Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Desarrollo de un sistema de extracción de terminología náutica en entornos multilingües

Grado en Ingeniería Mecánica

ALUMNO: Sergio Montes Vélez

DIRECTORA: Milagros Fernández Gavilanes

CURSO ACADÉMICO: 2020-2021

Universidade Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Desarrollo de un sistema de extracción de terminología náutica en entornos multilingües

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

Universidade Vigo

RESUMEN

Las comunicaciones en el mar han sido de vital importancia para los navegantes desde el inicio de los tiempos. Además, la incorporación de nuevas tecnologías a este ámbito ha producido un aumento exponencial de la cantidad de mensajes que se emiten a diario desde los barcos. Como consecuencia, los operadores humanos, en muchos casos, resultan insuficientes al no ser capaces de gestionar estos volúmenes de información.

Con el presente TFG se pretende realizar una aproximación a este problema, con la que se buscará desarrollar un sistema de apoyo a la motorización de dichos mensajes en un entorno multilingüe. Para ello, se usarán técnicas de procesamiento de lenguaje natural, apoyadas por el empleo de inteligencia artificial mediante algoritmos de aprendizaje automático supervisado con el fin de obtener un sistema capaz de detectar las comunicaciones relacionadas con el ámbito naval.
De manera adicional, se implementó un algoritmo que permite extraer información relevante que pueda resultar de potencial interés para los operadores de radio, que posteriormente será almacenada en un diccionario de términos náuticos con el léxico obtenido.

PALABRAS CLAVE

Aprendizaje automático, Inteligencia artificial, Procesamiento de lenguaje natural, Python

AGRADECIMIENTOS

A mi familia, por haberme apoyado y guiado a lo largo de mi vida de manera incondicional.

A mi tutora, Dª Milagros Fernández Gavilanes, por su ayuda e implicación prestadas durante el desarrollo de este TFG.

A la promoción 421-151 de la Escuela Naval Militar por acogerme con los brazos abiertos, en especial a la Vieja Guardia y nuestras numerosas noches de batalla.

Por último, pero no menos importante, a mi gran amigo Guille, por no haberme abandonado bajo ninguna circunstancia.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

La aparición de nuevas tecnologías a principios del siglo XX, unida a la necesidad de dominar la mar, dio lugar a la potenciación del comercio marítimo y a un aumento del tráfico, así como la necesidad de monitorizarlo [1]. En este contexto aparecen los primeros sistemas de comunicaciones con la finalidad de mejorar la seguridad marítima, la protección del medio marino y/o la zona costera adyacente, y la eficiencia de la navegación. Es cierto que al principio dicha necesidad de comunicarse podía subsanarse mediante el uso de balizas y otros objetos señalizadores luminosos, sin embargo, poco a poco estos sistemas fueron sustituidos por sistemas de comunicaciones por voz, como por ejemplo las comunicaciones por radio mediante VHF. Además, el paso de los años y la consecuente evolución de la tecnología propiciaron la migración de las comunicaciones analógicas hacia un escenario digital, en el cual se encuentran los sistemas que prevalecen en la actualidad, como pueden ser AIS[1] (*Automatic Identification System*).

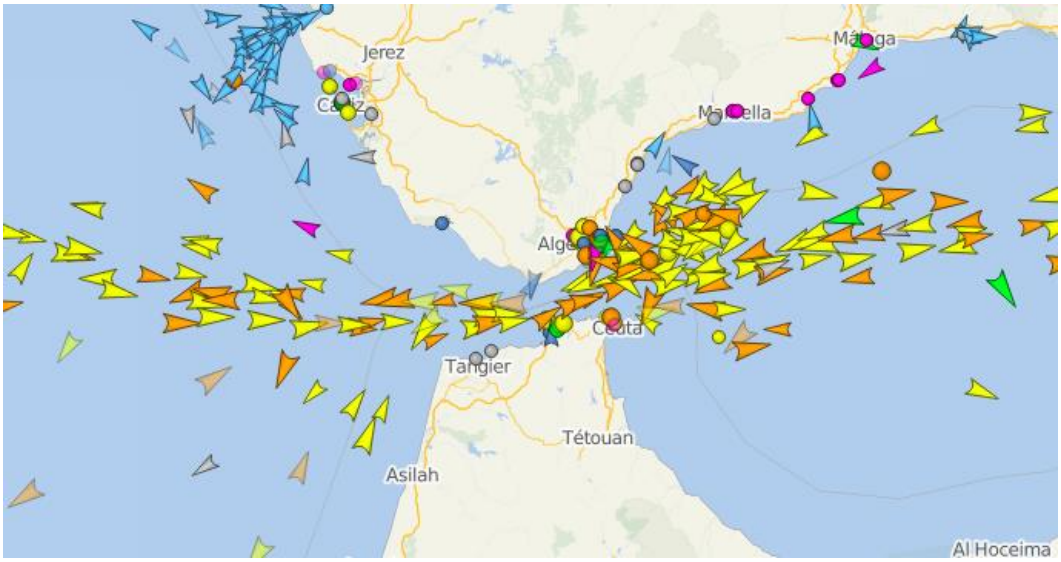


Figura 1-1: Imagen datos AIS del Estrecho del día 1/03/2021 (extraída de [2])

Hoy en día existe, además, una inquietud por comunicarse con propósitos más amplios de los que se acaban de citar, como puede ser el establecer contacto de carácter personal desde una embarcación

[1] Se trata de un sistema para ver la posición, rumbo y velocidad de una embarcación, además de otros datos, de manera totalmente automática. [73]

por parte de los integrantes de la tripulación o los pasajeros o llevar a cabo una serie de preguntas por parte de las Fuerzas de un Estado para asegurar el cumplimiento de las actuales leyes.

