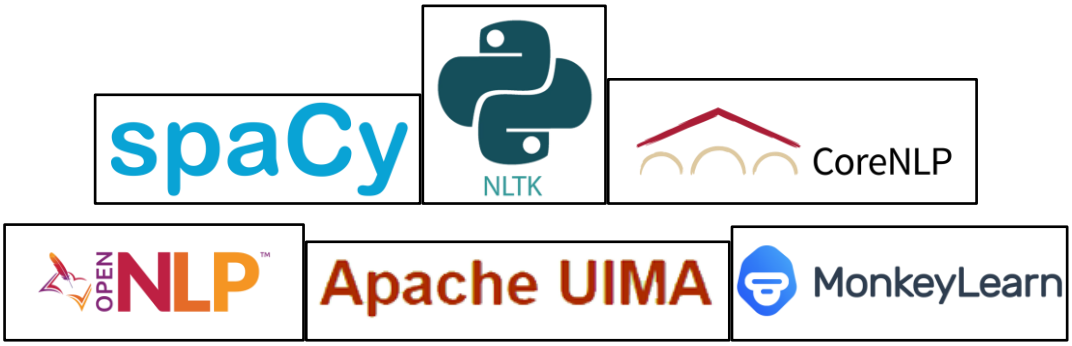


Figura 9. Logotipos de cada una de las herramientas analizadas

4.2.3. Comparativa y elección final

A continuación, se presenta una tabla comparativa (Tabla 1) entre las distintas herramientas descritas anteriormente, seguida de un análisis y fundamentación de cuál de ellas fue elegida para el presente proyecto.

	Velocidad de respuesta	Soporte español	Compatibilidad con Python	Comunidad activa	Módulos requeridos	Libre	Ejecución local
spaCy	Rápido	Sí	Sí	Sí	Sí	Sí	Sí
NLTK	Lento	No	Sí	Sí	Sí	Sí	Sí
Stanford CoreNLP	Lento	Sí	Sí	Sí	Sí	Sí	Sí
Apache OpenNLP	Sin información	Sí	No	Sí	Sí	Sí	Sí
Apache UIMA	Sin información	No	No	No	No	Sí	Sí
MonkeyLearn	Lento	Sí	Sí	Sí	Sí	No	No

Tabla 1. Comparativa entre las diferentes herramientas a analizar

Las tecnologías analizadas aquí tienen un amplio rango de características, como se puede observar en la tabla comparativa. Uno de los principales aspectos a tener en cuenta, como se mencionó con anterioridad, es que la herramienta se pueda ejecutar de manera local. Esto quiere decir que se pueda instalar en una PC y que no requiera de conexión a internet para funcionar. Esto trae consigo dos ventajas: la primera es que se ahorran los gastos derivados de tener una conexión a internet funcionando constantemente; la otra, es que los datos obtenidos no son enviados a la nube, protegiendo la privacidad de las personas. Por esta razón, como primera medida, podemos descartar a MonkeyLearn por no cumplir con la característica mencionada.

Ahora bien, este proyecto está destinado a los límites de la República Argentina, cuyo idioma oficial es el español, por lo tanto es muy importante que la herramienta tenga soporte para este idioma. De esta forma, Apache UIMA y NLTK quedan fuera de las opciones posibles.

Como se mencionó en la sección “4.1. Lenguajes de programación”, se decidió utilizar Python como lenguaje de programación para el componente de software a realizar, ya que su performance en trabajos de ML es superior a la de otros lenguajes. De las tecnologías analizadas, Apache OpenNLP solo está disponible para Java, con lo cual no es posible utilizarla en este PFC.

Finalmente, solo quedan spaCy y Stanford CoreNLP como posibles opciones, las cuales son muy similares en cuanto a su funcionamiento. Ambas poseen casi todas las características

deseables para el proyecto; sin embargo, la diferencia radica en la velocidad de respuesta de ambas y en su precisión. Según estudios [35], spaCy lidera a sus competidores en términos de velocidad de respuesta y precisión de modelos.

En conclusión, por las razones dispuestas hasta aquí, la herramienta que se utilizó en el presente proyecto es spaCy.

4.2.4. spaCy

spaCy es una librería libre y de código abierto desarrollada por Explosion AI, destinada al procesamiento avanzado del lenguaje natural. Permite crear aplicaciones que analizan y "entienden" grandes volúmenes de texto. De esta manera, puede ser utilizada para la realización de sistemas que trabajen directamente con PLN, o bien para el preprocesamiento de texto para sistemas basados en ML. Una importante característica de spaCy es que brinda soporte a más de 16 (dieciséis) idiomas.

A diferencia de otras herramientas, spaCy no es una plataforma que provee servicios, como sí lo haría una aplicación web. En cambio, spaCy es una biblioteca que está diseñada para ayudar a la creación de aplicaciones. A continuación, se listan las características más destacadas que provee spaCy:

- Part-Of-Speech Tagging. Luego de realizar el proceso de tokenization, se pueden clasificar las palabras obtenidas en categorías como sustantivos, verbos, entre otros.
- Dependency parsing. Es el análisis gramatical de una oración, estableciendo las relaciones entre sus partes.
- Named Entity Recognition. Proceso explicado en la sección "2.4.7. Named Entity Recognition (NER)".
- Text Classification. Proceso explicado en la sección "2.4.8. Text Classification".
- Tokenization. Proceso explicado en la sección "2.4.3. Tokenización".
- Lemmatization. Proceso explicado en la sección "2.4.5. Lematización".
- Sentence Boundary Detection. El objetivo es delimitar el comienzo y fin de cada oración dentro de un texto.
- Entity Linking. Cumple la función de asignar una identidad única a entidades obtenidas en un texto, de acuerdo a una base de conocimiento.

- Similarity. Proceso que consiste en comparar textos para ver qué tan parecidos son unos con otros.
- Rule-based Matching. Consiste en encontrar una secuencia de palabras de acuerdo a expresiones regulares.
- Training. spaCy está compuesto por módulos (NER, TEXTCAT, entre otros), donde cada uno de ellos ofrece una funcionalidad específica. Ahora bien, cada módulo resulta funcional gracias a un modelo estadístico asociado a él. Dicho modelo es estadístico ya que cada resultado que se obtiene de ellos conforma una predicción. Estas predicciones se hacen en base a ejemplos que se utilizaron para entrenar a los modelos. De esta manera, spaCy ofrece la posibilidad de entrenar uno o más módulos con un conjunto de datos deseado, que se ajuste a las necesidades y/o requerimientos buscados.
- Serialization. Es el proceso de guardar los cambios que se realicen sobre los módulos de spaCy en forma de archivo, para luego poder acceder a ellos.

4.2.4.1. Arquitectura de spaCy

spaCy, para cada uno de sus modelos, utiliza una librería de aprendizaje profundo de origen propio llamada *thinc*. En ella, se definen los modelos para cada uno de los módulos que presenta spaCy, y cada modelo está basado en una estructura de red neuronal convolucional (o CNN por sus siglas en inglés: Convolutional Neural Network).

La estructura de la red neuronal está basada en un framework propio llamado “Embed, encode, attend, predict” [37]. De forma resumida, este framework funciona de la siguiente manera:

- Embed. Cada caracter del texto que recibe spaCy es transformado en un vector que representa su valor en formato ASCII.
- Encode. Los vectores obtenidos en el paso anterior se utilizan para formar una matriz.
- Attend. Se reduce el tamaño de la matriz, quedándose con las partes más importantes de la misma, hasta transformar la matriz en un vector nuevamente.
- Predict. Una vez obtenido el vector del paso anterior, éste se utiliza para entrenar el modelo y poder realizar una predicción.

4.3. Bibliotecas y herramientas de soporte

Diversas herramientas y librerías de soporte fueron necesarias para la ejecución de ciertas tareas específicas durante las iteraciones del proyecto. Éstas fueron las siguientes:

- AppJar.
- Doccano.
- Docker.
- SubtitleEdit.
- Google Drive.
- Google Hangouts.
- PineTools.
- Draw.io.

A continuación, se explica brevemente cada una de ellas.

4.3.1. AppJar

Al igual que spaCy, es una biblioteca de código abierto en Python, pero en este caso es para el desarrollo de interfaces de usuario. Brinda una gran cantidad de opciones de personalización o "widgets" que permiten crear interfaces que se adaptan a casi cualquier desarrollo de software.

4.3.2. Doccano

Doccano es una herramienta también de código abierto, que permite etiquetar texto de diversas maneras. Por lo tanto, con Doccano se pueden crear datasets para diversos algoritmos, entre los cuales se encuentran NER y TEXTCAT. Además, da la posibilidad no solo de importar datos en diversos formatos, sino también de exportarlos a otros tantos.

4.3.3. Docker

Es una herramienta que permite la ejecución de software dentro de contenedores, brindando una capa adicional de abstracción en múltiples sistemas operativos. Docker aisla recursos para que de esta manera cada

contenedor se ejecute de manera independiente. Su utilización fue necesaria para la ejecución de Doccano.

4.3.4. SubtitleEdit

Se trata de un software gratis y de código abierto que permite crear, editar, ajustar o sincronizar subtítulos para videos. Su utilización fue necesaria para eliminar líneas de texto o caracteres innecesarios, lo que permitió posteriormente etiquetar subtítulos de series y películas sin mayores problemas.

4.3.5. Google Drive

Google Drive es un servicio de alojamiento de archivos en la nube. Casi todos los archivos relacionados al presente proyecto fueron almacenados en un repositorio compartido de Google Drive, donde todos los integrantes tenían permisos de lectura y escritura.

4.3.6. Google Hangouts

Google Hangouts es una herramienta que permite realizar llamada, pudiendo compartir la pantalla de escritorio entre los participantes. Fue una herramienta crucial para el desarrollo del proyecto, permitiéndonos a cada uno de los participantes contribuir al mismo sin necesidad de asistir personalmente a reuniones periódicas.

4.3.7. PineTools

Es una herramienta web que brinda una gran cantidad de servicios, dentro de los cuales fue utilizado aquel que permite eliminar duplicados en archivos de texto. De esta manera, fue posible quitar texto duplicado en los datasets antes de que los mismos fueran etiquetados.

4.3.8. Draw.io

Página web que permite la creación de distintos tipos de diagramas. Es gratuita y se puede vincular con Google Drive.

4.4. Entornos de desarrollo

Para escribir el código fuente del software desarrollado, se hizo uso de los siguientes entornos de desarrollo:

- Jupyter Notebook.
- Visual Studio Code.
- Google Colab.

A continuación, se explica brevemente la funcionalidad de cada software, así como los motivos de su uso.

4.4.1. Jupyter Notebook

Es un entorno interactivo basado en la web. Fue el primer entorno de desarrollo que se utilizó, elegido principalmente por su simplicidad y porque se conocía su funcionamiento con anterioridad, pues se ha utilizado durante el cursado de la cátedra de Inteligencia Artificial. Fue empleado en la primera iteración del PFC, para ejecutar los ejemplos de códigos que fueron recopilados de la web, como así también para la creación del primer algoritmo en Python desarrollado por el equipo de trabajo.

4.4.2. Visual Studio Code

Es un editor de código fuente con una gran cantidad de funciones, tales como soporte para la depuración, integración con Git, entre otras. A partir de la segunda iteración, se requirió de un entorno de desarrollo más potente y con mayores capacidades que el usado hasta el momento. La opción a elegir fue Visual Studio Code ya que, nuevamente, todos los integrantes del equipo tenían conocimiento en este software.

4.4.3. Google Colab

Es una herramienta que permite la ejecución de código Python en la nube. A partir de la tercera iteración fue necesario recurrir a alternativas para el entrenamiento de los modelos, ya que el entrenamiento local dependía de los recursos disponibles en la PC donde se ejecutaba. Se optó por esta herramienta debido a que es gratuita y forma parte del ecosistema de los servicios de Google, los cuales también fueron utilizados.

5. Conformación de los datasets

5.1. Introducción

Para el desarrollo de este PFC, fue necesario crear dos datasets: uno para el módulo NER, y otro para el correspondiente a TEXTCAT. En las siguientes dos secciones se detalla la forma de obtención de cada uno de estos datasets.

5.2. Dataset para el módulo NER

En esta sección se explican, primeramente, las entidades definidas para el dataset correspondiente al módulo NER, así como los motivos de dicha elección. Luego, se describe brevemente el origen de los datos recolectados para este dataset. Finalmente, se justifican las estrategias utilizadas para etiquetarlo.

5.2.1. Definición de entidades

El análisis para la detección y definición de entidades se basó en tres puntos esenciales. A continuación, se enumera cada uno de ellos y se exponen los motivos de cada premisa.

1. Cada entidad debe caer dentro de uno de los siguientes dos conjuntos: entidad de riesgo o de uso común.

Los móviles de esta necesidad son los siguientes:

- Si solo se detectan entidades de riesgo, resulta imposible determinar el "tipo" de conversación que se está llevando a cabo. Por ejemplo, sea una conversación en la que se presenta una amenaza seguida de un pedido de disculpas. En este caso, si solo se consideran las entidades de riesgo, sólo se detectaría la amenaza, resultando en que el sistema devuelva dicha acción como peligrosa, cuando evidentemente no lo es.
- Si solo se hace uso de entidades de riesgo, resultaría imposible diferenciar los niveles de inseguridad.
- NER intenta encontrar las entidades en un texto dado. Si solo se tuvieran las entidades de riesgo, NER enfocaría los esfuerzos en solo encontrar estas entidades y, quizás, por fallos en la precisión, podría llegar a detectar, por ejemplo, un saludo como un insulto.

En cambio si explícitamente se le especifica a NER que detecte saludos, las probabilidades de que se equivoque en este caso serán mucho menores.

2. Las entidades deben ser lo más específicas y atómicas posibles.

Una entidad debe ser algo que pueda ser identificable con pocas palabras. NER establece, como premisa, que las entidades sean justamente lo más atómicas posibles. Es por ello que este punto es importante para la definición de las entidades.

3. Las entidades no deben depender del contexto.

Una máquina, en su más bajo nivel, solo reconoce bits, y no palabras como lo hacen los humanos. Este es un problema al que se enfrenta el PLN. Si una entidad depende del contexto al que pertenece, entonces se requiere un esfuerzo mayor por parte de la máquina para que dicha entidad pueda ser detectada. Esto reduce la precisión del algoritmo. En cambio, si las entidades no dependen del contexto, NER puede detectarlas con mayor precisión.

Teniendo en cuenta estos tres puntos, se concluyó en definir las entidades mostradas en la Tabla 2.

Entidades comunes	Entidades de riesgo
Agradecimiento	Insulto
Saludo	Amenaza
Aprobación	Rechazo
Orden Amistosa	Orden Amenazante
Pedir disculpas	Pedido de Ayuda

Tabla 2. Definición de entidades para el módulo NER

Se describe a continuación el significado semántico que se le dio a cada entidad, seguido de un ejemplo práctico de su uso (Tabla 3):

Entidad	Descripción semántica	Ejemplo
Agradecimiento	Expresiones de gratitud, reconocimiento y estima por haber hecho un favor o prestado un servicio.	“Te agradezco”
Saludo	Expresiones de cortesía en encuentros o despidos	“Buen día”
Aprobación	Situaciones en las que se aprueba cierto comportamiento, decisión, opinión o acción.	“Bueno, sí”
Orden Amistosa	Mandatos o solicitudes manifestadas de manera amistosa, con buenos modales.	“Llevale eso”
Pedir disculpas	Expresiones empleadas para pedir perdón, disculparse o lamentarse por algún hecho o situación particular.	“Lo lamento”
Insulto	Palabras o expresiones empleadas para provocar una ofensa hacia una persona.	“La p*** madre”
Amenaza	Advertencias en las que existe cierto nivel de violencia implicado	“Te voy a matar”
Rechazo	Situaciones en las que se desaprueba cierto comportamiento, decisión, opinión o acción.	“No te creo”
Orden Amenazante	Mandatos o solicitudes manifestadas con cierto nivel de violencia, intimidación, aviso, apercibimiento o amenaza.	“¡Al piso carajo!”
Pedido de Ayuda	Peticiones de auxilio ante situaciones extremas.	“¡Ayúdenme! ¡Por favor!”

Tabla 3. Descripción semántica y ejemplo de cada una de las entidades del dataset correspondiente a NER

5.2.2. Recolección de datos

Todos los datos recolectados para este dataset fueron obtenidos de subtítulos de películas y series, enfocando los esfuerzos en aquellas en las que se presentan con más frecuencia situaciones de violencia e inseguridad. Los motivos de esta elección fueron los siguientes:

- Los subtitulos de peliculas y series son fáciles de obtener en internet.
- Las escenas de películas y series representan, usualmente, conversaciones entre personas, siendo este el foco de esfuerzo del PFC.

- Los subtítulos son simples de tratar. Una vez obtenido el archivo de subtítulos (normalmente con extensión *.srt*), se lo pasa a algún formato de texto plano (como *.txt*) y ya está disponible para su tratamiento posterior.

5.2.3. Etiquetado

Para cada frase de un archivo de subtítulos, se etiquetó la o las palabras que podían ser reconocidas como alguna de las entidades mencionadas en la sección "5.2.1. Definición de entidades". Por tanto, una frase puede contener una o más etiquetas.

Es importante destacar que, naturalmente, algunas frases no podían ser emparejadas con ninguna de las entidades definidas por el equipo de trabajo. En esos casos, las frases quedaron sin etiquetar.

Se muestran a continuación algunas capturas del etiquetado de este dataset (Capturas 1 a 4). Para el proceso de etiquetado se utilizó la aplicación Doccano, tal como se mencionó en la sección "4.3.2. Doccano".

Para evitar posibles ofensas al lector de este informe, se decidió censurar aquellas palabras que contienen insultos o ciertos conceptos inapropiados.

Captura 1. Ejemplo de una frase etiquetada con el label "Amenaza"

Captura 2. Ejemplo de una frase etiquetada con dos labels "Aprobación"

Captura 3. Ejemplo de una frase etiquetada con el label "Insulto"

Captura 4. Ejemplo de una frase etiquetada con los labels "Orden amenazante" e "Insulto"

Las estadísticas correspondientes al dataset mencionado se encuentran en la sección "11.1. Anexo A" del presente informe.

5.3. Dataset para el módulo TEXTCAT

En esta sección se explican, primeramente, los motivos por los cuales se definieron los niveles de inseguridad correspondientes al dataset de TEXTCAT. Luego, se expone brevemente el algoritmo desarrollado para recolectar los datos de dicho dataset. Finalmente, se describen las estrategias utilizadas para etiquetarlo.

5.3.1. Definición de niveles de inseguridad

La primera opción que se consideró para definir los niveles de inseguridad, fue la de establecer categorias de situaciones de inseguridad, tales como asesinato, violación, crimen,

entre otros. Esta contemplación fue luego descartada, pues se requería de un gran dataset para ello, ya que las categorías eran demasiado específicas.

Por tanto, se concluyó que las categorías debían ser definidas en base a propiedades cualitativas. Inicialmente, se eligieron los niveles “Seguro” e “Inseguro” (durante la ejecución de la segunda iteración), con fines de empezar a hacer pruebas básicas. Más adelante, durante el desarrollo de la tercera iteración del proyecto, se decidió agregar los niveles “Muy seguro” y “Muy inseguro”. Así, se optó por no agregar más categorías, ya que los límites se hubieran vuelto confusos entre los diferentes niveles.