Además, por su situación geográfica, España se encuentra en unos de los puntos más importantes del mundo en lo que a tráfico marítimo se refiere. Concretamente, para el caso del estrecho de Gibraltar, según fuentes externas [3], son alrededor de 100.000 barcos los que circulan por el mismo al año, lo que representan un 10% del tráfico marítimo mundial.

1.2 Objetivos

Debido a la gran cantidad de tráfico marítimo que concurre a diario en el Estrecho, se produce una gran cantidad de comunicaciones por voz vía VHF, UHF, etc. que hace que sea prácticamente imposible analizar y procesar en tiempo real este basto volumen de información por parte de personas. Además, teniendo en cuenta la falta de personal en estaciones de radio y que, un operador se encuentra limitado por sus capacidades de procesado, se hace más que evidente la necesidad de implementar algún tipo de sistema informático capaz de procesar los mensajes de voz en tiempo real con el fin de ayudar a los operadores en su función de análisis, procesamiento y clasificación de estos mensajes.

Por tanto, con el presente trabajo se pretende construir un sistema desarrollado en Python que sea capaz de analizar los mensajes de voz en un entorno multilingüe, haciendo uso de técnicas de Procesamiento de Lenguaje Natural (PLN), así como de algoritmos de aprendizaje automático (*Machine Learning*) con el fin de extraer aquellos mensajes que podrían resultar ser importantes desde un punto de vista náutico y de este modo facilitar la tarea de filtración de mensajes relevantes, de tal forma que permita seleccionar las conversaciones de potencial interés para el operador humano. De este modo se pretende aumentar las capacidades de análisis del operador, permitiéndole centrar su atención en aquellas comunicaciones de radio donde realmente es necesaria.

En este sentido, el audio se procesará y se transcribirá a texto para, a continuación, realizar el proceso de clasificación y extracción de la terminología en los mismos. Debido al tiempo limitado que supone la realización de este TFG se asumirá que las conversaciones radio en frecuencia VHF serán conversaciones a través del micro del ordenador. Somos conscientes de que la utilización de una frecuencia VHF implicaría una parte de ruido en la señal de entrada. Sin embargo, el objetivo del trabajo no está en relación con el tratamiento del audio, sino en la correcta determinación de su contenido una vez este transcrito a texto.

1.3 Organización de la memoria

La memoria de este TFG está dividida en cinco capítulos, complementados con cinco anexos.

En el presente capítulo se describe una breve introducción sobre las comunicaciones en el mar y su imperiosa razón de ser, seguido de los objetivos marcados para el TFG y la organización de la memoria.

En el Capítulo 2 desarrollamos el estado del arte. Se mencionan algunos trabajos relacionados con el ámbito de este proyecto y se definen conceptos como inteligencia artificial. Además, se comenta el lenguaje de programación usado para llevar a cabo este trabajo.

El Capítulo 3 se divide en cuatro partes. En primer lugar, se establece el entorno de trabajo, seguido de la explicación de la metodología empleada para crear ese detector de odio y finalizando por la explicación de las funciones que se implementan para llevarlo a cabo.

En el Capítulo 4 se expondrán los resultados experimentales obtenidos, así como la evaluación de todos los algoritmos utilizados.

Por último, en el Capítulo 5 se concluirá con unas breves conclusiones, además de unas líneas futuras.

2 ESTADO DEL ARTE

2.1 Trabajos relacionados con el ámbito

Trabajos recientes en esta línea como el proyecto ARTUS [14] han explorado la posibilidad de transcribir conversaciones en la banda marítima VHF y destacan su posible utilidad como herramienta de apoyo de las misiones de búsqueda y rescate (SAR), mientras que la aplicación de técnicas de reconocimiento de voz al análisis y transcripción de señales de llamada en conversaciones radiofónicas con aviones se explora en [15]. También podrían ser de utilidad estudios, corpus, herramientas y técnicas desarrolladas previamente en el ámbito de la lingüística y el procesamiento de lenguaje natural, como por ejemplo las investigaciones llevadas a cabo en [4], que, mediante diferentes experimentos, confirmaron que el uso de un dataset específico interviene de forma muy positiva en el resultado de los algoritmos. Además, remarcaron que la sustracción de elementos de puntuación no afectaba a los resultados, ya que para algunos idiomas como puede ser el inglés, estos signos no son tan relevantes. También son interesantes trabajos como [5] en el que se estudia el uso de la lengua inglesa y su relevancia en las comunicaciones navales tanto en la mar como en estaciones costeras debido a que dicho idioma se considera el lenguaje normalizado en la mar. Se analizan los puntos en común y las características generales que predominan en eventos comunicativos marítimos.