Para este módulo, quedaron determinados los siguientes niveles de inseguridad:

- Muy seguro. Representa situaciones de la vida cotidiana, tales que no existe ningún tipo de entidad de riesgo. Un ejemplo de secuencia de entidades que derivan en la categoría de “Muy seguro” puede ser la siguiente: Pedir Disculpas - Orden Amistosa - Aprobación - Aprobación - Saludo - Aprobación - Agradecimiento - Agradecimiento. Este nivel representa un 0% (cero por ciento) de inseguridad.
- Seguro. Representa situaciones en donde existen ciertos insultos aislados u órdenes amenazantes, seguidas de un pedido de disculpas. Corresponde a aquellas situaciones en las que, por ejemplo, dos amigos se saludan con insultos de por medio, o donde existe una pequeña “tensión” asociada a órdenes amenazantes, pero luego se piden disculpas. Un ejemplo de secuencia de entidades que derivan en la categoría de “Seguro” puede ser la siguiente: Insulto - Pedir Disculpas - Rechazo - Saludo. Este nivel representa un 33% (treinta y tres por ciento) de inseguridad.
- Inseguro. Representa aquellas circunstancias en las cuales hay uno o más insultos o amenazas, pero no están presentes los pedidos de ayuda. También fueron consideradas en este nivel todas las situaciones en las que existen muchas órdenes consecutivas, independientemente de si sean amenazantes o no. Un ejemplo de secuencia de entidades que derivan en la categoría de “Inseguro” puede ser la siguiente: Pedir Disculpas - Rechazo - Orden Amenazante - Aprobación - Agradecimiento - Amenaza - Aprobación - Orden Amistosa. Este nivel representa un 66% (sesenta y seis por ciento) de inseguridad.
- Muy inseguro. En esta categoría caen aquellas situaciones en las que hay insultos, amenazas, órdenes amenazantes y, sobre todo, pedidos de ayuda. En su composición

estructural, hay más entidades de riesgo que entidades de uso común. Un ejemplo de secuencia de entidades que derivan en la categoría de "Muy Inseguro" puede ser la siguiente: Pedido de ayuda - Saludo - Pedido de ayuda - Insulto - Orden Amenazante. Este nivel representa un 100% (cien por ciento) de inseguridad.

Los porcentajes de inseguridad definidos en los ítems anteriores son meramente ilustrativos. Simplemente representan, de una escala del 0 al 100, una porción de la barra que expresa el nivel de inseguridad en la interfaz gráfica del prototipo. Estos porcentajes son independientes de los números de entidades de riesgo.

5.3.2. Recolección de datos

Para llevar a cabo la recolección de datos, se desarrolló un algoritmo sencillo en Python, que genera pseudo-aleatoriamente frases, tales como las mencionadas en los ejemplos de la sección anterior. Estas frases son de diferentes longitudes, es decir, poseen distintas cantidades de entidades para cada frase. El rango de generación de entidades para cada frase se estableció de 1 a 10.

Se decidió no establecer una longitud fija por cada frase debido a que el módulo NER no siempre devuelve el mismo número de entidades. De esta manera, la salida del primer módulo (NER) se adapta mejor a la entrada del segundo módulo (TEXTCAT).

Otro aspecto importante a mencionar, alusivo a la recolección de datos, es el hecho de que las entidades que conforman cada frase del dataset tienen una determinada probabilidad de ocurrir. Esto es porque si se deja librada su generación al azar (sin probabilidades predeterminadas), todas las frases tendrían entidades de inseguridad, imposibilitando etiquetar frases con los niveles "Muy seguro" y "Seguro".

El código fuente de este sencillo algoritmo consiste básicamente en dos estructuras *for* anidadas. La primera itera sobre la cantidad de filas (frases del dataset), y la segunda sobre la cantidad de entidades por cada frase (o columnas, si se piensa en la estructura de datos como una matriz).

La cantidad de columnas (entidades) es un número aleatorio entre 1 y 10. Dentro del *for* más interno, lo primero que se hace es seleccionar una entidad para concatenarla a la frase actual. Esta elección de la entidad se realiza en base a probabilidades, donde las entidades de

uso común tienen tres veces más posibilidades ocurrir que las de inseguridad. De esta manera, se van concatenando progresivamente las entidades y, cuando se alcanza el máximo (el número aleatorio entre 1 y 10, obtenido anteriormente), se pasa a la siguiente frase. Esto último lleva a la iteración del *for* más externo. Luego, cuando se terminó de ejecutar el algoritmo, se generó un archivo de texto plano (de extensión *.txt*) con 5000 (cinco mil) frases.

A continuación, se muestra un pseudocódigo para el algoritmo mencionado anteriormente (Figura 10).

```
Algoritmo generarDataset
    Crear Archivo Dataset.txt
    cantidadFilas = 5000
    frase = " "
    Desde 1 hasta cantidadFilas
        cantidadColumnas = NumeroAleatorio(1, 10)
        Desde 1 hasta cantidadColumnas
            entidad = elegirEntidadSegunProbabilidad()
            frase = frase + entidad
            Si ultimaColumna Entonces
                AgregarAlArchivo(frase)
                frase = " "
        FinDesde
    FinDesde
    Cerrar Archivo Dataset.txt
FinAlgoritmo
```

Figura 10. Pseudocódigo para el algoritmo generarDataset

Finalmente, se quitan las frases repetidas, que siempre las habrá, debido a la forma en que se genera el archivo en cuestión. Para lograr esto, se recurrió a la herramienta PineTools (véase sección "4.3.7. PineTools" para más detalles sobre esta herramienta). De esta manera, luego de extraer las frases repetidas, se pasó de 5000 (cinco mil) frases a aproximadamente 3000 (tres mil), listas para ser etiquetadas.

5.3.3. Etiquetado

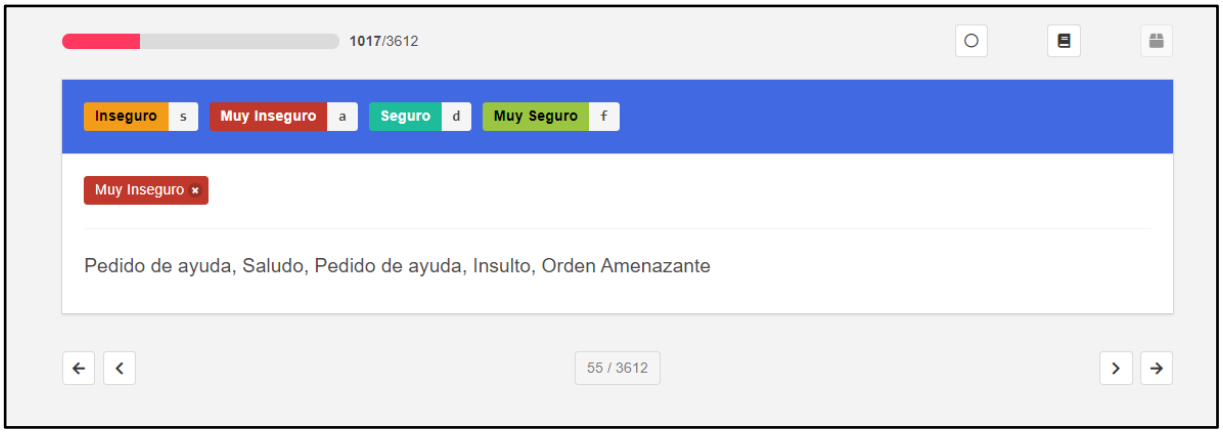
A diferencia del dataset de NER, en este caso se etiquetan las frases enteras, con alguna de las categorías mencionadas en la sección "5.3.1. Definición de niveles de inseguridad".

Para asignarle una etiqueta a cada frase, se intentó quitarle la mayor abstracción posible a la misma, pensando en la secuencia de entidades de dicha frase como si ocurriera en un escenario de la vida real. Por ejemplo, si se presenta una orden amenazante, seguida de un pedido de disculpas, entonces es posible que la persona haya pedido algo con cierto nivel de

intensidad (y no precisamente con los mejores modales), pero luego se retractó y pidió disculpas por lo ocurrido. Esto deriva, entonces, en una frase etiquetada como “Segura”.

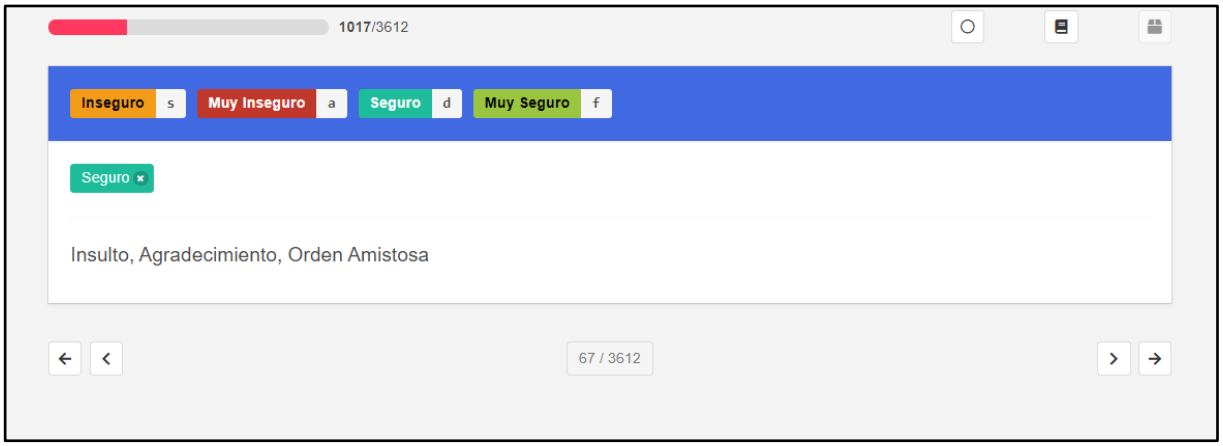
Muchas veces, debido a la pseudo-aleatoriedad, ocurrió que ciertas frases no tienen sentido, semánticamente hablando (por ejemplo, al existir un pedido de ayuda seguido de un saludo). De igual manera, estas frases fueron etiquetadas, pues estas pueden ser consideradas como la continuación de otra frase que justo fue “cortada”.

Se muestran a continuación algunas capturas del etiquetado de estas frases (Capturas 5 a 8), extraídas del archivo de texto plano mencionado anteriormente. Para etiquetar estas frases se hizo uso de la aplicación Doccano.

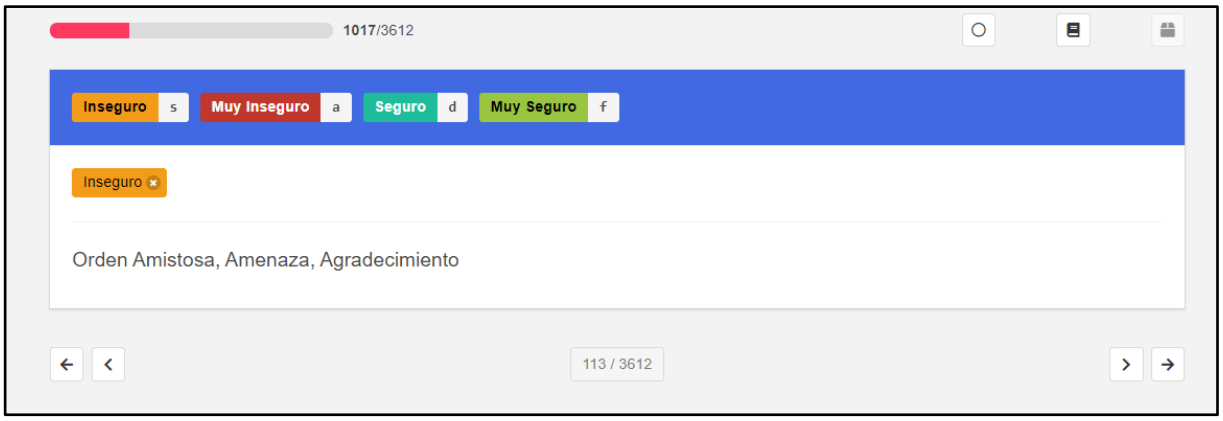


Captura 5. Ejemplo de una frase etiquetada con el label “Muy inseguro”

Captura 6. Ejemplo de una frase etiquetada con el label “Muy seguro”



Captura 7. Ejemplo de una frase etiquetada con el label "Seguro"



Captura 8. Ejemplo de una frase etiquetada con el label "Inseguro"

Las estadísticas correspondientes a este dataset se encuentran en la sección "11.1. Anexo A" del presente informe.

6. Entrenamiento de módulos

Ahora bien, una vez conformados los datasets descritos en la sección anterior, el siguiente paso de suma importancia para la continuidad del PFC es el de entrenar los módulos existentes. En función del nivel de exhaustividad de dicho entrenamiento, y de la convención empleada para etiquetar el sistema, resultará en un determinado nivel de aprendizaje del prototipo propuesto.

Por tanto, el objetivo de esta sección es explicar la metodología empleada para entrenar los módulos, aspecto crucial para el desarrollo del proyecto, ya que los mismos determinarán el funcionamiento de la herramienta de seguridad ciudadana.

Para llevar a cabo el proceso de entrenamiento, se sigue haciendo uso de spaCy. Como punto de partida, se detalla cual es el proceso general a seguir para poder entrenar cualquier módulo de spaCy . Seguidamente, se explica cómo fueron entrenados los módulos elegidos (NER y TEXTCAT), haciendo especial hincapié en la configuración de sus parámetros. Por último, se muestran los resultados obtenidos para cada módulo, a través de la precisión obtenida.

6.1. Metodología

spaCy provee una serie de módulos previamente entrenados, que pueden ser utilizados de manera directa, sin mayores complicaciones. Estos poseen una gran precisión y se adecuan a varias situaciones. Aun así, estos módulos no se adaptan del todo a cada problemática existente. Por esta razón, cada módulo de spaCy puede ser personalizado para adaptarlo a cada circunstancia. Esta personalización se logra con un entrenamiento de origen propio. Es importante mencionar que cuando decimos “entrenar un módulo” nos referimos a entrenar el modelo asociado a dicho módulo.

Los modelos de spaCy son estadísticos, y cada resultado que este brinda se obtiene en base a predicciones. Cada predicción se logra gracias a una serie de ejemplos que fueron otorgados al modelo durante su entrenamiento. Cada ejemplo consiste en una tupla con los siguientes valores: un texto (la entrada propia), y una etiqueta (resultado esperado).

Durante su entrenamiento, el modelo hará predicciones en base a cada entrada y comparará si las mismas concuerdan con el resultado esperado (obtenido por medio de las etiquetas), generando así un gradiente de error. Este valor indica qué tan preciso es el modelo a entrenar.

Por otro lado, es importante mencionar que, al entrenar un modelo, no solo se busca que el modelo memorice los ejemplos que se le provean, sino que además sea capaz de crear una teoría que le permita generalizar el resultado esperado para un gran número de situaciones. Esto solo es posible si los datos de entrenamiento son representativos para el problema que fue planteado. Por lo tanto, al entrenar un modelo, no solo son necesarios datos de entrenamiento, sino también datos que permitan evaluarlo. Si se lo evalúa con los mismos datos con los cuales fue entrenado, no se podrá saber que tan bien generaliza el modelo.

6.1.1. Entrenamiento en spaCy

A continuación, se ofrece una breve explicación de cómo funciona el mecanismo para entrenar un modelo de spaCy, y qué aspectos se deben tener en cuenta para lograr una buena precisión en el modelo.

Cualquier conjunto de datos que se utilice para entrenar, puede ser dividido en dos partes: la primera corresponde al texto, que son los ejemplos que se han recopilado y que representan los datos de entrada; la segunda, a sus etiquetas, que representan la salida esperada que el modelo debe predecir (Figura 11).

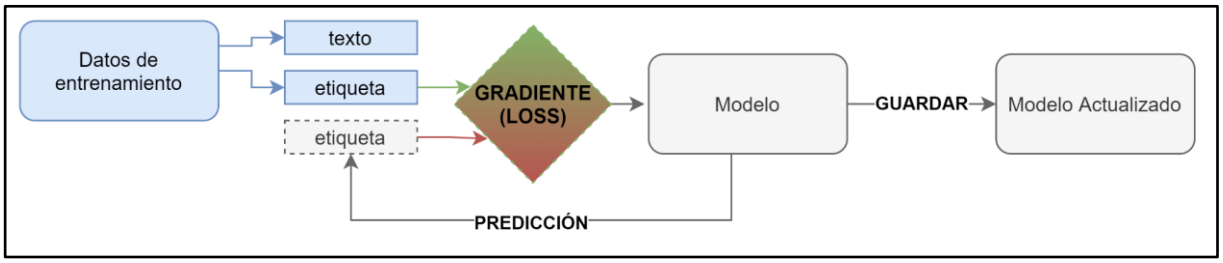


Figura 11. Proceso de entrenamiento de un modelo de spaCy

Antes de comenzar a entrenar, spaCy crea un objeto llamado *GoldParse* (Figura 12), que contiene las dos partes que se mencionaron anteriormente: el texto (que a partir de ahora conforma un objeto llamado *Doc*), y sus etiquetas. El objetivo del *GoldParse* es codificar los datos de entrenamiento en una estructura que sea de acceso rápido y eficiente.

En este punto, spaCy entrena el modelo (función *nlp*), actualizando (función *update*) su estado entre cada iteración, y guardando el estado del modelo entre cada iteración, gracias a una función llamada *optimizer (Figura 12)*.

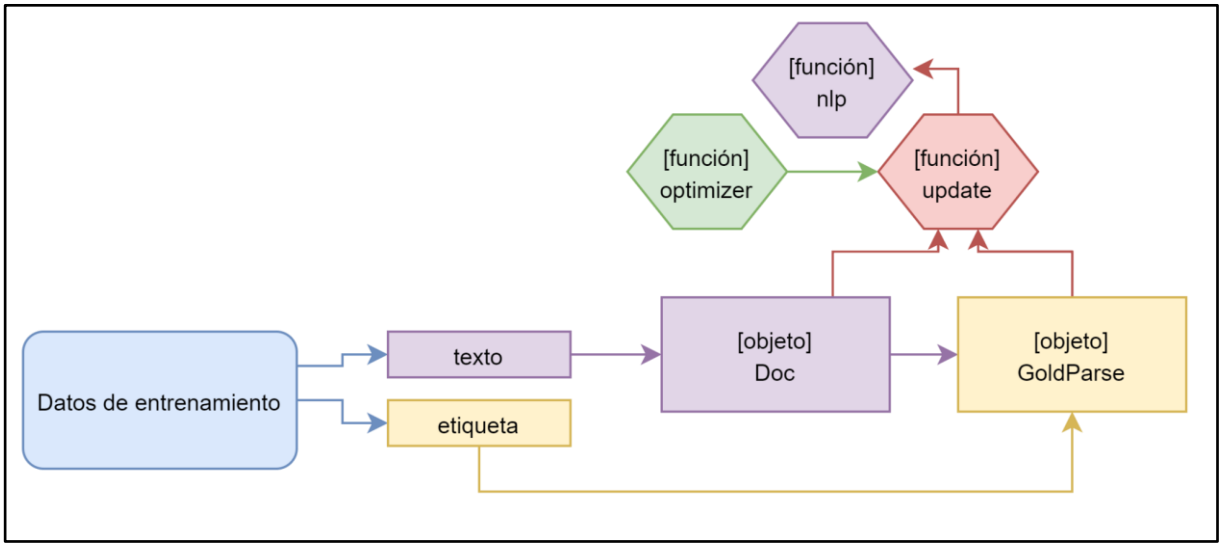


Figura 12. Entrenamiento en spaCy

Siempre se busca obtener la mejor precisión al momento de entrenar un modelo, para que las predicciones del mismo sean lo más acertadas posible. Así, no solo es necesario proveer un número considerable de datos de entrenamiento, sino también configurar ciertos parámetros para que el entrenamiento sea óptimo.Se listan a continuación los parámetros utilizados:

- **Número de iteraciones**. Cantidad de veces en la cual el modelo es entrenado. Un número muy bajo de iteraciones puede lograr que no se alcance el valor óptimo de precisión, mientras que un valor muy alto puede llevar a que el modelo se sobreentrene, y así imposibilitar que generalice correctamente. Entre cada iteración, los datos de entrenamiento son mezclados para evitar que el modelo aprenda en base al orden de los mismos.
- **Dropout rate**. Ritmo con el cual el modelo olvida (*drop*) ciertas características aprendidas. Esto hace que sea más difícil para el modelo memorizar los datos de entrenamiento. Por ejemplo, un *dropout rate* con un valor de 0.25 significa que, en cada iteración, el modelo olvida un 25% de lo aprendido.
- **Batch size**. Número de datos que se consideran en cada iteración para entrenar el modelo. Por ejemplo, si se tienen 1250 datos de entrenamiento, y se define un *batch size* de 100, entonces en la primera iteración se toman los primeros 100 datos de los 1250,

luego en la segunda iteración los siguientes 100 datos, y así sucesivamente. Si se elige un número bajo para el *batch size*, se necesitará menos memoria en cada iteración, y por lo tanto el modelo se entrenará más rápido. La desventaja de esto es que si el número es muy bajo, la precisión del modelo no será muy buena.