El reconocimiento de entidades nombradas es un componente clave para la extracción de información relevante. En artículos como en [6] se habla sobre la creciente cantidad de información en forma de texto que se está generando a diario. Dicho volumen de información no se puede procesar de forma sencilla y es aquí donde juegan un papel muy importante los procesos de "*text mining"* que consisten en la extracción de la información relevante dentro de un texto. Dichas técnicas han cobrado un gran protagonismo en los últimos años y algunas de estas técnicas son el preprocesamiento de textos, su clasificación y clustering (explicado en Anexo IV: Clustering de términos náuticos).

2.2 Inteligencia artificial

Tal y como la Revolución Industrial del siglo XIX tiene su origen en el incremento de la capacidad del hombre por el empleo de las máquinas, se puede hablar de una Segunda Revolución provocada por la aparición de otro tipo de máquinas: los ordenadores. Estos surgieron en la segunda mitad del siglo XX, pero, a diferencia de la anterior revolución, se caracteriza por el uso de las máquinas con el fin de sustituir al hombre en actividades de tipo intelectual y no físicas.

El uso de los ordenadores ha aumentado con el paso del tiempo y han irrumpido con gran fuerza en la vida cotidiana. Tanto es así que actualmente no nos sorprende la opción de usar el ordenador o

incluso el teléfono móvil para llevar a cabo complicados procesos de manejo de información o desarrollos para obtener ayuda en determinadas decisiones, o incluso comunicarnos con cualquier parte del mundo en tiempo real.

La pregunta de si los ordenadores pueden llegar a pensar y razonar es algo que se lleva planteando desde tiempo, antes incluso de la globalización de los mismos. Autores, entre ellos Alan Turing, llegaron a cuestionárselo muy seriamente, a la vez que estaban madurando la idea de ordenador, dando por hecho que estos podrían llegar a materializar en última instancia la construcción de objetos inteligentes, hechos que más caracterizan al ser humano [7]. Para conocer más acerca de la historia de la IA Anexo II: Historia de la IA

Para el autor Lasse Rouhiainen [8], la inteligencia artificial podría definirse como "*la habilidad de los ordenadores para hacer actividades que normalmente requieren inteligencia humana*".

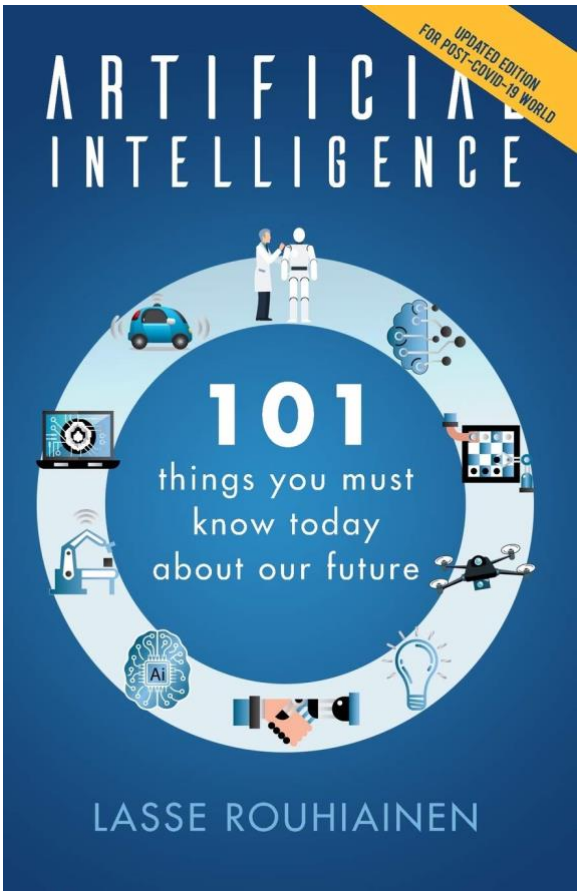


Figura 2-1 Libro del autor Lasee Rouhiainen (extraída de [9])

Sin embargo, dicha definición podría resultar incompleta, y por ello muchos autores afirman que la inteligencia artificial es la capacidad que tienen las máquinas para usar algoritmos y códigos para, aprender utilizando un conjunto de datos y posteriormente realizar una predicción en función a lo aprendido de la misma manera que un ser humano lo haría. Una de las diferencias más importantes entre un dispositivo basado en Inteligencia Artificial (IA) y un humano, es que los primeros no necesitan descansar y son capaces de trabajar y analizar grandes volúmenes de información al mismo tiempo. Además, los errores que presentan dichos sistemas son significativamente menores.

El hecho de que los ordenadores y softwares puedan aprender y tomar decisiones por ellos mismos, está cobrando capital importancia ya que su uso y procesos se encuentran en auge en los últimos años [8]. Debido a estas capacidades, los sistemas de IA han empezado a realizar tareas que

antes las llevaban a cabo los humanos ya que la sociedad puede beneficiarse de esta sustitución y disponer de un rendimiento y eficiencia mejores.

Algunos de los campos en los que puede aplicarse la IA son los siguientes.

- Reconcomiendo de imágenes, etiquetado y clasificación.
- Mejoras del desempeño de la estrategia comercial.
- Procesamiento eficiente y escalable de datos de pacientes médicos.
- Detección y clasificación de objetos.
- Distribución de contenido en las redes e internet.
- Protección contra amenazas de seguridad cibernética.
- Análisis y procesamiento de texto con el fin de extraer información o conclusiones.

Los sistemas de IA, en un futuro también serán capaces de ayudarnos en la toma de decisiones relacionadas con asuntos importantes de nuestra vida, como por ejemplo la salud, la educación, las finanzas o incluso las relaciones interpersonales.

Otra ventaja de la IA es que las máquinas realicen los trabajos que los humanos consideran difíciles o tediosos, incluso peligrosos. En algunas ocasiones, según diversos estudios, el término "Inteligencia Artificial" puede generar cierta incomodidad en la sociedad y es por ello que expertos como Sebastian Thrun piensan que sería mejor darle el nombre de "ciencia de datos".

El aprendizaje, aunque es una característica primordial de la inteligencia humana, es un fenómeno mal conocido y es por ello que no existe una forma sencilla de construir procesos automáticos de aprendizaje basados en los propios procedimientos humanos [7].

Una de las grandes dificultades a las que tienen que enfrentarse los investigadores del área de aprendizaje automático es la diferencia entre la memoria del ordenador y la humana. Sin profundizar en la materia, se puede observar que el ser humano no recupera fácilmente información memorizada, pero, por el contrario, un ordenador es capaz de acceder a ella de manera casi instantánea, para las máquinas esto se trata de un proceso casi trivial. Por otro lado, los seres humanos somos capaces de diferenciar entre la información que es realmente importante y los datos que no lo son; se puede memorizar, almacenar, recuperar y manejar información incompleta o con ruido a partir de estímulos mal formulados. El punto clave es ser capaces de incorporar estas cualidades a los procesos de aprendizaje automático, teniendo en cuenta que la forma en que llevamos a cabo estos procesos no es conocida en su totalidad.

Según otros autores, el aprendizaje automático, en inglés "*machine learning*", es uno de los campos principales en los que se centra la inteligencia artificial. Consiste en un aspecto de la informática en que las máquinas u ordenadores disponen de la capacidad de aprender sin estar programados para ese fin [8]. Un ejemplo típico son las predicciones en una situación particular, o como en el caso de este TFG, la estimación de si una oración en concreto es náutica o no.

Machine learning utiliza algoritmos para aprender con la información que extrae o lee de los datos. Un claro ejemplo de la implementación de *machine learning* en el día a día es el filtro de correo no deseado del que disponen los servicios de correo electrónico.

Tal y como muestra la Figura 2-2, son tres los tipos o subgrupos de aprendizaje automático que existen, el supervisado, el no supervisado y el de refuerzo.

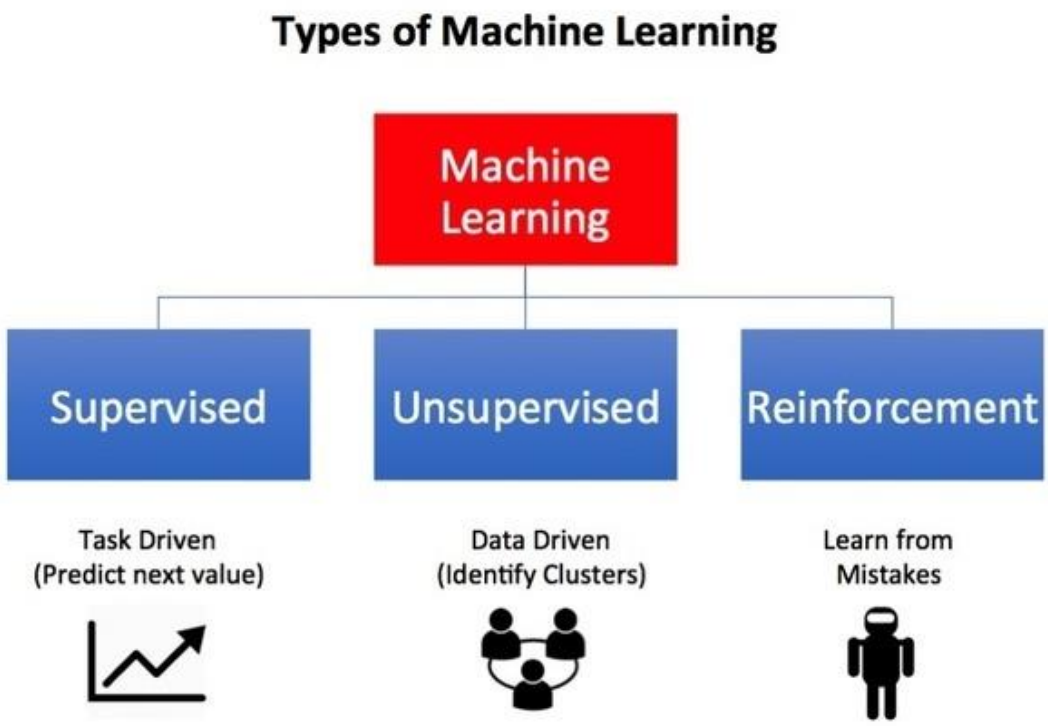


Figura 2-2 Tipos de aprendizaje automático (extraída de [11])