Ahora bien, ante una predicción de un modelo, esta puede caer dentro de una de las cuatro categorías disponibles de acuerdo a su relación con el valor de verdad. Simplificando a un problema de tipo binario, se pueden tener las siguientes categorías (Tabla 4):

- TN (*true negative*). Se predijo como falso, y la realidad coincide con la predicción.
- FP (*false positive*). Se predijo como verdadero, siendo que en la realidad es falso.
- FN (*false negative*). Se predijo como falso, siendo que en la realidad es verdadero.
- TP (*true positive*). Se predijo como verdadero, y la realidad coincide con la predicción..

		Predicción	
		0	1
Realidad	0	TN	FP
	1	FN	TP

Tabla 4. Tipos de predicciones obtenidas

Por último, para saber cuándo es necesario detener el entrenamiento y conocer qué tan bien va a predecir nuestro modelo, spaCy brinda los siguientes valores:

- **Loss.** Mide la diferencia entre el valor real y la predicción. En cada iteración, este valor se modifica y lo ideal es que decremente, acercándose lo más posible al 0. Este valor es adimensional.
- **Precision**. Indica la calidad del modelo obtenido. Se relaciona con el FP y TP, en el sentido de que responde a la siguiente pregunta: ¿qué porcentaje de lo que se predijo como clase positiva en realidad lo es?
- **Recall**. Valor que se relaciona con el FN y TP, es decir, responde a la pregunta: ¿qué porcentaje de la clase positiva se predijo como tal?
- **F-Score**. Combina los valores de precision y recall en uno solo.

6.2. Entrenamiento para el módulo NER

Como se mencionó en la sección “5.2. Dataset para el módulo NER”, NER permite reconocer distintas entidades en una entrada de texto, bajo la cual su eficiencia dependerá de los parámetros a ser considerados. Estos mismos son los que se detallan en la presente sección.

6.2.1. Configuración de parámetros

A continuación, se mencionan los valores que se escogieron para cada uno de los parámetros posibles, así como la justificación de dicha elección. Estos parámetros son configurables al entrenar un módulo NER de spaCy.

- **Número de iteraciones:** 80.-

Para estimar el número de iteraciones, se eligió inicialmente un número alto (100) y se sacaron conclusiones en cuanto al valor de *Loss*. Se observó que, más allá de un determinado número (80), el valor de *Loss* no cambiaba. Por tanto, el módulo se sobreentrenaba, perdiendo así efectividad.

- **Dropout rate:** 0.3.-

El valor de *Dropout* indica cual es el porcentaje de pérdida de aprendizaje del módulo en cada iteración de entrenamiento. Para elegir el correspondiente valor de *Dropout*, nuevamente se recurrió al *Loss*. Mientras se entrenaba el módulo, se analizaba constantemente dicho valor. Si en pocas iteraciones el valor de *Loss* se estancaba, entonces se aumentaba el valor de *Dropout*. Bajo estas condiciones, se arribó a un valor de 0.3 (30%).

- **Batch size**: mínimo = 4, máximo = 32.-

Se utilizó una función propia de spaCy llamada *Compounding*. Esta función se caracteriza por tomar inicialmente el valor mínimo elegido de *batches* (cantidad de frases etiquetadas a considerar), y en cada iteración ir incrementando dicho valor mínimo hasta llegar al máximo, sin tomar nunca un valor superior. Al llegar al máximo, las siguientes iteraciones utilizan este valor. Los valores que se eligieron aquí (mínimo = 4, máximo = 32) fueron escogidos luego de diversas pruebas. Los mejores resultados fueron aquellos en los que se obtuvo mayor precisión.

Utilizando una configuración diferente a los valores mencionados anteriormente, la precisión resultante era menor.

6.2.2. Proceso de entrenamiento

A continuación, se puede visualizar gráficamente las salidas de información por cada iteración realizada (Figura 13), donde cada valor corresponde a un valor de Loss en cada iteración del entrenamiento. Estos valores de Loss obtenidos en la gráfica, corresponden a un entrenamiento cuya configuración de parámetros corresponde a la mencionada en la sección anterior ("6.2.1. Configuración de parámetros"). Si se emplea otra configuración de parámetros, la gráfica resultante será distinta.

Tal como se mencionó en la sección "6.1.1. Entrenamiento en spaCy", el valor de Loss mide, para una iteración en específico, la diferencia entre las predicciones hechas por el modelo en ese momento y el valor real. En otras palabras, el Loss indica que tan mal resultó la predicción al momento de entrenar el modelo.

Si el valor de Loss es 0, significa que la predicción fue perfecta, ya que coincide con el valor real de la iteración. Sin embargo, este valor de 0 no deja de ser un ideal teórico, pues en la práctica resulta imposible alcanzarlo.

En términos del modelo, el Loss permite al mismo estimar qué tan bien está aprendiendo y, naturalmente, lo ideal es que dicho valor tienda a cero. Por tanto, se busca que este valor decremente en cada iteración, lo cual implicaría que el modelo mejoró con respecto a la iteración anterior. Si este valor permanece constante u oscila, se corre el riesgo de que el modelo quede sobreentrenado. Si esto ocurre, dicho modelo no responderá correctamente ante entradas nuevas.

Para el caso del presente proyecto, el gráfico resultante que corresponde al entrenamiento de NER cumple con lo especificado anteriormente: comienza con un valor de Loss alto y, a medida que las iteraciones avanzan, este valor va decrementándose y tendiendo a cero. Además, como se puede ver, los valores de Loss en las últimas iteraciones son muy similares entre sí, por lo que, si se continuaba entrenando el modelo, posiblemente se podría haber arribado a un caso de sobreentrenamiento.

Figura 13. Loss contra iteración para el módulo NER

Como se observa en la Figura 13, el valor obtenido en la última iteración es, aproximadamente, el valor promedio de las últimas 10 iteraciones. Si se aumentan las iteraciones, estas seguirán un patrón ondulatorio sobre este valor obtenido.

6.2.3. Resultado del entrenamiento

spaCy provee una clase denominada Scorer que permite evaluar la precisión de un módulo entrenado. Por tanto, haciendo uso de esta clase, se obtuvieron diferentes métricas. Para el módulo NER, se alcanzó un porcentaje de precisión de dicho módulo del 62,2%, un F-Score del 60,18% y un Recall del 58,29%.

A continuación, se muestran los valores de F-Score, Precision y Recall por cada entidad considerada en el módulo (Tabla 5).

Entidad	F-Score	Precision	Recall
Agradecimiento	77,42	75	80
Amenaza	45,45	50	41,67
Aprobación	64,29	60	69,23
Insulto	73,33	78,57	68,75
Orden Amenazante	43,75	46,67	41,18
Orden Amistosa	43,75	50	38,89
Pedido de Ayuda	60	60	60
Pedir Disculpas	73,47	75	72
Rechazo	42,86	42,86	42,86
Saludo	64,87	70,59	60
PROMEDIOS	**60,18**	**62,2**	**58,29**

Tabla 5. Valores de F-Score, Precision y Recall para cada entidad

Por otro lado, se volcaron los datos de la tabla en el siguiente gráfico (Figura 14), con fines de tener una comparación visual de los valores, para cada una de las entidades utilizadas.

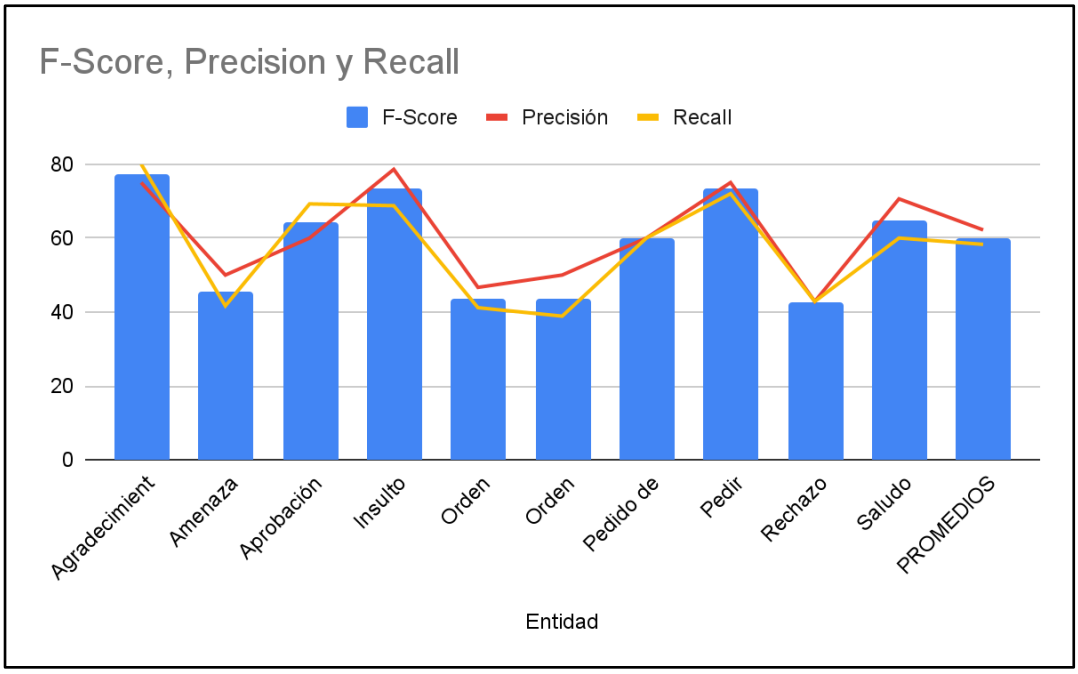


Figura 14. Gráficos de F-Score, Precision y Recall para cada entidad

6.3. Entrenamiento para el módulo TEXTCAT

Asegurando cierta precisión en el módulo TEXTCAT, se puede obtener un mejor o peor resultado al momento de determinar si se trata de una situación particular de tipo segura, muy segura, insegura o muy insegura, según los parámetros elegidos. A continuación, se definen dichos valores.

6.3.1. Configuración de parámetros

Se detallan los valores escogidos para cada uno de los parámetros posibles, configurables al entrenar un módulo TEXTCAT de spaCy.

- **Número de iteraciones:** 30.-

Nuevamente, comparando con el entrenamiento del módulo NER, se escogió un número alto de iteraciones (100), y se fue bajando progresivamente, hasta que el valor de *Loss* ya no se encontrara estancado. Se pudo observar que, a partir de las 30 iteraciones, el valor de Loss comenzaba a estancarse y a perder efectividad.

- **Dropout rate:** 0.2.-

Al mismo tiempo que se entrenaba el módulo, se observaba constantemente el valor de *Dropout*. Si en pocas iteraciones este valor se estancaba, entonces se aumentaba el valor de *Dropout*. Así, se obtuvo un valor de 0.2 (20%).

- **Batch size:** mínimo =1 , máximo = 32.-

Para determinar el valor de este parámetro, se utilizó la misma función que en la sección anterior, *Compounding*. En este caso, el valor mínimo es de 1 y el máximo de 32.

6.3.2. Proceso de entrenamiento

A continuación, se puede visualizar en el siguiente gráfico (Figura 15) las salidas de información por cada iteración realizada. En dicho gráfico, a cada iteración le corresponde un valor de Loss, correspondiente a un entrenamiento en particular.

Tal como se explicó en la sección “6.2.2. Proceso de entrenamiento” (NER), el objetivo es lograr que el valor de Loss se decremente en cada iteración, evitando que se estanque en cierto

valor o que oscile. Durante este entrenamiento, se comienza con un valor de Loss alto, pero en pocas iteraciones este valor decrementa rápidamente (a diferencia del entrenamiento de NER, que fue más lento). En definitiva, esto significa que a este modelo le resultó más fácil aprender, en comparación con el modelo de NER.

Es importante destacar que, para la gráfica correspondiente a la Figura 15, puede interpretarse erróneamente que el valor de Loss permanece constante a partir de la quinta iteración. Esto ocurre simplemente porque se requirió que la escala utilizada para el eje vertical (Loss) sea lo suficientemente grande para mostrar el valor de la primera iteración (cercano a 600).

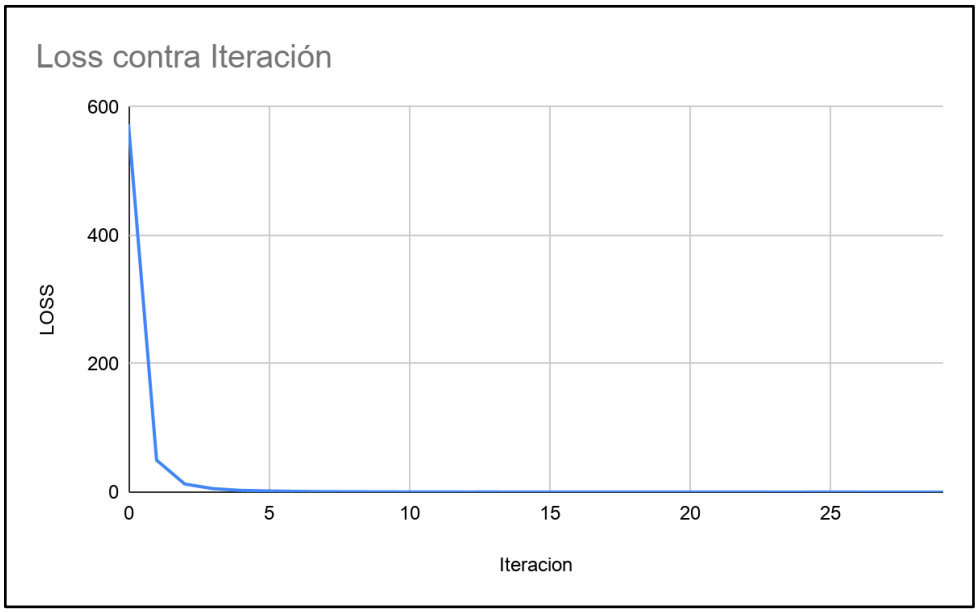


Figura 15. Loss contra iteración para el módulo TEXTCAT

6.3.3. Resultado del entrenamiento

Nuevamente, tal como en el caso de NER, se volvió a emplear la clase Scorer para obtener la precisión del modelo de TEXTCAT. Como se observa en la Tabla 6, se tienen los valores de F-Score, Precision y Recall para cada una de las categorías, y un valor promedio de 71.5% para el caso de F-Score, 75.7% para el caso de Precision, y 67.8% para el caso de Recall.

Categoría	F-Score	Precision	Recall
Muy Seguro	93.46	90.91	96.15
Seguro	68.24	82.86	58
Inseguro	54.17	46.43	65
Muy Inseguro	79.31	82.14	76.66
PROMEDIOS	**71.5**	**75.7**	**67.8**

Tabla 6. Valores de F-Score, Precision y Recall para cada categoría

Por otro lado, los datos de la Tabla 6 se volcaron en la siguiente gráfica (Figura 16), para visualizarlos de forma más clara.

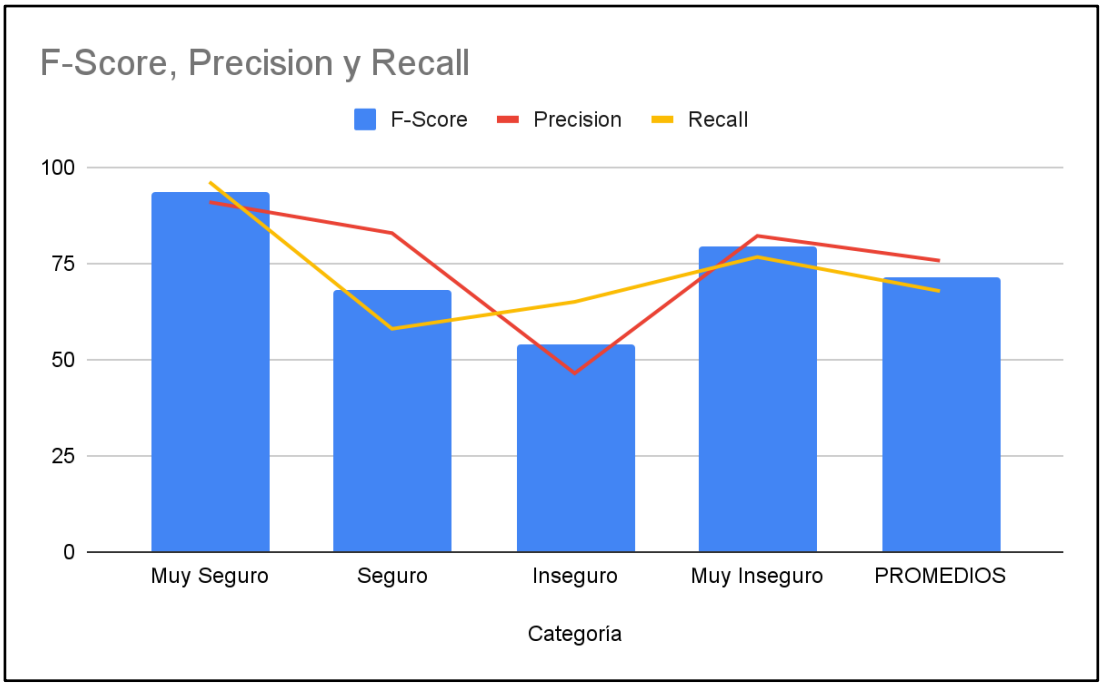


Figura 16. Gráficos de F-Score, Precision y Recall para cada categoría

7. Herramienta de software para la detección de hechos de inseguridad

En esta sección se explicará el funcionamiento de la herramienta de software desarrollada durante el presente PFC. Además, se detallará cada una de sus partes, para poder obtener un total entendimiento de la misma.

Inicialmente, se definen los requerimientos del sistema, tanto funcionales como no funcionales. Luego, se detalla la arquitectura que tiene el aplicativo, en conjunto con un diagrama de clases de entrenamiento, y otro de la herramienta propiamente dicha. Más adelante, se adjuntan algunas capturas de la interfaz gráfica del software. Finalmente, se listan los parámetros seleccionados, sus valores y la justificación de los mismos. Asimismo, se provee un hipervínculo al repositorio online donde se encuentra alojado el código fuente del proyecto.

Conceptualmente, y a modo introductorio, la secuencia de pasos ejecutados por el prototipo se puede reducir a los siguientes puntos:

1. Una frase es ingresada al sistema.
2. Un módulo NER se encarga de dividir la frase en un conjunto de entidades, en función de su contenido.
3. Las entidades se almacenan en un lote de entidades mientras el temporizador asociado lo disponga o hasta que el mismo se llene, lo que ocurra primero.
4. El lote es enviado a un segundo módulo (TEXTCAT), que analiza el lote y determina su nivel de inseguridad.

7.1. Requerimientos

7.1.1. Requerimientos funcionales

Los requerimientos funcionales describen las características y funciones que un sistema debe brindar, y establecen la forma en que reaccionará ante ciertos estímulos.

Para el componente de software desarrollado en este PFC, los requisitos funcionales fueron los siguientes:

- Detectar hechos de inseguridad a partir de entradas de texto.
- Medir el nivel de peligro de las situaciones de inseguridad.
- Analizar situaciones en tiempo real.