- En el aprendizaje supervisado, los algoritmos hacen uso de datos que han sido organizados y etiquetados o clasificados previamente (dataset) con el fin de poder aprender a etiquetar futura información. En este método se requiere la acción del ser humano para proporcionar retroalimentación, así como los parámetros para entrenar y predecir.
- En el aprendizaje no supervisado, a diferencia del anterior, los algoritmos no emplean datos etiquetados ni organizados, sino que deben encontrar la manera de poder clasificar la nueva información por ellos solos.
- En el caso de aprendizaje por refuerzo la máquina aprende de la experiencia, y para ello debemos hacerle saber cuándo las predicciones son correctas y cuando no.

Para el desarrollo de este TFG, se han hecho uso de los siguientes algoritmos pertenecientes al aprendizaje de tipo supervisado.

Logistic Regression (LR)

Es un algoritmo de clasificación que se usa para intentar predecir la probabilidad de una variable dependiente, en este caso texto, que además es categórica. Este modelo permite predecir que la presencia de un factor aumenta la probabilidad de un resultado en concreto. Uno de los aspectos negativos es que requiere de una muestra de datos amplia. Sin embargo, como punto positivo se puede destacar su eficiencia y pocos requisitos computacionales, lo que hacen que sea un algoritmo liviano para las máquinas [10].

Multinomial Naive Bayes (MNB)

Se trata de uno de los modelos probabilistas más sencillos y usados en la clasificación de textos y documentos debido a que produce resultados tan buenos o incluso mejores que otros modelos más sofisticados.

Este modelo se basa en la Regla de Bayes para intentar predecir la probabilidad condicional de que un documento, o texto en este caso, pertenezca a una clase determinada. Además, de manera adicional, en el caso de Naive Bayes Multinomial tiene en cuenta la frecuencia en que aparece cada término en los textos [11].

Passive Agressive Classifier (PAC)

Este algoritmo de aprendizaje automático no es uno de los más conocidos. Sin embargo, ha demostrado tener un rendimiento bastante alto en ciertas aplicaciones. Este tipo de modelo suele ser usado en aprendizaje a gran escala. Además, los datos de entrada suelen tomarse de manera secuencial mientras que el modelo se actualiza poco a poco, al contrario de lo que ocurre en otros tipos de algoritmos en donde el modelo utiliza toda la información a la vez. Esto puede resultar especialmente útil en los casos en donde el dataset es especialmente extenso o cuando no se disponga de una máquina con gran tamaño de memoria [12].

Random Forest Classifier (RFC)

Un modelo *Random Forest* está constituido por un conjunto de árboles de decisión individuales y diferentes entre ellos, entrenados con muestras ligeramente distintas construidos con los datos de entrenamiento y las predicciones que se llevan a cabo se obtienen agregando las predicciones de todos los árboles anteriormente mencionados. Este tipo de algoritmo se ha convertido en uno de los más usados dentro del ámbito en que se encuadra este proyecto, es decir, la clasificación de textos [13].

2.3 El lenguaje

El lenguaje es el principal medio de comunicación, si no el único, entre los seres vivos. Gracias a él, los humanos somos capaces de interactuar entre nosotros y de transmitir los conocimientos a lo largo de las generaciones. Este lenguaje contiene los elementos necesarios que permiten dicha interacción y pueden ser palabras, señas o sonidos, es decir, elementos por si solos abstractos pero que cobran sentido y adquieren significado cuando se introducen en el ámbito adecuado (sociedad e individuos).

Haciendo una analogía, podría decirse que los ordenadores son la sociedad, las piezas del hardware que lo componen son los individuos y el software es el lenguaje que permite la comunicación entre las diferentes partes.

Basándonos en la definición más clásica de Inteligencia, como la de Turing, la generación y comprensión de oraciones con sentido en lenguaje natural es una actividad fundamental para ser capaces de calificar un sistema como inteligente y es por ello que dicho ámbito dentro de la Inteligencia Artificial ha recibido tanta atención a lo largo de los años. Además, a las razones teóricas que motivan dicho interés, habría que añadir las razones prácticas ya que, si los ordenadores fueran capaces de procesar el lenguaje natural de la misma manera que lo hacemos los humanos nuestra comunicación con ellos se vería enormemente facilitada [7]. Con el fin de intentar llegar a comprender mejor el nivel de complejidad que encarna el procesamiento del lenguaje, en la Figura 2-3 se puede observar una pequeña esquematización simplificada del proceso que tendría lugar en la mente humana.

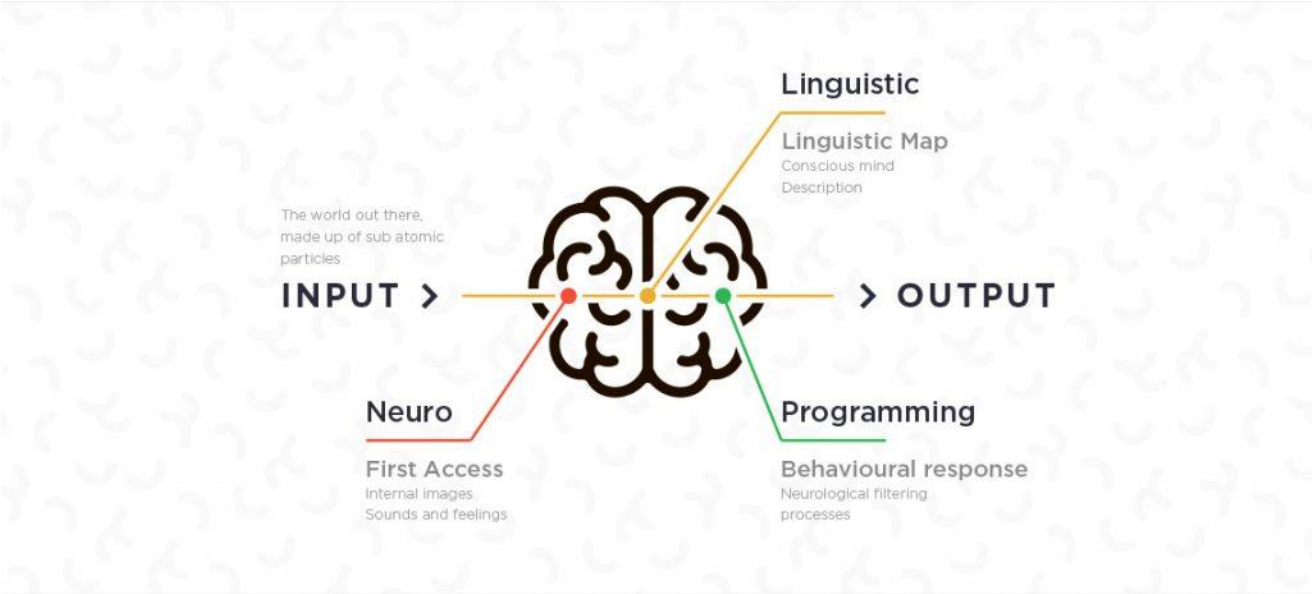


Figura 2-3 Procesamiento del lenguaje en humanos (extraída de [12])

Para llegar a comprender las dificultades que entraña la automatización del procesamiento del lenguaje, es suficiente mencionar que nuestro lenguaje es sumamente informal y en ocasiones ambiguo, incompleto y poco preciso a pesar de que las oraciones estén correctamente formuladas.

Entre los primeros intentos de comprensión del lenguaje que se realizaron dentro del campo de la IA, se encuentra el diseño de programas para la traducción automática. Desde el año 1956 se contaba con programas que era capaces de traducir automáticamente del ruso al inglés, aunque no de la manera más fiel. Estos programas en realidad no era más que diccionarios y posteriormente se comprobó que no era suficiente con añadir un poco de gramática a un léxico para logar un buen resultado, sino que además era necesaria la compresión del mismo texto a traducir ya que las palabras y las oraciones pueden cambiar de significado en función de la intención del hablante. Por otro lado, se plantea el problema de que un programa de traducción basada en un diccionario está estrechamente ligado a dos lenguas concretas en un orden determinado y es por ello que la traducción entre varios idiomas requiere también el uso de varios programas.

Una vez encontradas y recogidas estas dificultadas, el informe ALPAC [14], emitido por un conjunto de expertos del gobierno norteamericano, establecía la imposibilidad de obtener resultados decentes en este campo, lo que produjo que los fondos gubernamentales destinados a este campo se retiraran, con el consiguiente parón de las investigaciones relacionadas. No obstante, en esa misma época R. Shank había sido capaz de encontrar un nuevo enfoque del problema que, a la larga demostró ser tremendamente beneficioso.

La idea de este autor era llevar a cabo las traducciones pasando previamente por un intermediario, que se puede entender como la representación del sentido de las oraciones independientemente de la estructura superficial de la lengua natural en que estaba escrito dicho texto. De esta manera para traducir serían necesarios dos programas, un “analizador” que fuera capaz de extraer el sentido y el significado del texto y un “generador” que fuera capaz de representar el significado de las oraciones haciendo uso del lenguaje natural. Con esta forma de abordar el problema se resuelven las dificultades apuntadas anteriormente, primero se traduce pasando por la comprensión de los textos, lo que permitiría obtener buenas traducciones para posteriormente trabajar en cualquier idioma, reduciendo así de manera significativa la cantidad de programas necesarios para traducir los mismos idiomas.

A partir de la década de los setenta, y debido a la proliferación y el aumento de tamaño de los Sistemas de Bases de Datos, aparece un nuevo problema ligado precisamente al procesamiento del lenguaje natural, y este no es otro que la búsqueda y recuperación inteligente de información. Esta

dificultad alcanzó en los noventa una trascendencia importante como consecuencia de la gran expansión de las posibilidades de consulta que produjeron las comunicaciones en red. Además, el desarrollo de Sistemas inteligentes para el manejo de la información es una línea vanguardista de gran importancia en áreas como la salud, gestión de documentación o gestiones bancarias entre otras. Es por ello que se le ha bautizado con el nombre de "*Data Mining*".