7.1.1.1 Diagramas de casos de uso

En base a los requisitos funcionales enunciados anteriormente, se puede arribar a un único caso de uso abarcativo de los mismos, el cual se encargue de detectar un hecho de inseguridad particular.

En definitiva, para los límites de este proyecto, existirá un rol de usuario único, asociado a un único caso de uso, definido como “Detectar hecho de inseguridad” (Figura 17).

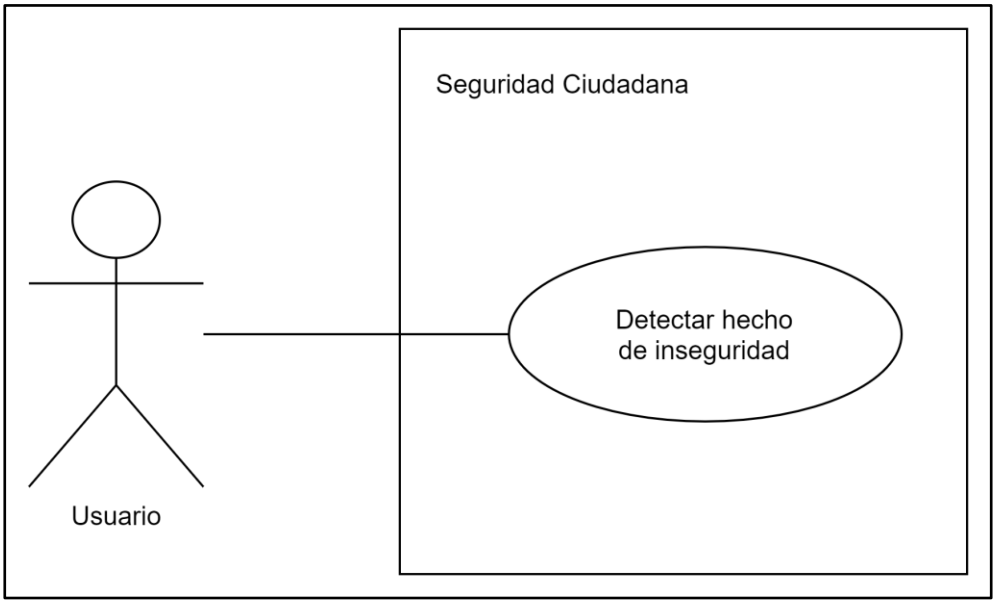


Figura 17. Diagrama de casos de uso del prototipo de software

7.1.1.2 Especificacion de casos de uso

A continuación, se define el único caso de uso del proyecto a través de su Especificación de caso de uso. Éste define el objetivo de dicho caso de uso, las precondiciones y postcondiciones asociadas, y los flujos principales y alternativos del mismo (Tabla 7).

<table>
<tr><td colspan="2">Proyecto: Diseño e implementación de un agente inteligente capaz de clasificar situaciones de inseguridad mediante técnicas de Machine Learning y de Procesamiento del Lenguaje Natural</td></tr>
<tr><td>Nombre de CU: Detectar hecho de inseguridad</td><td>CU:01</td></tr>
<tr><td colspan="2">Objetivo: Detectar un hecho de inseguridad a partir de una entrada de texto.</td></tr>
<tr><td colspan="2">Actores: Usuario</td></tr>
<tr><td colspan="2">Precondiciones: Ninguna.</td></tr>
<tr><td colspan="2">Postcondiciones: Ninguna.</td></tr>
<tr><td colspan="2">Prioridad: Alta</td></tr>
<tr><td>Flujo Principal</td><td>Flujo Alternativo</td></tr>
<tr><td>La herramienta recibe una entrada de texto para ser analizada.
La frase es analizada por el módulo NER y es descompuesta en cero o más entidades.
Se almacenan las entidades en un lote y se setea el temporizador en cero.
El temporizador se vence y se envían las entidades almacenadas a analizar por el módulo TEXTCAT.
El módulo TEXTCAT analiza el lote de entidades e informa el nivel de inseguridad.
Se espera por una nueva entrada de texto.</td><td>3.1 El lote de entidades se llena y se envía al módulo TEXTCAT. Se genera un nuevo lote con las entidades que no pudieron agregarse al lote anterior. Se continúa con el paso 5 del flujo principal.

4.1. Se recibe una nueva entrada de texto antes de que el temporizador finalice. Se continua con el paso 2 del flujo principal.</td></tr>
<tr><td colspan="2">Flujos de excepción: El usuario en cualquier momento puede presionar el botón cancelar haciendo que el CU termine.</td></tr>
<tr><td colspan="2">Observaciones: La herramienta puede recibir una entrada de texto que fue ingresada manualmente por el usuario o bien leída de un archivo de subtítulos de forma automática.</td></tr>
</table>

Tabla 7. Especificación CU01: Detectar hecho de inseguridad

7.1.2. Requerimientos no funcionales

Los requerimientos no funcionales no se refieren a los servicios que un sistema debe proveer, sino a características o propiedades que debe satisfacer. Se los conoce también como atributos de calidad del sistema.

Los requisitos no funcionales para el presente software prototipo son los siguientes:

- Para evitar problemas de privacidad de datos, el algoritmo debe poder correr localmente.
- Debe ser gratis en cualquier escenario de uso. Por lo tanto, para su desarrollo, se tiene que recurrir a librerías y tecnologías de código abierto.
- Debe ser confiable. Esto implica que se deben minimizar las ocurrencias de falsos positivos y de falsos negativos.
- Los períodos de tiempo de respuesta del sistema deben ser en tiempo real.

7.2. Arquitectura

La herramienta desarrollada, como se mencionó con anterioridad, cumple el siguiente propósito: analizar los diálogos que ocurren entre las personas, y clasificar dicha situación de acuerdo a su nivel de riesgo. Por ejemplo, si se presenta una conversación tranquila entre dos amigos, el nivel de peligro será muy bajo; por el contrario, si ocurren diversas amenazas a causa de un delito, el nivel de inseguridad será elevado.

Es importante mencionar que el sistema no detecta situaciones específicas de inseguridad tales como robos, crímenes o violaciones, sino que las clasifica de acuerdo a su nivel de peligro en cuatro categorías, donde la más inofensiva es “Muy Seguro” y la más peligrosa es “Muy Inseguro”.

Por su parte, es necesario aclarar que la herramienta solo funciona con entradas de texto, y no con audio, es decir, el diálogo que se desee analizar deberá estar escrito. La detección vía audio está fuera del alcance de este PFC.

Una vez que se haya detectado una situación insegura o riesgosa, queda en manos de quien utilice el sistema en tomar las acciones adecuadas para mitigar el impacto de lo que está ocurriendo en ese momento.

En la figura 18 se pueden observar gráficamente los módulos de la herramienta, así como sus conexiones. Por un lado, existe el componente de seguridad ciudadana que enlaza los módulos y muestra los resultados al usuario. Luego, existen dos subcomponentes (NER y TEXTCAT) que se encargan de analizar y clasificar el texto ingresado. Cada uno de ellos funciona bajo modelos de ML previamente entrenados, utilizando para ello la biblioteca spaCy, como se detalló con anterioridad.

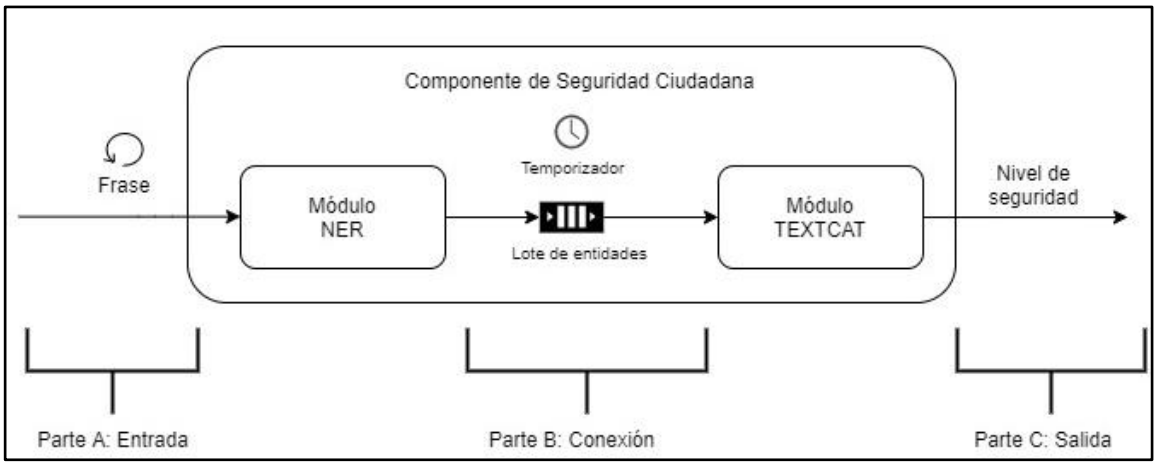


Figura 18. Módulos de la herramienta y sus conexiones

Cuando es ingresada una frase, el módulo NER se encarga de dividirla en un conjunto de entidades en función del contenido de la misma. Naturalmente, habrá frases a las cuales el prototipo no será capaz de descomponer en una o más entidades, y esto puede deberse a dos motivos, a saber:

- Los tipos de entidades que la herramienta identifica son limitados y no taxativos.
- Problemas de precisión. Debido a que la herramienta no es perfecta, puede haber casos donde ésta no detecte entidades aún cuando estén presentes.

A modo de ejemplo, se propone el siguiente escenario: si la frase es “Hola”, la entidad asociada es “Saludo”; si, en cambio, es “Te voy a matar”, entonces corresponde a una “Amenaza”. Las entidades son almacenadas en un lote de entidades, a la espera del siguiente paso.

El proceso descrito anteriormente se repite por cada frase ingresada y cada nuevo conjunto de entidades es almacenado en el lote mencionado. Bajo circunstancias especiales, que serán

descritas más adelante, el lote de entidades es enviado al siguiente módulo (TEXTCAT), en donde el mismo se analiza y se determina su nivel de inseguridad.

Gracias a este mecanismo, las frases de una conversación pueden analizarse en conjunto y no de manera aislada. Además, esta estructura brinda la posibilidad de obtener resultados en tiempo real, para así no esperar a que la charla finalice por completo para conseguirlos.

El componente de seguridad ciudadana está por encima de los demás, y cumple dos funciones esenciales:

- Permitir que los demás módulos se comuniquen y funcionen en conjunto para, de esta manera, poder obtener los resultados esperados.
- Posibilitar la interacción con el usuario, permitiendo el ingreso de los datos, así como también ir mostrando los resultados a través de una interfaz gráfica.

Como se puede ver en la Figura 18, la interacción entre los módulos puede dividirse en tres partes. A continuación, se explican cada una de ellas.

- Parte A. Se ingresa una cadena de texto como entrada al sistema, donde inmediatamente el módulo NER obtiene las entidades asociadas. Esta cadena de texto puede provenir de dos fuentes:
    - Ser ingresada manualmente por un usuario. En este caso, el usuario debe ingresar la conversación de forma secuencial, frase a frase.
    - Ser tomada de un archivo de texto plano (para el caso del PFC, este archivo contiene subtítulos de escenas). Si el archivo es importado, cada frase debe tener asociada un determinado tiempo, con el fin de "simular" una conversación en tiempo real.
- Parte B. Se almacenan las entidades resultantes del módulo NER para luego ser enviadas al módulo TEXTCAT, al cumplir una de las siguientes dos condiciones:
    - El lote de entidades se ha llenado. El tamaño del lote es un valor fijo, el cual establece la cantidad de entidades a analizar en cada iteración. Al alcanzar su límite, dicho lote se envía al siguiente módulo y se vacía, para dejar paso a la recepción de nuevas entidades.
    - El temporizador se ha vencido. Cada vez que una frase ingresa en el sistema, se inicia un temporizador, el cual indica si la conversación ha terminado, para poder

dar lugar al análisis de las entidades encontradas hasta ese instante. Cada frase nueva que ingresa al sistema reinicia dicho temporizador, marcando la continuidad de la conversación actual. Si no ingresa ninguna frase en el lapso determinado por el temporizador, se asume que la conversación culminó y se envía el lote, completo o incompleto, a analizar.

- Parte C. Al ser analizado por el módulo TEXTCAT, se muestra el nivel de seguridad de dicho lote, correspondiente a una de las cuatro (4) posibles categorías descritas con anterioridad.

En términos generales, se puede pensar que cada conversación que se quiera analizar es dividida en partes. Cada parte corresponde a un lote de entidades que se analiza, y así se obtiene su nivel de seguridad (Figura 19).

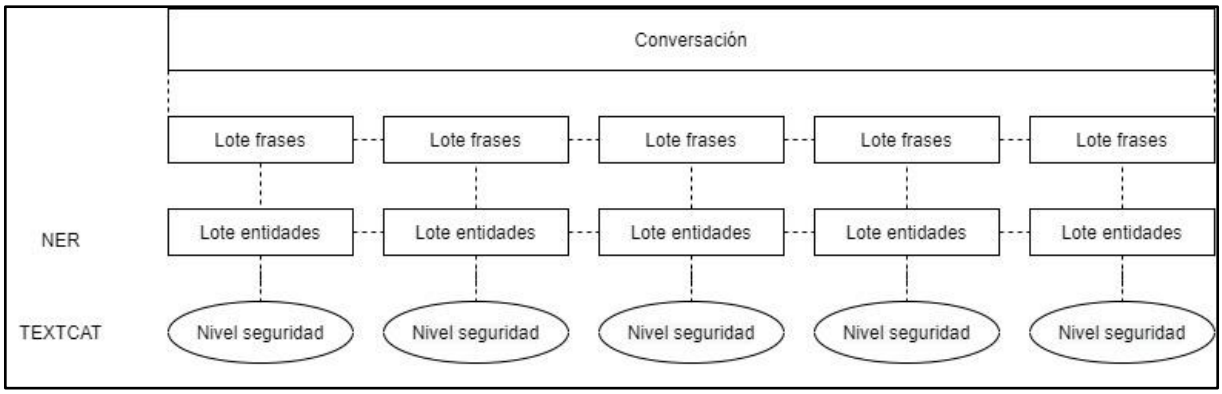


Figura 19. Lotes de entidades

Si bien este formato de ejecución no es incorrecto, puede ocasionar un problema al truncar un conjunto de frases: cuando las frases entre lote y lote dejan de estar relacionadas entre sí, se pueden obtener resultados inconsistentes o no deseados.

Para solucionar lo anterior, se recurrió al hecho de solapar cada lote de frases para que, de esta manera, haya una relación entre cada lote y así evitar la pérdida de coherencia en el diálogo. El solapamiento de frases consiste en que en cada lote de entidades se agregan las últimas entidades del lote anterior.

Dado el siguiente escenario, en el que una conversación posee las siguientes entidades: *Saludo - Amenaza - Insulto - Orden Amistosa - Insulto - Amenaza.* Si el tamaño de lote de entidades tiene un tamaño de 3, entonces, se tiene lo siguiente:

- Lotes sin solapamiento:
    - Lote 1: Saludo - Amenaza - Insulto.
    - Lote 2: Orden amistosa - Insulto - Amenaza.
- Lotes con solapamiento de dos entidades:
    - Lote 1: *Saludo - Amenaza - Insulto.*
    - Lote 2: *Amenaza - Insulto - Orden Amistosa.*
    - Lote 3: *Insulto - Orden Amistosa - Insulto.*
    - Lote 4: *Orden Amistosa - Insulto - Amenaza.*

Podemos observar que, al existir solapamiento, cada lote posee las últimas entidades del lote anterior, logrando que exista una relación entre cada uno de ellos, y de esta manera no perder el hilo de la conversación (Figura 20).

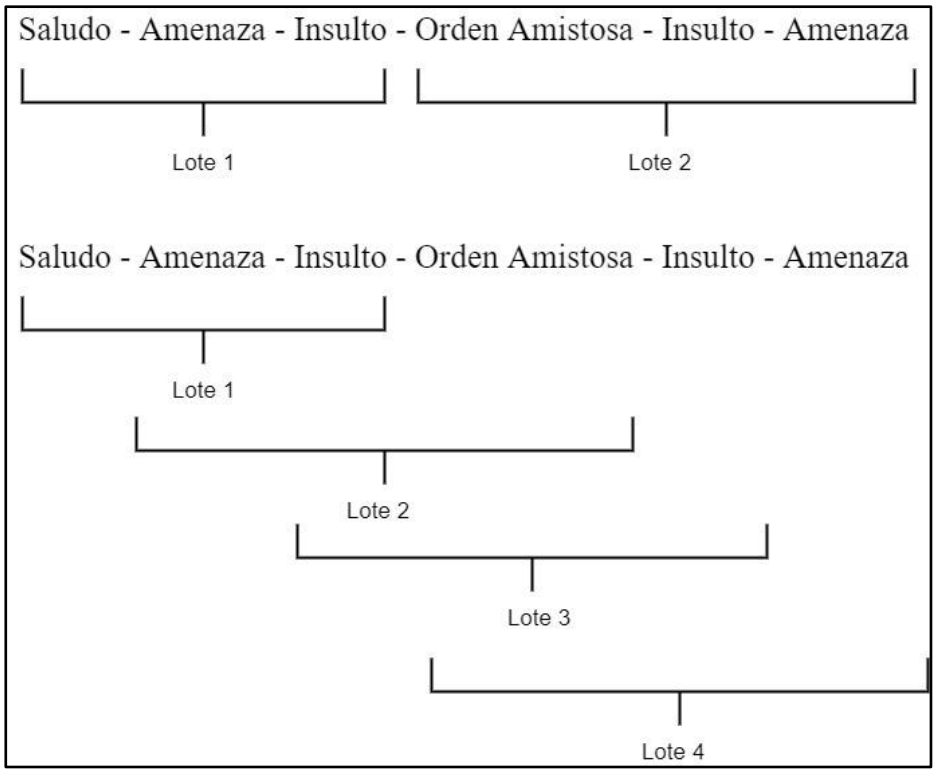


Figura 20. Solapamiento de entidades

En las siguientes dos subsecciones se muestran los diagramas de clases del presente proyecto. Es importante mencionar que son dos los diagramas (y no uno, como comúnmente se estila) ya que, por un lado, existen las clases encargadas de entrenar los módulos de NER y TEXTCAT y, por otro, se tienen las clases de la herramienta de seguridad ciudadana

desarrollada. Esta distinción se hace explícita porque el código relacionado al entrenamiento no se relaciona con el de la ejecución de la herramienta. Como ya se sabe, primeramente fue necesario entrenar los módulos y, una vez que se obtuvieron los mismos, el código relacionado al entrenamiento ya no se ejecutó. Luego, el código relacionado a la herramienta de seguridad ciudadana toma los módulos ya entrenados y los usa con el fin de detectar las situaciones de inseguridad.

7.2.1. Diagrama de clases de entrenamiento

En la figura 21 se puede observar el diagrama de clases para el proceso de entrenamiento. Por un lado, *EntrenadorNER* y *EntrenadorTEXTCAT* son las clases encargadas de entrenar a los módulos: cargan los datasets, y luego los emplean para entrenar y obtener la precisión (métodos *entrenarModeloNER*, *entrenarModeloTEXTCAT* y *obtenerPrecision*).

El resto de clases son bibliotecas o librerías empleadas por las dos clases mencionadas anteriormente. La más importante de ellas es la de spaCy, ya que contiene todo lo relacionado con el entrenamiento de cada módulo (los métodos *pipe* permiten entrenar los módulos elegidos), y además permite guardar en disco cada módulo (*to_disk*).

A continuación, se brinda una breve descripción del resto de las clases del diagrama:

- *Scorer*. Se utiliza para obtener la precisión del módulo NER. Este, a su vez, emplea la librería *GoldParse* para lograr este fin.
- *Utils*. Contiene métodos empleados para el manejo de los tamaños de los batches.
- GeneradorDatasetTEXTCAT. Contiene los métodos usados para crear el dataset, que se emplean para entrenar a TEXTCAT.

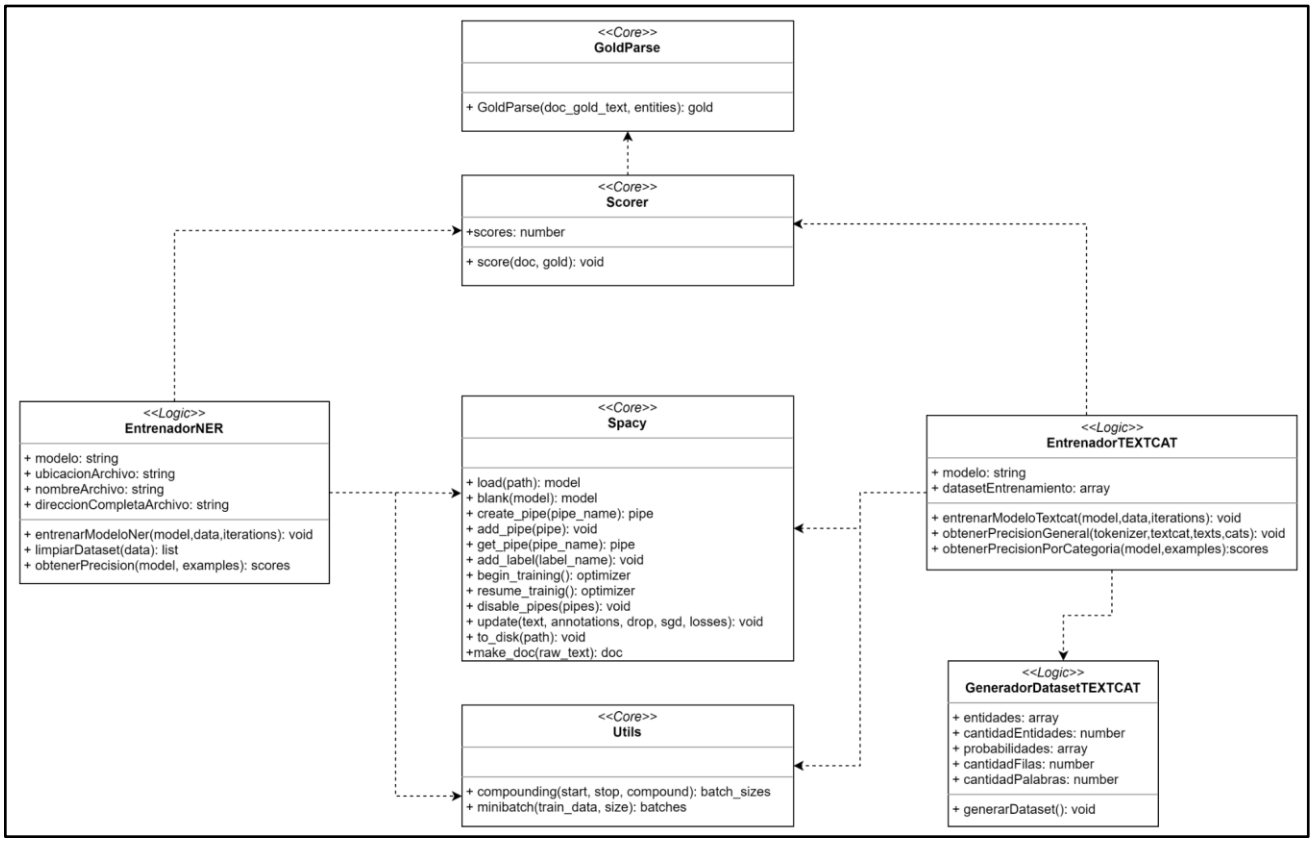


Figura 21. Diagrama de clases de entrenamiento

7.2.2. Diagrama de clases de la herramienta

A continuación, se ofrece una descripción de las clases empleadas para diseñar la herramienta (Figura 22):

- *SeguridadCiudadana*. Es la clase más importante del diagrama. Es el componente de seguridad ciudadana, por lo que se encarga de enviar el texto a cada módulo, y de la conexión entre ambos.
- *ChatView*. Es la interfaz del prototipo. Contiene los mensajes que se van imprimiendo en pantalla, así como el comportamiento de cada botón.
- *AppJar*. Es la biblioteca que brinda los componentes para crear las interfaces. Permite añadir botones, labels, tamaños, paddings, separaciones entre cada pantalla, etc.
- *Os*. Permite abrir el visor con los videos de cada escena utilizada en el prototipo para cada escena a analizar bajo la funcionalidad “demo en vivo”. Esto es meramente ilustrativo, ya que como bien se sabe, la herramienta solo analiza texto.
- *spaCy*. Permite cargar los módulos entrenados previamente.

- *Timer*. Provee los métodos utilizados para el temporizador.
- *Pysrt*. Se utiliza para cargar los subtítulos de cada escena a analizar bajo la funcionalidad de “demo en vivo”.

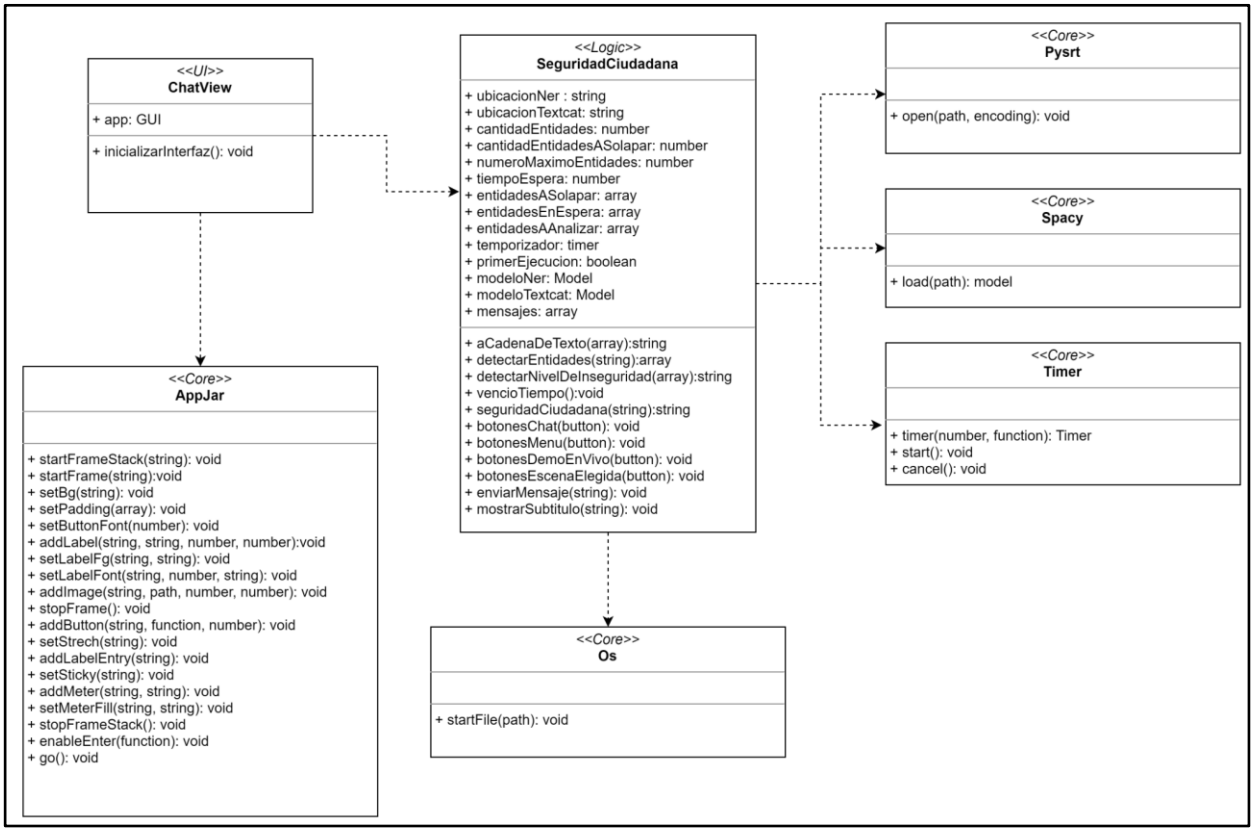


Figura 22. Diagrama de clases de la herramienta

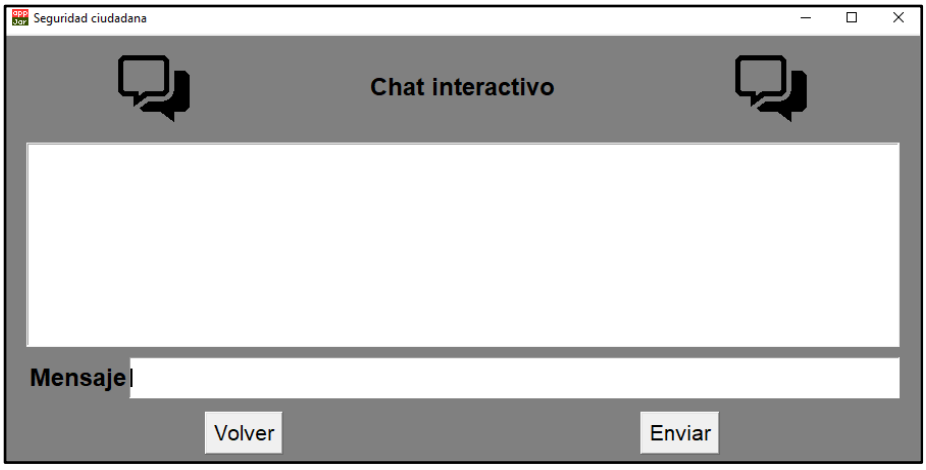
7.3. Interfaz gráfica

La interfaz gráfica posee un menú principal con dos (2) opciones, a saber:

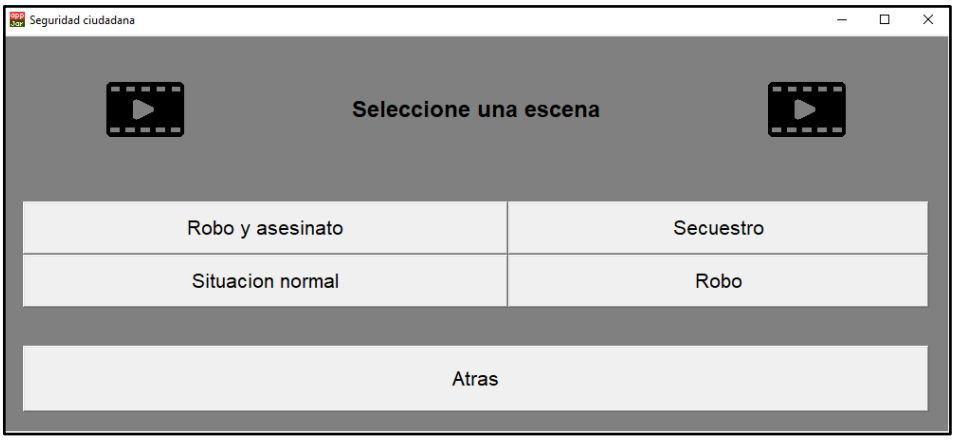
1. Un chat en vivo, en donde el usuario puede ir ingresando frase a frase.
2. Una opción de demo en vivo, donde se presentan videos de ejemplos de películas/series que se analizan en tiempo real.

A continuación, se adjuntan capturas tanto del menú principal (Captura 9) como de las dos opciones que ofrece la herramienta (Capturas 10 y 11).

Captura 9. Pantalla inicial de la herramienta



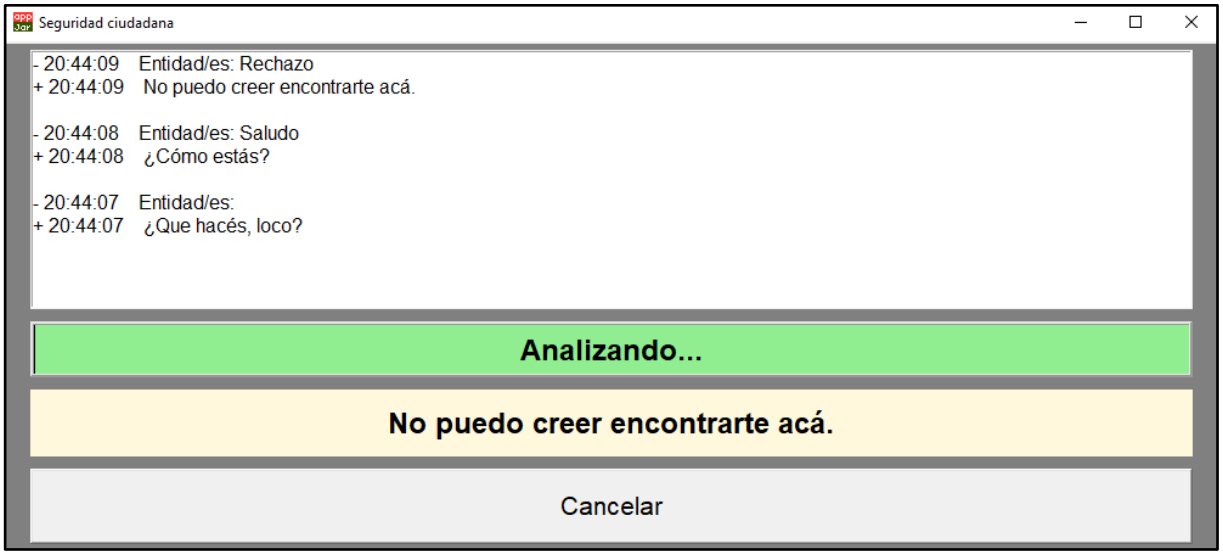
Captura 10. Pantalla de chat interactivo



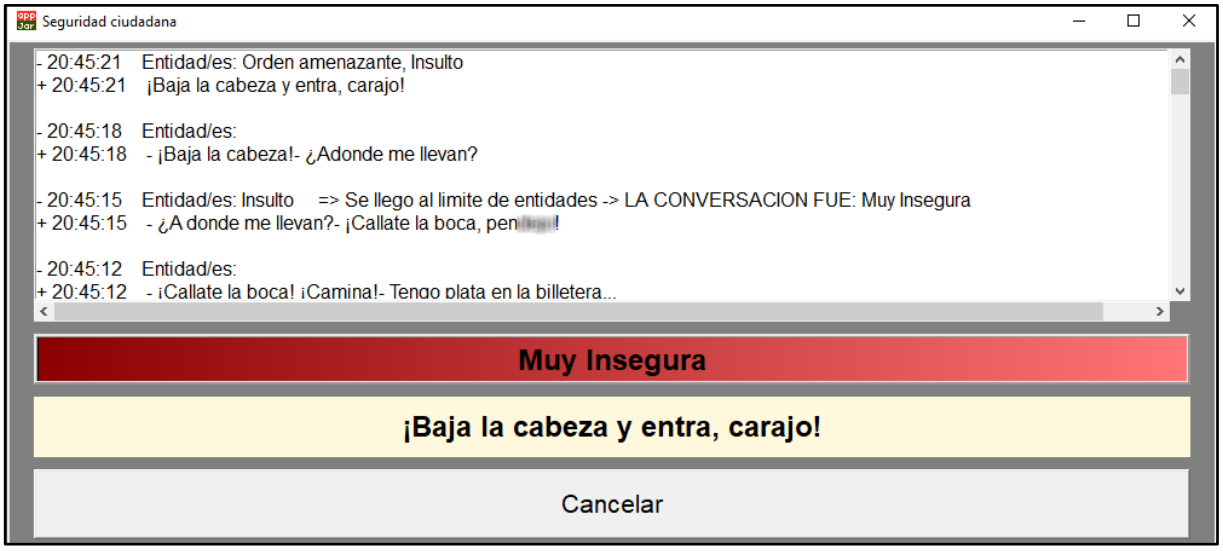
Captura 11. Pantalla del menú demo en vivo

Si se ingresa a alguna de las escenas, se abrirá el reproductor nativo del ordenador, con la escena seleccionada. Además, se mostrará por pantalla los diálogos de la escena a medida que

los personajes los van emitiendo, en tiempo real. Asimismo, se muestra un indicador del nivel de seguridad, que se actualiza a medida que se va analizando la conversación (Capturas 12 y 13).



Captura 12.Pantalla de ejecución de un caso de prueba



Captura 13. Indicador de nivel de inseguridad

7.4. Selección de parámetros

Para el correcto funcionamiento de la aplicación, fue necesaria la configuración de los siguientes tres (3) parámetros:

- *numeroMaximoEntidades*. Representa el tamaño del lote de entidades. Si el valor de este era muy grande, entonces se demoraba mucho en analizar y mostrar por pantalla el nivel de seguridad, dando la posibilidad de que el robo o crimen fuese realizado, ignorando así el fin último de la aplicación: el de ser una herramienta de seguridad ciudadana. Si el valor era muy bajo, entonces el sistema analizaba e informaba constantemente el nivel de seguridad, con resultados incoherentes, ya que eran muy pocas las entidades a analizar. Luego de varias pruebas, se optó por el valor de **cuatro (4)** para el tamaño del lote. El análisis efectuado para arribar a este valor fue el siguiente:
    - Para valores menores a 4. El problema principal con lotes de tamaño menor a 4 era que, a veces, algunas entidades no se identificaban correctamente. Luego, como el lote era muy pequeño, resultaba poco o nada representativo para una conversación real. Por ejemplo, sea el caso de una conversación que, en efecto, debe ser catalogada como insegura. Si el lote es muy chico, la conversación es dividida en varios lotes pequeños, donde cada uno de ellos puede ser identificado como "Seguro" y, por tanto, el hecho de inseguridad no es detectado.
    - Para valores mayores a 4. Si bien un lote mayor a 4 entidades implica una mejor representación de la realidad (ya que cada lote es más grande y puede conocerse con más precisión lo que realmente está ocurriendo), en la práctica se demoraba demasiado en analizar y mostrar un resultado en pantalla. Esto puede ocasionar que el hecho de inseguridad sea cometido y la herramienta informe tardíamente lo sucedido. El hecho de que la herramienta informe lo ocurrido con cierta tardanza puede ocurrir, principalmente, en conversaciones en las cuales la herramienta no puede identificar entidades, no por problemas de precisión sino porque la conversación propiamente dicha no tiene entidades que sean identificables de la lista que la herramienta reconoce. Una potencial solución a esto sería aumentar la lista de entidades.
- *tiempoEspera*. Representa el tiempo del temporizador, el cual indica cuando la conversación terminó y es necesario enviar el lote de entidades actual a analizar. Si el número es muy bajo, quizás la conversación aun no finaliza, pero se trata como si hubiera terminado ya que el temporizador se venció. Si el número es muy alto, entonces los resultados se muestran de forma tardía, ya que es necesario esperar más tiempo. Luego de varias pruebas, se optó por establecerlo en diez (10) segundos.

    - Para valores menores a 10 segundos. Si el valor es menor a 10 segundos, el temporizador se vence más seguido, y se asume que la conversación terminó, cuando lo más probable es que no sea así. Cuando se vence el temporizador, se asume que la conversación finalizó, y lo que viene no tiene relación con lo anterior. Luego, se pierde solapamiento entre lotes, generando problemas de precisión.
    - Para valores mayores a 10 segundos. Para este caso, ocurren dos problemas. Por un lado, si la conversación efectivamente terminó, se debe esperar hasta que el temporizador finalice y, si es mayor a 10 segundos, se debe esperar innecesariamente hasta conocer la respuesta de la herramienta. Por otro lado, si la conversación termina y luego comienza otra, y el temporizador aún no venció, tiene lugar el solapamiento de entidades. generando nuevamente errores de precisión en el nivel de inseguridad informado.
- *cantidadEntidadesASolapar*. Indica la cantidad de entidades que se van a solapar entre los distintos lotes de entidades. Naturalmente, el número de entidades a solapar tiene que ser menor al tamaño del lote. Entonces, luego de varias pruebas, los mejores resultados se obtuvieron con un valor de una (1) entidad a solapar.
    - Para valores mayores a 1. Si la cantidad de entidades a solapar es mayor a 1, cada uno de los lotes presenta demasiadas similaridades entre sí. Durante las pruebas realizadas con más de una entidad a solapar, los lotes son identificados con la misma categoría (debido a que comparten, en gran medida, el mismo tipo de entidades), lo cual en muchas ocasiones resulta erróneo. En pocas palabras, el problema que se presenta es que se está analizando constantemente "casi" el mismo lote varias veces, arrojando siempre la misma categoría, siendo que lo mejor es que la herramienta lo arroje una sola vez.