El Procesamiento del Lenguaje Natural o "*PLN*". El PLN [15] es el campo de estudio que se enfoca en la compresión mediante ordenador del lenguaje humano. Incluye parte de la Ciencia de Datos, Inteligencia Artificial (más concretamente el Aprendizaje Automático) y la Lingüística. Los ordenadores analizan el lenguaje humano, lo procesan y lo interpretan con el fin de poder usarlo de una manera práctica. Gracias a esto, haciendo uso del aprendizaje automático, el programa es capaz de etiquetar las oraciones como náuticas o no, por ejemplo. En la Figura 2-4 Esquema funcionamiento etiquetado de texto (Extraída de [16] puede verse un esquema de cómo funciona el etiquetado de textos haciendo uso de PLN

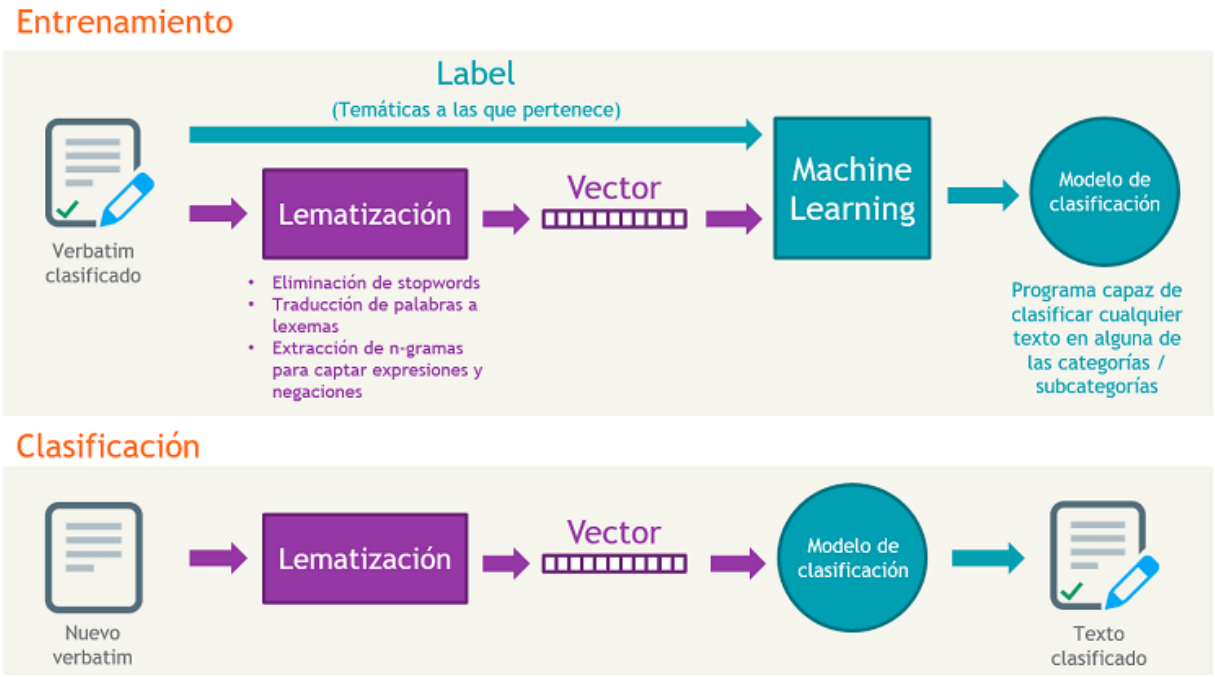


Figura 2-4 Esquema funcionamiento etiquetado de texto (Extraída de [16])

Lo que se pretende conseguir con el PLN, puede ir desde traducir de un idioma a otro hasta comprender o representar el contenido de un texto, crear una base de datos, crear resúmenes o incluso mantener un diálogo con los usuarios como parte de una interfaz, pero conseguir todos estos objetivos es una tarea sumamente compleja. No es nada sencillo determinar si algún día sabremos si un ordenador realmente entiende nuestro idioma, lo único que podemos intentar es demostrar que el sistema parece entender el lenguaje en función de si es capaz de llevar a cabo o no alguno de los objetivos mencionados anteriormente.

Nuevos caminos de investigación y desarrollo están consiguiendo construir modelos capaces de demostrar una mayor sensibilidad para interpretar el lenguaje humano. La forma más habitual de llevar a cabo este proceso es la realización de pruebas empíricas de las cuales se pueden extraer valores de rendimiento.