7.5. Código fuente

Todos los algoritmos mencionados en esta sección, así como también los módulos entrenados de NER y TEXTCAT se encuentran disponibles en el siguiente repositorio online:

https://github.com/lautinudel/PFCSeguridadCiudadana

Los pasos de instalación de la herramienta se encuentran en el archivo *README.md* ubicado en el repositorio. De todas formas, se facilita a continuación un hipervínculo para leer estos pasos desde la propia web de GitHub:

https://github.com/lautinudel/PFCSeguridadCiudadana#instalaci%C3%B3n

8. Casos de prueba

En esta sección se busca evaluar la capacidad de detección de hechos de inseguridad, en tiempo real, de la herramienta desarrollada, por medio de diversos escenarios previamente propuestos. En primer lugar, se presentan los casos de prueba que se utilizarán, seguido de la ejecución de cada uno de ellos, para finalmente arribar a un análisis sobre los resultados esperados contra los obtenidos.

8.1. Obtención de casos de prueba

Para poder evaluar la herramienta desarrollada, se propuso la utilización de subtítulos de películas y series argentinas. Más específicamente, se buscaron escenas de diversos estilos, procurando siempre que se asemejen lo más posible a situaciones de la vida real, y diferenciando escenas de peligro o que presentan cierto grado de inseguridad con aquellas que representan situaciones cotidianas sin ningún tipo de amenaza.

A continuación, se listan los diferentes casos de prueba realizados, junto con la situación concreta que acontece durante el intervalo de tiempo considerado en la escena (Tabla 8).

Prueba	Origen	Intervalo de tiempo	Contexto	Enlace público
1	Película argentina “Nueve Reinas”	2 minutos 19 segundos	Robo en la vía pública, con amenazas de muerte.	bit.ly/3dHMCN0
2	Película argentina “El Clan”	1 minuto 23 segundos	Secuestro a dos personas.	bit.ly/3uBDzUH
3	Serie argentina “Apache: la vida de Carlos Tevez”	1 minuto 49 segundos	Robo y asesinato en el domicilio de una pareja.	bit.ly/3mrEhRo
4	Película argentina “El Clan”	1 minuto 58 segundos	Reencuentro entre familiares luego de un largo viaje. Representa una situación de ausencia de peligro.	bit.ly/39QHGEr

Tabla 8. Descripción de casos de prueba

8.2. Ejecución de pruebas

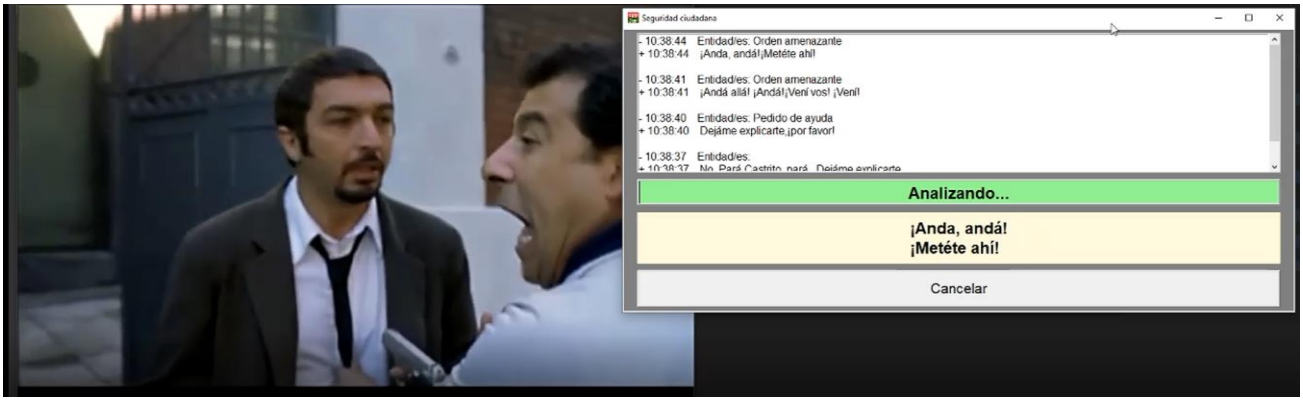
Esta sección está destinada a presentar las ejecuciones para cada caso de prueba. Para cada uno, se explican brevemente los resultados obtenidos junto con una descripción de la escena analizada.

Es necesario mencionar que, en cada ejecución, se muestra el video que representa la escena a analizar. Esto, por supuesto, es meramente ilustrativo pues la herramienta solo analiza los subtítulos de la misma.

En cada captura que se presente, los insultos están censurados para evitar que sean considerados ofensivos para el lector. Además, cada captura fue elegida al azar y se muestra en orden cronológico.

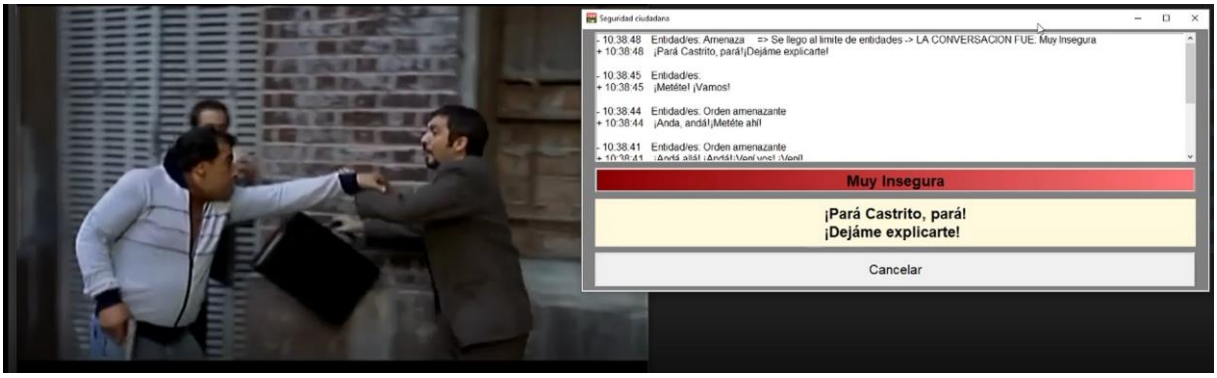
8.2.1. Caso de Prueba Nº1: Robo

En este caso de prueba, se presentan diferentes órdenes amenazantes y pedidos de ayuda, las cuales son efectivamente detectadas por la herramienta (Capturas 14 a 17).



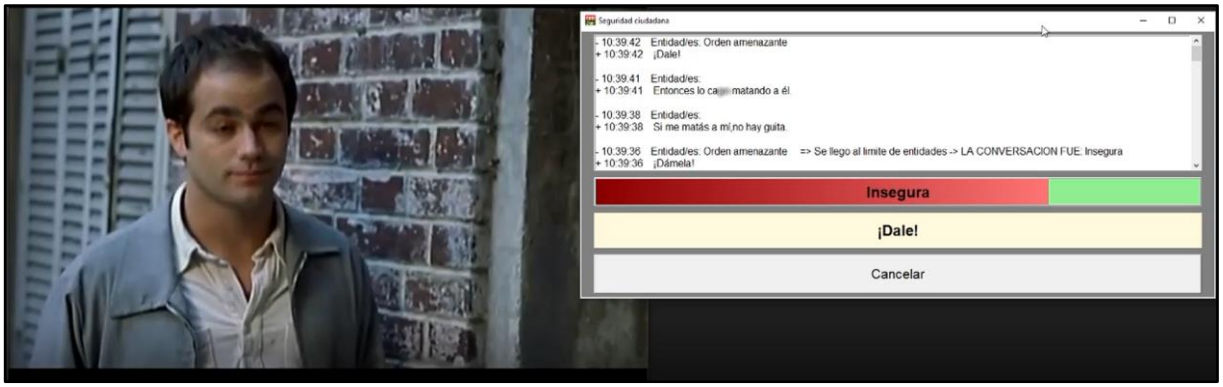
Captura 14. Ejecución de caso de prueba 1

Al determinarse el lote correspondiente, se obtiene el resultado esperado: “Muy Insegura”.

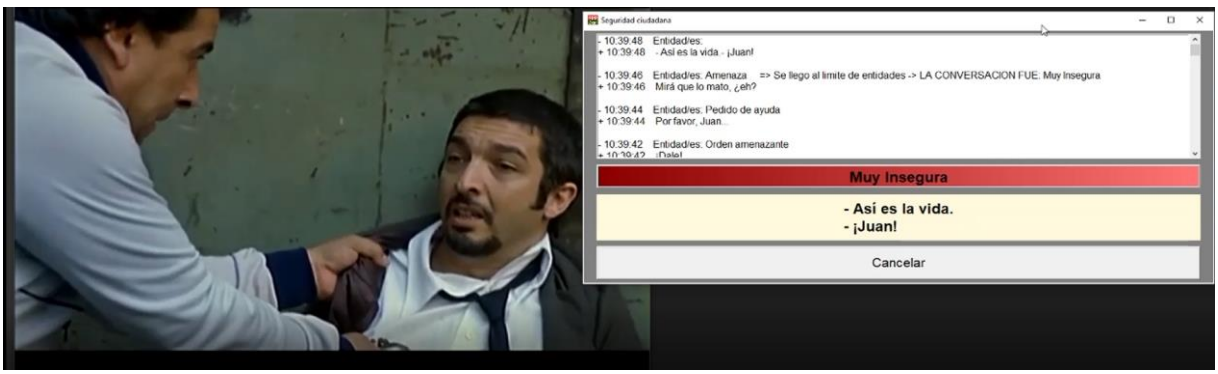


Captura 15. Ejecución de caso de prueba 1

Si se sigue analizando el resto de los lotes que se van generando conforme pasa el tiempo, se puede observar que el patrón de inseguridad se repite en varios casos, de manera más grave o más leve.



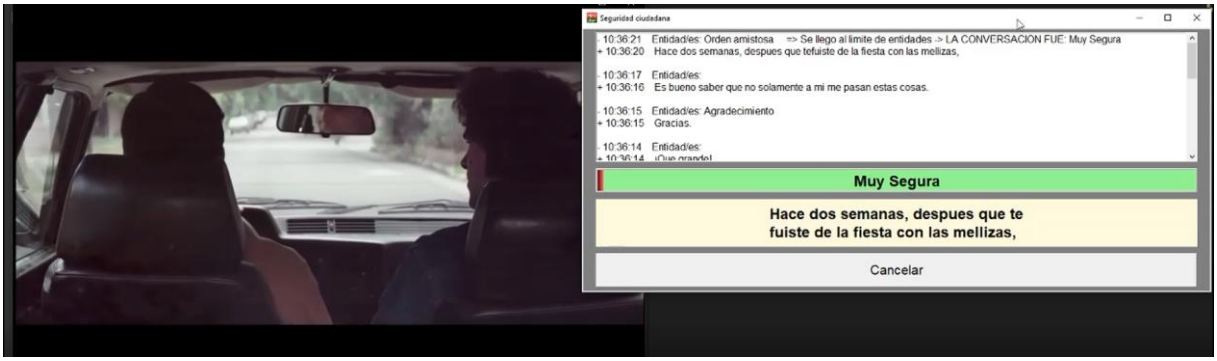
Captura 16. Ejecución de caso de prueba 1



Captura 17. Ejecución de caso de prueba 1

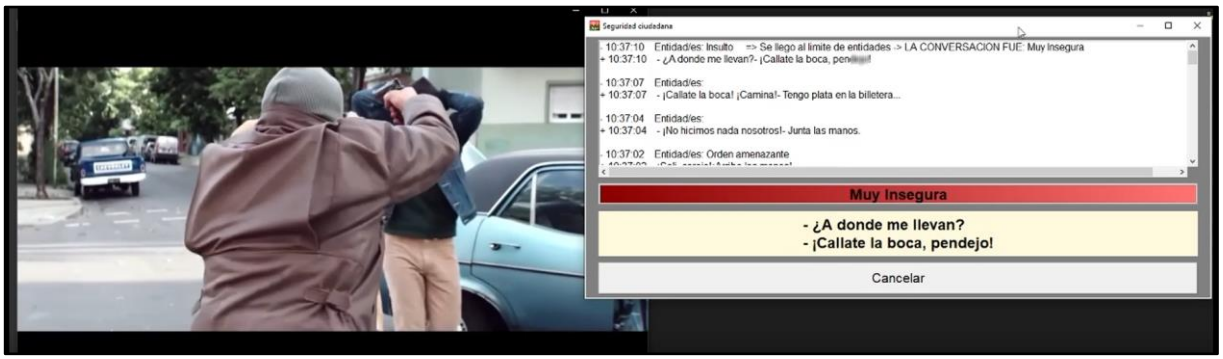
8.2.2. Caso de Prueba Nº2: Secuestro

En este caso, inicialmente, se presenta una conversación entre dos personas. En ella, no existe ningún tipo de peligro, pues simplemente se trata de dos amigos hablando entre sí. A medida que la misma va transcurriendo, el sistema detecta que es una situación “Muy segura”, lo cual es correcto (Captura 18).



Captura 18. Ejecución de caso de prueba 2

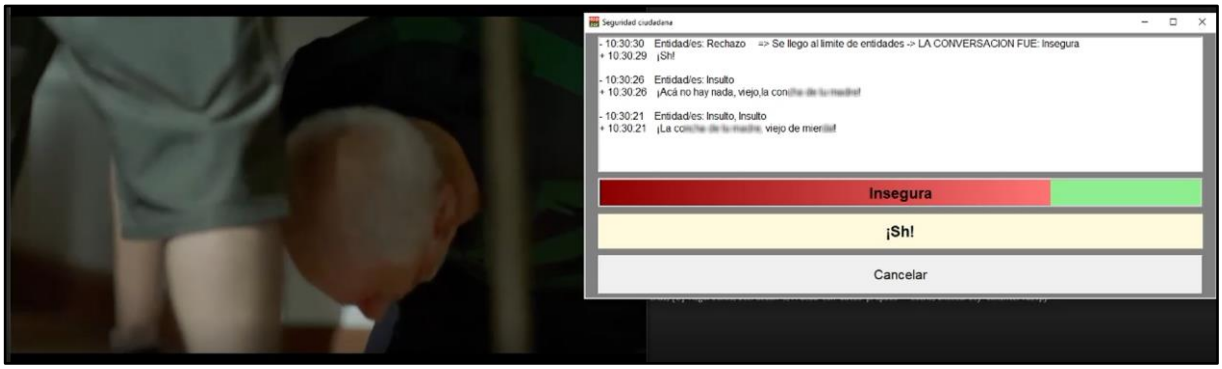
En un determinado momento, se produce un secuestro, momento en que la herramienta revela inseguridad al analizar el lote, clasificando la situación como “Muy Insegura” (Captura 19).



Captura 19. Ejecución de caso de prueba 2

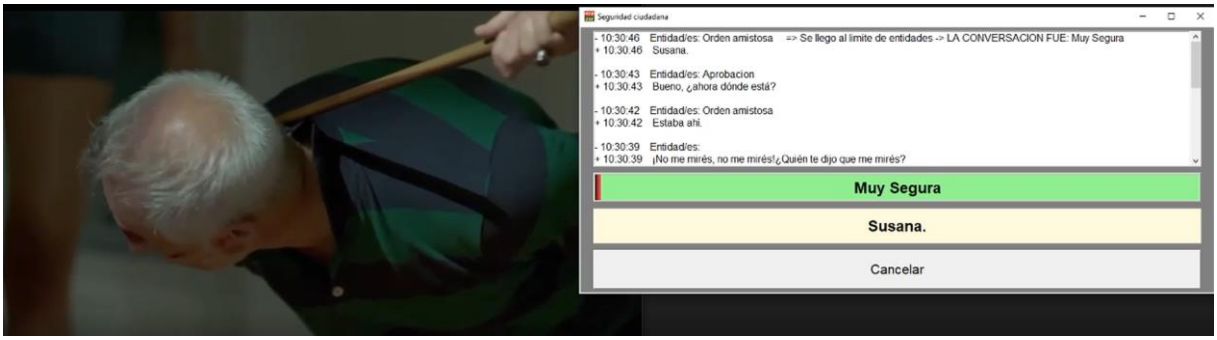
8.2.3. Caso de Prueba Nº3: Robo y asesinato

Inicialmente, se presenta un contexto de robo, en el cual el asaltante insulta y golpea a la víctima. Al solo detectar insultos y ningún tipo de amenaza verbal, el sistema clasifica la situación como “Insegura” (Captura 20).



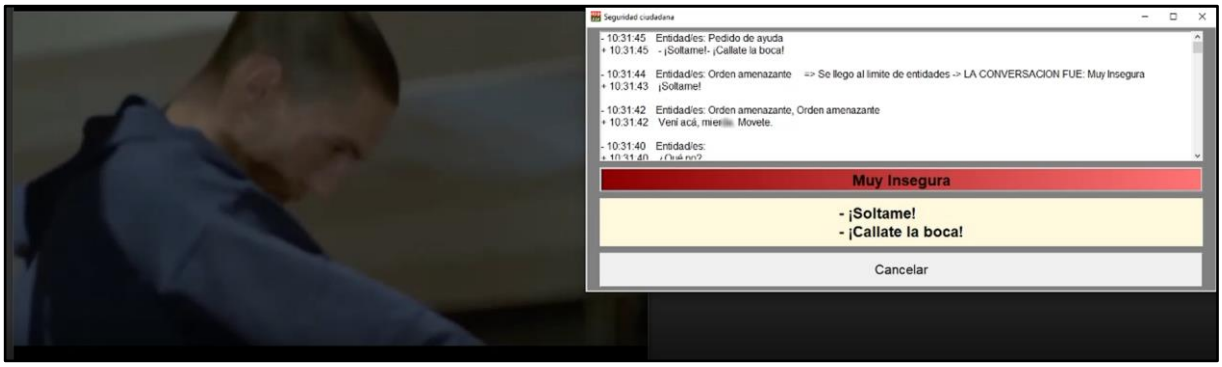
Captura 20. Ejecución de caso de prueba 3

Luego, el sistema presenta un error: detecta el siguiente conjunto de frases como muy segura (Captura 21). Esto se debe a que las entidades detectadas para esta parte no siempre son correctas, debido que, tal como se mencionó en la sección “7.2. Arquitectura”, la herramienta no es perfecta y puede introducir errores.



Captura 21. Ejecución de caso de prueba 3

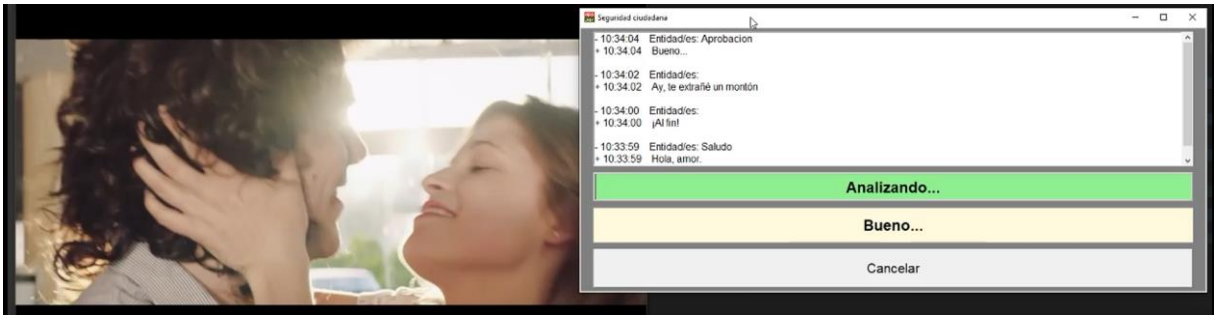
A continuación, al presentarse una secuencia de órdenes amenazantes, insultos y pedidos de ayuda, la herramienta arroja como resultado la presencia de una situación “Muy Insegura” (Captura 22).



Captura 22. Ejecución de caso de prueba 3

8.2.4. Caso de Prueba Nº4: Situación normal

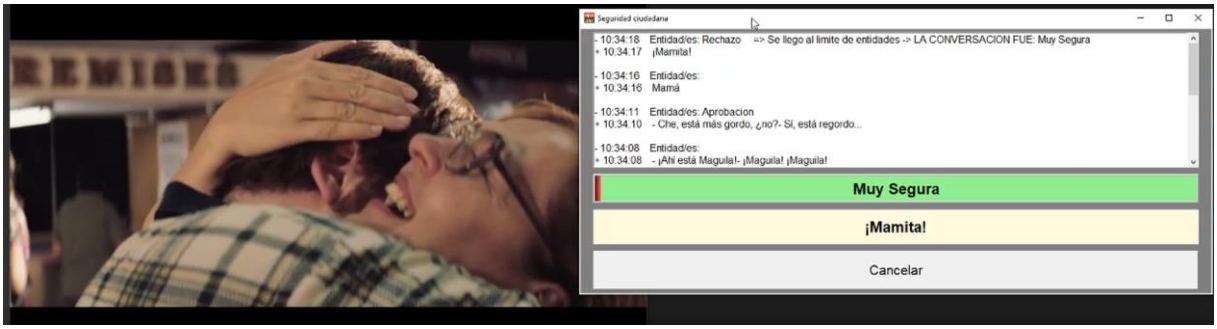
Inicialmente, se presenta una escena en la que acontece un reencuentro entre dos personas (Captura 23). En ella, la aplicación detecta efectivamente que está ocurriendo un saludo entre los participantes de la escena.



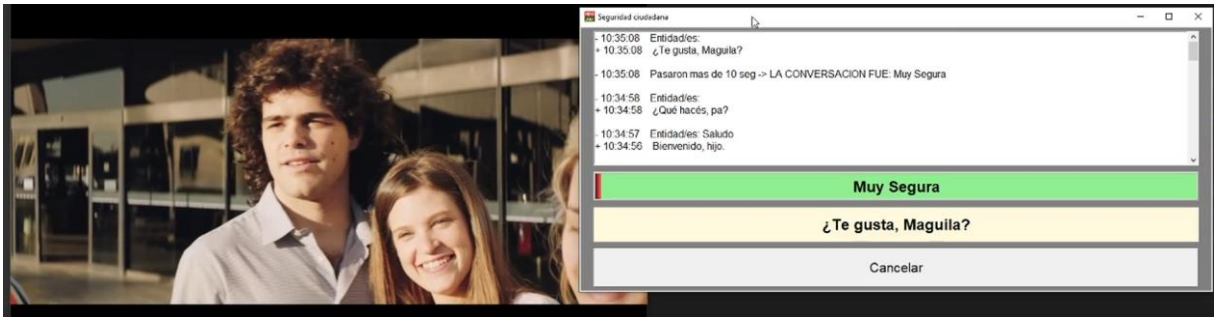
Captura 23. Ejecución de caso de prueba 4

La escena continúa dentro del contexto de una situación normal de reencuentro: se hacen diferentes preguntas, afirmaciones y aprobaciones, sin ningún tipo de amenaza, pedido de ayuda e insulto. Estas afirmaciones y aprobaciones fueron oportunamente detectadas por el sistema (Captura 24).

Al llenarse el lote correspondiente, el sistema responde con una clasificación de la situación para el lote analizado, que en este caso es “Muy Segura” (Captura 25).



Captura 24. Ejecución de caso de prueba 4



Captura 25. Ejecución de caso de prueba 4

8.3. Resultados de pruebas

A continuación, se muestra una tabla comparativa entre los resultados esperados y los resultados obtenidos, para cada una de las pruebas (Tabla 9).

Prueba	Input **(Anexo B)**	Resultados esperados		Resultados obtenidos	
		Tipos de entidades esperadas	Tipos de niveles de seguridad esperados	Tipos de entidades obtenidas	Tipos de niveles de seguridad obtenidos
1	Subtítulos de escena de Nueve Reinas	Pedido de ayuda Amenaza Orden amenazante Rechazo Insulto Aprobación	Muy Insegura Insegura	Pedido de ayuda Orden amenazante Amenaza Pedir disculpas Insulto Rechazo Saludo Aprobación	Muy Insegura Insegura
2	Subtítulos de escena de El Clan	Saludo Agradecimiento Orden amistosa Orden amenazante Pedido de ayuda Insulto	Muy Segura Insegura Muy Insegura	Saludo Rechazo Agradecimiento Orden amistosa Pedido de ayuda Orden amenazante Insulto Aprobación	Muy Segura Insegura Muy Insegura

3	Subtítulos de escena de Apache la vida de Carlos Tevez	Insulto Orden amenazante Rechazo Amenaza Pedido de ayuda	Muy Insegura Insegura	Insulto Rechazo Orden amistosa Aprobación Pedir disculpas Orden amenazante Pedido de ayuda	Insegura Muy Segura Muy Insegura
4	Subtítulos de escena de El Clan	Saludo Aprobación Orden amistosa Rechazo	Muy Segura	Saludo Aprobación Orden Amistosa Rechazo	Muy Segura

Tabla 9. Resultados de las pruebas realizadas

8.3.1. Resultados de CP Nº1: Robo

En términos generales, la prueba marchó correctamente. ya que los niveles de seguridad resultantes fueron los esperados. No se detectaron los niveles “Seguro” y “Muy seguro”, lo cual es correcto pues se trata de una escena con considerable nivel de violencia. El problema principal estuvo en la detección de las entidades ya que, si bien la mayoría fueron correctamente identificadas, también se detectaron las entidades “Saludo” y “Pedir disculpas”, que no se encontraban dentro del set de entidades esperadas para este caso de prueba. Esto ocurrió en las siguientes dos frases de la escena:

- “¡Tu vieja está muerta!” -> Pedir disculpas
- “¿Qué? -> Saludo

Otro punto importante a mencionar para este caso de prueba, es el hecho de que algunas frases que evidentemente contenían entidades, no fueron detectadas, tal como el siguiente ejemplo, que podría catalogarse como Amenaza :“Dame la guita ya, sino te hago mie***”.

8.3.2. Resultados de CP Nº2: Secuestro

Este caso resultó muy similar a la prueba anterior, ya que los niveles de seguridad detectados fueron correctos. La diferencia principal reside en el hecho de que, en este caso, se puede observar un contraste de situaciones: la escena comienza con una situación normal, sin peligro (que la herramienta detecta sin problemas), para luego detectar también la situación de peligro.

Si bien, en general, fueron detectadas las entidades esperadas, también se introdujeron ciertos errores, a saber:

- “¡No puedo creer encontrarte acá!” -> Rechazo
- “Hace dos semanas, después que te fuiste de la fiesta con las mellizas.” -> Orden amistosa.

8.3.3. Resultados de CP Nº3: Robo y asesinato

En este caso de prueba no se arribó a los resultados esperados. Se presentaron problemas tanto en las entidades detectadas como en los niveles de seguridad calculados. Algunas entidades mal identificadas fueron las siguientes:

- “¡Sh!” -> Rechazo
- “Susana” -> Orden amistosa
- “P******, ¿por qué la cambiaste?” -> Pedir disculpas

Además, en un lote del caso de prueba se detecta el nivel de seguridad “Muy Seguro”, el cual es incorrecto. Dicho lote conflictivo fue el siguiente:

- Lote: Orden amistosa, Pedir disculpas -> Muy Segura

El error en este caso no corresponde a un problema en el módulo TEXTCAT (ya que el lote si corresponde a un nivel Muy Seguro), sino al módulo NER, como consecuencia de que las entidades fueron mal identificadas.

8.3.4. Resultados de CP Nº4: Situación normal

Durante la ejecución de esta prueba, tanto los tipos de entidades como los niveles de seguridad coincidieron con los esperados. Al tratarse de una situación sin mayores peligros, las entidades esperadas son aquellas referidas a situaciones comunes y no de riesgo. Por su parte, el nivel de seguridad esperado es el de “Muy seguro”, el cual es, en efecto, el único resultado obtenido.

Esta prueba resultó importante para verificar que la herramienta funciona de la forma esperada en situaciones cotidianas, sin peligro, demostrando así que se evitan los falsos positivos.

8.3.5. Conclusiones finales

Los resultados obtenidos en el proceso de evaluación de las pruebas planteadas fueron exitosos, detectando situaciones de inseguridad ciudadana donde las hubiera y donde no.

Sin embargo, entrando en detalle, se puede observar que el principal problema que presenta la herramienta se encuentra en la detección de entidades, ya que a veces esta es errónea. Esto se ve reflejado en la precisión obtenida para NER, la cual no es del todo elevada.

Esta problemática podría afrontarse empleando un dataset mucho más grande, exhaustivo y consistente. Para lograrlo, una posibilidad sería la de etiquetar una mayor cantidad de subtítulos de películas o series, o bien aplicar técnicas de tratamiento de datasets para aumentar el tamaño del mismo sin la necesidad de etiquetar nuevamente. Una de estas técnicas es la denominada *data augmentation*, en la que se busca aumentar el tamaño del dataset en base a los datos originales ya etiquetados. Por ejemplo, se toma una frase, se la traduce a algún idioma y luego se vuelve a traducirla a su idioma original, resultando en una frase similar a la original, pero aportando mayor nivel de conocimiento al dataset. Sin embargo, ninguna de estas estrategias fue implementada en el proyecto, ya que quedaron fuera del alcance del mismo, debido al tiempo que demandaba aplicarlas.

9. Conclusiones y trabajos futuros

La principal conclusión que podemos afirmar es que se pudieron realizar satisfactoriamente los objetivos planteados en la etapa de planificación del proyecto: La investigación de tecnologías de información asociadas a Machine Learning y al Procesamiento del Lenguaje Natural, la definición de un dataset de situaciones de inseguridad ciudadana, el diseño de un algoritmo basado en Machine Learning, el entrenamiento de los modelos, la red neuronal con el dataset definido y selección de modelo más adecuado, la creación de un prototipo ejecutable y su correspondiente validación.

Considerando el alcance establecido, junto con los tiempos acordados en el plan asociado, podemos decir que el resultado obtenido no contempla todos los posibles casos. No se comporta de la misma forma con todos los posibles escenarios. Como desarrollo futuro, se plantea construir un dataset más exhaustivo para lograr un mejor resultado.

Respecto a los plazos establecidos, no se contempló la posibilidad de una pandemia mundial, por lo cual los plazos de entrega se vieron afectados ante la incertidumbre en este contexto.

Con la realización del presente PFC, se puede desdoblar el alcance considerado, junto con toda su correspondiente investigación y funcionalidad.

El objetivo propuesto fue crear una herramienta capaz de detectar situaciones de inseguridad en una conversación textual, donde la misma pueda aprender en base a un entrenamiento previo y sin comprometer la privacidad de las personas. Considerando esto, podríamos extender esta meta abarcando aspectos tales como el *grooming*, principal delito online que implica el contacto de un adulto a un menor, generando confianza en este último y concluyendo en un abuso. Para atacar este problema, se propone a futuro cambiar nuestro enfoque y nuestras variables, de tal forma que en una determinada conversación textual se puedan detectar material de carácter sexual y poder realizar acciones frente a ello para prevenir un posible abuso.

Siguiendo el eje establecido, donde se abarcó la detección vía texto de situaciones de inseguridad, podríamos abordar situaciones en la que un determinado vendedor contacta a sus clientes de manera indebida, utilizando insultos de por medio, hasta incluso amenazas. Ante

esto, se podría no solo detectar dichas situaciones, sino relevar estadísticas sobre el uso de determinadas palabras y trato en concreto, permitiendo al empleador realizar ciertas métricas sin conocer exactamente cómo se llevó a cabo toda la conversación.

Fuera de estas situaciones planteadas, en un futuro se podría desarrollar un módulo de detección de sonido capaz de convertir voz a texto, de tal forma de poder anexar dicho componente a nuestro proyecto y poder implementar el mismo en conversaciones telefónicas, videollamadas, entre otras.

En el comienzo del proyecto, el entorno del mismo resultó totalmente desconocido. Ninguno de los integrantes conocía nada sobre las ramas de ML y PLN, lo cual significaba un gran desafío para todo el equipo. Gracias al desarrollo incremental del proyecto, en conjunto con la debida paciencia y dedicación, se logró sobrellevar exitosamente el curso del proyecto. Como equipo, se logró coordinar las actividades de buena manera, junto con un correcto diálogo y disposición de cada uno de los integrantes. La consecución de este PFC denota una gran experiencia para todo el equipo, que conlleva a un crecimiento tanto personal como profesional en extremo importante.

10. Referencias

Introducción.

[1] Gobierno de la República Argentina. *Estadísticas criminales de la República Argentina. Recuperado el 27 de noviembre de 2019,* del website oficial del Gobierno de la República Argentina*:* https://www.argentina.gob.ar/seguridad/estadisticascriminales

Marco Teórico.

[2] Russell, S. J.; Norvig, P., (2004). *Inteligencia Artificial: Un enfoque moderno,* Madrid, España: Pearson Educación

[3] Artificial Intelligence and Machine Learning: Policy Paper (2017). Recuperado el 7 de agosto de 2020, de Internet Society website: https://www.internetsociety.org/resources/doc/2017/artificial-intelligence-and-machine-learning-policy-paper/?gclid=CjwKCAjw97P5BRBQEiwAGflV6XsUJvtYekeiQKHn8PlpbygsyOWNVcKIZJNO3vxs2yv_smjptJvEHRoCyh8QAvD_BwE

[4] Ajanki, A. (2018). *Differences between machine learning and software engineering*. Recuperado el 19 de agosto de 2020, de Futurice website: https://futurice.com/blog/differences-between-machine-learning-and-software-engineering

[5] Wilson, A. (2019). *A Brief Introduction to Supervised Learning.* Recuperado el 19 de agosto de 2020, de Futurice website: https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590

[6] IBM (2020). *Unsupervised Learning.* Recuperado el 09 de noviembre de 2020, de IBM website: https://www.ibm.com/cloud/learn/unsupervised-learning

[7] Ye, A. (2020). *Supervised Learning, But A Lot Better: Semi-Supervised Learning*. Recuperado el 10 de noviembre de 2020, de Towards Data Science website: https://towardsdatascience.com/supervised-learning-but-a-lot-better-semi-supervised-learning-a42dff534781

[8] Lopez Yse, D. (2019). *Your Guide to Natural Language Processing (NLP).* Recuperado el 23 de noviembre de 2020, de Towards Data Science website: https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1

[9] Marshall, C. (2019). *What is named entity recognition (NER) and how can I use it?* Recuperado el 23 de noviembre de 2020, de Medium website: https://medium.com/mysuperai/what-is-named-entity-recognition-ner-and-how-can-i-use-it-2b68cf6f545d

[10] MonkeyLearn. *Text Classification*. Recuperado el 23 de noviembre de 2020, de MonkeyLearn website: https://monkeylearn.com/text-classification/#:~:text=Text%20classification%20is%20the%20process,spam%20detection%2C%20and%20intent%20detection

[11] Amsantac. *Por qué es importante trabajar con datos balanceados para clasificación.* Recuperado el 14 de diciembre de 2020, de Amsantac website: http://amsantac.co/blog/es/2016/09/20/balanced-image-classification-r-es.html

[12] Teja, Sai (2020). *Stop Word in NLP*. Recuperado el 7 de junio de 2021, de Medium website: https://medium.com/@saitejaponugoti/stop-words-in-nlp-5b248dadad47

Metodología Utilizada.

[13] Course | ML0101SP | edX. (n.d.). *Machine-Learning Con Python: Una Introducción Práctica*. https://courses.edx.org/courses/course-v1:IBM+ML0101SP+3T2019/course/

[14] Deep Learning. (n.d.). Coursera. https://www.coursera.org/specializations/deep-learning

[15] *Python Programming for Beginners in Data Science*. (n.d.). Udemy. https://www.udemy.com/course/just-enough-python/

[16] Mete, E. (n.d.). *Simple Sentiment Analysis With NLP*. Dzone.Com. https://dzone.com/articles/simple-sentiment-analysis-with-nlp

[17] Li, S. (n.d.). *A Beginner's Guide on Sentiment Analysis with RNN* - Towards Data Science. Medium. https://towardsdatascience.com/a-beginners-guide-on-sentiment-analysis-with-rnn-9e100627c02e

[18] Haitz, D. (2019, April 8). *Creating and Deploying a Python Machine Learning Service*. Medium. https://towardsdatascience.com/creating-and-deploying-a-python-machine-learning-service-a06c341f020f

[19] *Is there any way to define custom entities in spaCy*. (2019, August 16). Data Science Stack Exchange. https://datascience.stackexchange.com/questions/57650/is-there-any-way-to-define-custom-entities-in-spacy

[20] Murugavel, M. (2020, July 20). *How to Train NER with Custom training data using spaCy*. Medium. https://manivannan-ai.medium.com/how-to-train-ner-with-custom-training-data-using-spacy-188e0e508c6

[21] *How to train a spacy model for text classification?* (2019, July 18). Data Science Stack Exchange. https://datascience.stackexchange.com/questions/55896/how-to-train-a-spacy-model-for-text-classification

[22] *Training spaCy's Statistical Models* · spaCy Usage Documentation. (n.d.). Training SpaCy's Statistical Models. https://spacy.io/usage/training/

[23] *Training spaCy's Statistical Models · spaCy Usage Documentation*. (n.d.-b). Training SpaCy's Statistical Models. https://spacy.io/usage/training/#ner

[24] *Scorer* · spaCy API Documentation. (n.d.). Scorer. https://spacy.io/api/scorer/

[25] *Evaluation in a Spacy NER model*. (2017, June 29). Stack Overflow. https://stackoverflow.com/questions/44827930/evaluation-in-a-spacy-ner-model

[26] E. (n.d.). *Spacy NER Evaluation is being incorrect · Issue #4181* · explosion/spaCy. GitHub. https://github.com/explosion/spaCy/issues/4181

Tecnologías utilizadas.

[27] SpaCy · Industrial-strength Natural Language Processing in Python. (n.d.). SpaCy. https://spacy.io/

[28] Wikipedia. (2020, October 26). Procesamiento de lenguajes naturales. Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/Procesamiento_de_lenguajes_naturales

[29] Moreno, A. (2021, January 14). Procesamiento del lenguaje natural ¿qué es? Instituto de Ingeniería Del Conocimiento. https://www.iic.uam.es/inteligencia/que-es-procesamiento-del-lenguaje-natural/

[30] Natural Language Toolkit — NLTK 3.5 documentation. (n.d.). NLTK. https://www.nltk.org/

[31] *Standford Core NLP*. (n.d.). Core NLP. https://stanfordnlp.github.io/CoreNLP/

[32] Team, T. A. O. (n.d.). Apache OpenNLP. Apache OpenNLP. https://opennlp.apache.org/

[33] Apache UIMA - Apache UIMA. (n.d.). Apache UIMA. https://uima.apache.org/

[34] Text Analysis. (n.d.). MonkeyLearn. https://monkeylearn.com/

[35] Facts & Figures · spaCy Usage Documentation. (n.d.). Facts & Figures. https://spacy.io/usage/facts-figures/

[36] Linguistic Features · spaCy Usage Documentation. (n.d.). Linguistic Features. https://spacy.io/usage/linguistic-features/#named-entities

[37] *Embed, encode, attend, predict: The new deep learning formula for state-of-the-art NLP models* ·. (2016, 10 noviembre). Explosion. https://explosion.ai/blog/deep-learning-formula-nlp/

[38] Guide to Text Classification with Machine Learning. (n.d.). MonkeyLearn. https://monkeylearn.com/text-classification/

[39] Appjar. (n.d.). Appjar. http://appjar.info/

[40] doccano - Document Annotation Tool. (n.d.). Doccano. https://doccano.herokuapp.com/

[41] Releases · SubtitleEdit/subtitleedit. (n.d.). GitHub. https://github.com/SubtitleEdit/subtitleedit/releases

[42] Google Drive. (n.d.). Google Drive. https://drive.google.com/

[43] Remover líneas duplicadas online. (n.d.). PineTools. https://pinetools.com/es/remover-lineas-duplicadas

[44] Project Jupyter. (n.d.). Home. https://jupyter.org/

[45] Microsoft. (2016, April 14). Visual Studio Code - Code Editing. Redefined. https://code.visualstudio.com/

[46] Google Colaboratory. (n.d.). Google Colab. https://colab.research.google.com/

Entrenamiento de módulos.

[47] Training spaCy's Statistical Models · spaCy Usage Documentation. (n.d.-c). Training SpaCy's Statistical Models. https://spacy.io/usage/training/

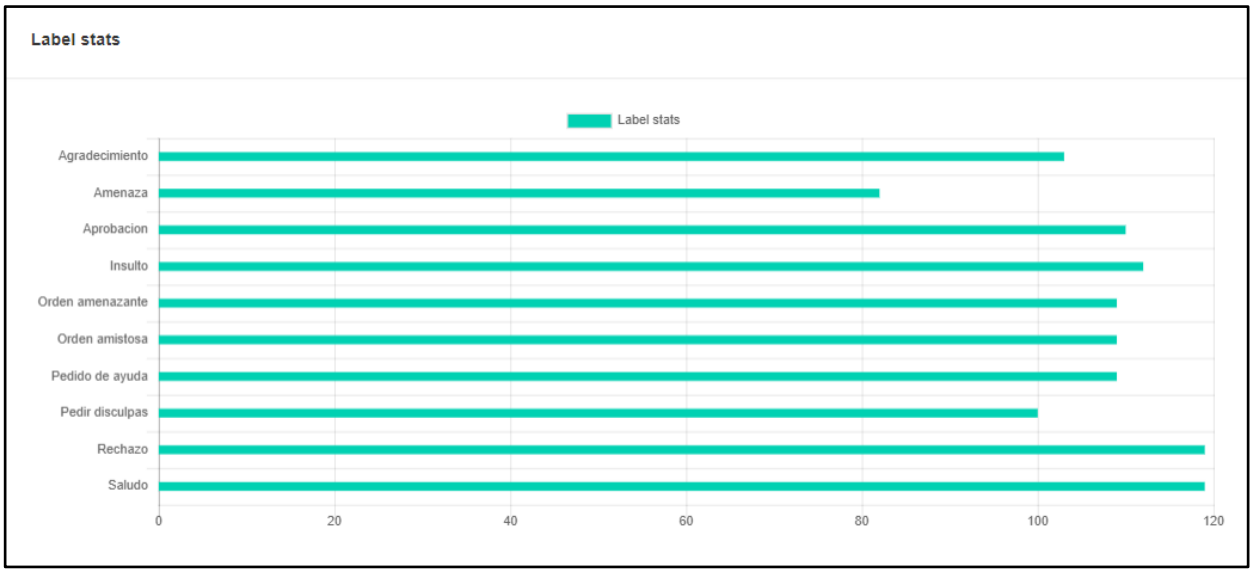
[48] Heras, J. M. (2020, October 9). Precision, Recall, F1, Accuracy en clasificación. IArtificial.Net. https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/

[49] What is batch size in neural network? (2015, May 22). Cross Validated. https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network

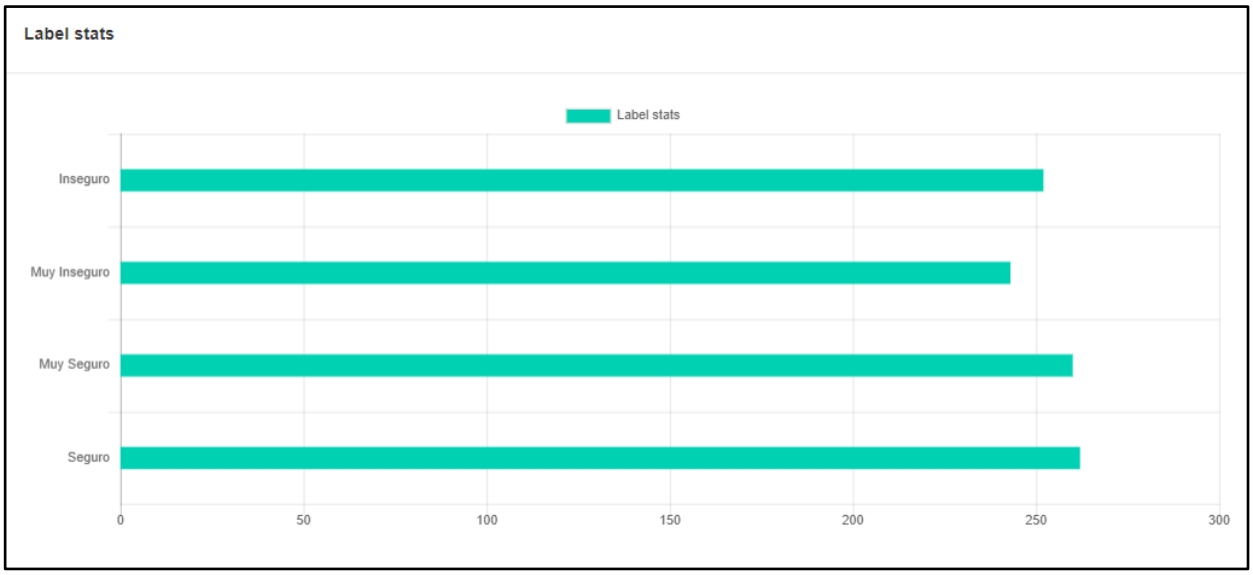
11. Anexos

11.1. Anexo A

Estadísticas del Dataset correspondiente a NER.



Estadísticas del Dataset correspondiente a TEXTCAT.



11.2. Anexo B

Caso de prueba N°1: Robo.

00:00:00,200 --> 00:00:02,800
No. Pará Castrito, pará...
Dejáme explicarte...

00:00:02,801 --> 00:00:04,200
Dejáme explicarte,
¡por favor!

00:00:04,201 --> 00:00:06,720
¡Andá allá! ¡Andá!
¡Vení vos! ¡Vení!

00:00:06,721 --> 00:00:07,721
¡Anda, andá!
¡Metéte ahí!

00:00:07,722 --> 00:00:09,600
¡Metéte! ¡Vamos!

00:00:10,520 --> 00:00:12,600
¡Pará Castrito, pará!
¡Dejáme explicarte!

00:00:12,601 --> 00:00:14,200
¡Te juro que te iba a pagar!
¡Se enfermó mi vieja!

00:00:14,201 --> 00:00:15,201
¡Tu vieja está muerta!

00:00:15,202 --> 00:00:16,600
¡Por eso! ¡Pobre, se enfermó
y se murió!

00:00:16,601 --> 00:00:18,280
- Porque...
- ¡Calláte, calláte!

00:00:18,281 --> 00:00:20,400
¡Hijo de p***! ¡Hace un año
que me venís c******...

00:00:20,401 --> 00:00:21,401
...con la misma historia!

00:00:21,402 --> 00:00:23,400
¡Dame la guita ya, sino
te hago m****!

00:00:23,401 --> 00:00:24,600
¡Por favor!
¡Te lo suplico, por favor!

00:00:24,601 --> 00:00:26,240
¡No supliques más!
¡Quiero mi guita ya!

00:00:26,241 --> 00:00:28,800
¡Ahora! ¡Si no te mato
como a un perro, hijo de p***!

00:00:31,000 --> 00:00:32,080
Pibe...

00:00:45,240 --> 00:00:46,560
¿Vos me estás jodiendo a mí?

00:00:46,561 --> 00:00:50,200
¿Vos querés hacerte matar
por un cheque de m****, p****?

00:00:50,201 --> 00:00:51,400
¡No! ¿por qué? si yo...

00:00:51,401 --> 00:00:53,120
¿Qué hicíste con la guita?

00:00:54,600 --> 00:00:55,800
La escondí...

00:00:56,200 --> 00:00:58,000
A ver si nos afanaban
de nuevo todavía...

00:00:59,400 --> 00:01:00,360
¡Dámela!

00:01:01,080 --> 00:01:03,080
Si me matás a mí,
no hay guita.

00:01:04,200 --> 00:01:05,440
Entonces lo c*** matando a él.

00:01:05,441 --> 00:01:06,880
¡Dale!

00:01:06,881 --> 00:01:08,040
Por favor, Juan...

00:01:08,800 --> 00:01:09,920
Mirá que lo mato, ¿eh?

00:01:11,000 --> 00:01:12,520
- Así es la vida.
- ¡Juan!

00:01:13,080 --> 00:01:15,080
- No.
- ¿No?

00:01:15,081 --> 00:01:16,960
- No.
- ¿Por qué?

00:01:16,961 --> 00:01:19,120
Porque dijo: "cheque de mierda".

00:01:19,121 --> 00:01:20,440
¿Qué?

00:01:20,441 --> 00:01:22,200
Que dijíste "cheque",
en lugar de "guita".

00:01:22,201 --> 00:01:23,600
¡Yo no dije "cheque"!

00:01:23,601 --> 00:01:24,800
Sí, dijíste "cheque".

00:01:24,801 --> 00:01:26,000
¡Yo no dije "cheque"!

00:01:26,001 --> 00:01:27,960
¡Sí, dijíste "cheque",
la re p*** que te parió!

00:01:27,961 --> 00:01:29,840
¡Dijíste "cheque"!
¡Dijíste "cheque"!

00:01:30,880 --> 00:01:32,040
Y bueno, me llamáste
a última hora...

00:01:32,041 --> 00:01:33,800
¿Qué querés?
No tuve tiempo de preparar nada.

00:01:33,801 --> 00:01:34,801
¡Qué pedazo de p*****!

00:01:34,802 --> 00:01:37,120
¡Tomatelas! ¿No ves que sos
un inútil de m****? ¡Andáte!

00:01:37,121 --> 00:01:40,200
¿Y los 50?
Los 50 para pagar el...

00:01:40,201 --> 00:01:41,000
¿Para pagar el qué?

00:01:41,001 --> 00:01:42,600
- El chumbo.
- ¡Metételo en el o*** el chumbo!

00:01:42,601 --> 00:01:44,600
¡Andá! ¡P**** de mierda!

00:01:45,800 --> 00:01:47,600
¿Te das cuenta por qué
te necesito yo a vos?

00:02:01,000 --> 00:02:02,400
¿Que me mirás con esa cara
de virgen violada?

00:02:02,401 --> 00:02:03,401
¿Qué te pasa?

00:02:03,402 --> 00:02:05,400
¿No sabés a qué me dedico?
¿Qué esperabas?

00:02:06,600 --> 00:02:08,000
Un milagro

Caso de prueba N°2: Secuestro.

00:00:00,000 --> 00:00:01,001

¿Que hacés, loco?

00:00:01,253 --> 00:00:02,169

¿Cómo estás?

00:00:02,254 --> 00:00:03,754

No puedo creer encontrarte acá.

00:00:04,005 --> 00:00:05,214

¿Que hacés vos acá?

00:00:05,423 --> 00:00:08,592

Me quede sin nafta. Tengo el auto

aca a 10 cuadras. ¿No me tiras?

00:00:08,843 --> 00:00:10,719

Por supuesto. Arriba, vamos, dale.

00:00:12,180 --> 00:00:13,389

¡Que grande!

00:00:13,640 --> 00:00:14,640

Gracias.

00:00:14,933 --> 00:00:17,810

Es bueno saber que no solamente a mi
me pasan estas cosas.

00:00:18,979 --> 00:00:23,274

Hace dos semanas, despues que te

fuiste de la fiesta con las mellizas,

00:00:23,525 --> 00:00:25,776

llegaron a casa unas escandinavas...

00:00:26,027 --> 00:00:27,528

Por acá a la izquierda.

00:00:27,779 --> 00:00:30,823

Eran danesas, suecas...

Me llevaban una cabeza.

00:00:31,157 --> 00:00:34,159

Y unos días después me las estoy
llevando al barco de papá,

00:00:34,369 --> 00:00:37,121

que ahora me lo presta porque estoy

trabajando con el...

00:00:37,330 --> 00:00:39,290

Y hasta ahi- todo bien,

todo fenomeno...

00:00:39,541 --> 00:00:43,127

Estoy llegando, las minas entregadas,

locas por navegar...

00:00:43,461 --> 00:00:44,837

¡Y me quede sin nafta!

00:00:48,842 --> 00:00:51,010

- ¿Que tiene, un arma?

- ¡Quietos ahi, c*****!

00:00:51,261 --> 00:00:53,637

- ¡Levanta las manos!

- ¡Ponete la capucha!

00:00:53,888 --> 00:00:56,807

- ¡Pará!, pará, baja el arma!

- ¡Ponete la capucha!

00:00:57,142 --> 00:01:00,019

- ¡Baja! ¡Tenelas bien arriba!

- Si, viejo, si...

00:01:00,312 --> 00:01:02,146

¡Sali, carajo!

¡Arriba las manos!

00:01:02,355 --> 00:01:04,857

- ¡No hicimos nada nosotros!

- Junta las manos.

00:01:05,108 --> 00:01:07,985

- ¡Callate la boca! ¡Camina!

- Tengo plata en la billetera...

00:01:08,236 --> 00:01:11,030

- ¿A donde me llevan?

- ¡Callate la boca, p****!

00:01:11,281 --> 00:01:13,490

- ¡Baja la cabeza!

- ¿Adonde me llevan?

00:01:13,783 --> 00:01:15,701

¡Baja la cabeza y entra, carajo!

00:01:15,952 --> 00:01:17,578

¡Nosotros no hicimos nada!

00:01:19,998 --> 00:01:21,665

- ¡Baja la cabeza!

- ¡Soltame!

00:01:22,334 --> 00:01:23,876

¡Baja la cabeza! ¡Metete!

00:01:27,922 --> 00:01:30,215

[Respira agitadamente]

00:01:45,732 --> 00:01:47,524

¿Estas bien?

00:01:49,277 --> 00:01:50,819

¿Alex?

00:01:51,363 --> 00:01:52,696

Si, estoy bien, papa.

Caso de prueba N°3: Robo y asesinato.

00:00:00,000 --> 00:00:02,100

¡La c**** de tu madre, viejo de m****!

00:00:04,780 --> 00:00:06,570

¡Acá no hay nada, viejo,

la c**** de tu madre!

00:00:07,980 --> 00:00:10,290

¡Sh!

00:00:14,100 --> 00:00:15,460

¿Por qué mentís, viejo?

00:00:16,420 --> 00:00:17,810

Te juro que estaba ahí.

00:00:17,900 --> 00:00:20,250

¡No me mirés, no me mirés!

¿Quién te dijo que me mirés?

00:00:20,340 --> 00:00:21,480

Estaba ahí.

00:00:21,900 --> 00:00:23,380

Bueno, ¿ahora dónde está?

00:00:24,420 --> 00:00:25,740

Susana.

00:00:29,060 --> 00:00:30,300

En el sótano.

00:00:31,260 --> 00:00:32,690

Hay un lavarropas viejo.

00:00:32,780 --> 00:00:34,900

P****, ¿por qué la cambiaste?

00:00:37,140 --> 00:00:38,620

Respeto a la doña, ¿eh?

00:00:39,820 --> 00:00:42,300

- Está ahí adentro.

- Sh, sh.

00:00:43,300 --> 00:00:44,260

¿Dónde está?

00:00:45,300 --> 00:00:48,060

- En el sótano, hay un lavarropas.

- ¿Dónde es el sótano?

00:00:48,940 --> 00:00:51,050

- Pasando la cocina.

- ¡No mirés!

00:00:51,140 --> 00:00:54,370

- ¿Dónde es el sótano?

- Pasando la cocina.

00:00:54,460 --> 00:00:55,690

Fondo a la derecha.

00:01:04,620 --> 00:01:05,780

Mirame, hija de puta.

00:01:06,300 --> 00:01:07,220

Mirame.

00:01:07,820 --> 00:01:09,210

- Mirame.

- No, pará.

00:01:09,300 --> 00:01:11,130

- ¿Qué "pará", la c**** de tu madre?

- Pará.

00:01:11,220 --> 00:01:12,610

- Mirame.

- No.

00:01:12,700 --> 00:01:14,010

- Quedate quiero.

- ¡No!

00:01:14,100 --> 00:01:17,050

- Matalo si se mueve.

- ¡No, no!

00:01:17,140 --> 00:01:18,850

- Vení para acá. ¡Vení para acá!

- ¡No!

00:01:18,940 --> 00:01:20,050

¿Qué no?

00:01:20,140 --> 00:01:21,850

Vení acá, mierda. Movete.

00:01:21,940 --> 00:01:23,090

¡Soltame!

00:01:23,180 --> 00:01:24,620

- ¡Soltame!

- ¡Callate la boca!

00:01:25,180 --> 00:01:27,290

¿Dónde está, b****?

¡La c**** de tu madre!

00:01:27,380 --> 00:01:29,810

- ¡Se la llevó!

- ¡Dejame, dejame!

00:01:31,580 --> 00:01:33,210

¿Qué hacés, b****? ¡Dale!

00:01:33,300 --> 00:01:34,930

- ¡Dale, dale! Tomátelas.

- Pará.

00:01:35,020 --> 00:01:36,450

- Dale, dale.

- Pará.

00:01:36,540 --> 00:01:38,690

¡Dale, dale, nos vamos, nos vamos!

00:01:46,100 --> 00:01:48,580

¡Dale!

Caso de prueba N°4: Situación normal.

00:00:08,841 --> 00:00:09,841

Hola, amor.

00:00:10,092 --> 00:00:11,092

¡Al fin!

00:00:12,219 --> 00:00:13,845

Ay, te extrañé un montón

00:00:14,305 --> 00:00:15,305

Bueno...

00:00:17,808 --> 00:00:20,268

- ¡Ahí está Maguila!

- ¡Maguila! ¡Maguila!

00:00:20,561 --> 00:00:23,480

- Che, está más gordo, ¿no?

- Sí, está regordo...

00:00:25,942 --> 00:00:26,942

Mamá

00:00:27,568 --> 00:00:28,568

¡Mamita!

00:00:33,699 --> 00:00:35,116

¡Vengan las dos!

00:00:36,077 --> 00:00:37,577

¡Vengan las tres!

00:00:39,288 --> 00:00:42,666

Mirá, este es para... ¡No, este!

00:00:43,000 --> 00:00:46,670

Este no, esto es para la ardilla,

esto es para la ardilla...

00:00:46,837 --> 00:00:48,296

Esto es para ustedes...

00:00:48,714 --> 00:00:49,839

Maguila, Mónica...

00:00:49,966 --> 00:00:51,675

Hola... ¡Bienvenido!

00:00:51,884 --> 00:00:53,259

Mucho gusto.

00:00:53,636 --> 00:00:54,803

Un placer.

00:01:02,812 --> 00:01:05,146

¿Creías que no iba a venir?

00:01:06,565 --> 00:01:07,691

Bienvenido, hijo.

00:01:07,942 --> 00:01:09,484

¿Qué hacés, pa?

00:01:18,536 --> 00:01:20,203

¿Te gusta, Maguila?

00:01:24,792 --> 00:01:25,792

¿Vamos?