El PLN es considerado uno de los grandes retos de la inteligencia artificial, debido a que se trata de una tarea complicada, además, a menudo el ser humano en su forma de usar el lenguaje incluye chistes, ironías, neologismos y otras muchas técnicas literarias que cuentan con un significado intrínseco que cambia por completo el sentido de las oraciones. Por ello para el PLN no es suficiente

con comprender el significado de ciertas palabras, si no que el programa debe ser capaz de entender el significado global de una oración o incluso texto. De hecho, la mayor dificultad que presenta el procesamiento del lenguaje natural es precisamente discernir entre la ambigüedad y la mejor forma de entenderlo es mediante ejemplos para los diferentes problemas que debe resolver un sistema de PLN:

- Ambigüedad léxica: Distinguir entre las diferentes categorías gramaticales ya que un mismo término puede desempeñar funciones de sustantivo o verbo, por ejemplo.
- Ambigüedad estructural: La frase "me he cortado el pelo" puede significar que he ido a un centro especializado para que me corten el pelo, pero también podría significar que me he cortado el pelo yo mismo.
- Ambigüedad semántica: Hay muchas palabras con más de una entrada en el diccionario.
- Ambigüedad pragmática: Como puede ser el caso de preguntas en las que no esperamos una respuesta verbal si no que un sujeto realice una acción determinada, por ejemplo, ¿Puedes cerrar la puerta?
- Ambigüedad en referencias: Este es el caso de los pronombres que sustituyen nombres propios.

Por supuesto todas estas formas de ambigüedad pueden interactuar entre ellas de tal forma que se genere un problema mucho más complejo [17].

2.4 El lenguaje de programación

Dicho de manera sencilla, el lenguaje de programación es el conjunto de instrucciones y mecanismos a través de los cuales los seres humanos somos capaces de interactuar con los ordenadores. Para ellos hacemos uso de algoritmos e instrucciones escritas basadas en sintaxis que el ordenador es capaz de entender e interpretar.

Estos lenguajes de programación permiten a las computadoras procesar de manera rápida y eficaz grandes y complejas cantidades de información y datos, es por ello uno de los motivos o causas por las que la sociedad ha evolucionado a gran escala en los últimos años. Múltiples lenguajes de programación son los usados hoy en día, entre ellos destacan C++, C#, Visual Basic, Go, Ruby, JavaScript, Java o Python. [18], tal y como se observa en la Figura 2-5. Para más información, consultar el Anexo I: Historia y evolución de los lenguajes de programación

Figura 2-5: Lenguajes de programación (extraída de [19])

A este respecto, Python es un lenguaje de programación de alto nivel, interpretado y multipropósito [20]. En los últimos años su uso se ha visto incrementado y actualmente es uno de los lenguajes de programación más empleados para el desarrollo del software. Para más información sobre la historia de Python Anexo I: Historia y evolución de los lenguajes de programación

Además, Python puede ser usado en múltiples plataformas y sistemas operativos, incluyendo los más populares como Mac OS X, Linux o Windows. Pero no solo eso, Python también funciona en smartphones, ya que Nokia desarrolló un intérprete de este lenguaje para su sistema operativo Symbian.

Muchos de los lenguajes de programación existentes tienen o fueron diseñados para un ámbito específico, como por ejemplo con PHP que fue creado con la idea de desarrollar aplicaciones web. Pero este no es el caso de Python. Con Python es posible desarrollar software para aplicaciones científicas, para comunicaciones de red, aplicaciones de escritorio con interfaz gráfica de usuario (GUI), para crear juegos y por supuesto, para aplicaciones web también.

Figura 2-6 Logotipo Python (extraída de [21])

Tanto es así, que empresas y organizaciones multinacionales de la entidad de la *NASA*, *Google* o *Nokia* entre otras, usan de forma intensiva de Python para desarrollar sus servicios y productos. Así pues, queda demostrado que este lenguaje puede ser usado en diversos tipos de sectores muy diferentes y por tanto se trata de uno de los lenguajes más versátiles y potentes.

Por ello, entre las principales razones para elegir Python y no otro, son muchos los que justifican que sus principales características lo convierten en un lenguaje muy productivo. Y es que, estamos hablando de un lenguaje que destaca por tu potencia, flexibilidad y su sintaxis clara y concisa. Además, no requiere dedicas tiempo a su compilación ya que es un lenguaje interpretado. Python es además '*open source*' y no es necesario pagar ninguna licencia para distribuir ningún tipo de software desarrollado con este lenguaje. Es más, incluso su intérprete se distribuye de manera gratuita en las diferentes plataformas.

Por último, la versión más actualizada de Python (usada para este proyecto) recibe muchos nombres, entre ellos Python 3000 y Py3K, aunque la más conocido y la empleada habitualmente simplemente se denomina Python 3.

Python es un lenguaje muy popular utilizado en varias tareas como Inteligencia artificial, Ciencia de datos (Data Science), Web. Actualmente, el uso de Python es muy extendido por la interfaz que facilita la programación, la rapidez de ejecución y la integración con otros sistemas fácilmente [40].

Característica de Python

Algunas de las principales características de Python son, por una parte, que es interpretado y, por otra, que es multiplataforma. La primera significa que no hace falta compilar el código para su ejecución, ya que dispone de un intérprete encargado de leer el fichero y ejecutarlo. Debido a este funcionamiento se puede ejecutar el mismo código en diferentes plataformas y sistemas operativos sin necesidad de modificar o adaptar el código fuente, basta con tener dicho intérprete

instalado.

Generalmente a los programas en Python se les denomina scripts, pero script es el término que se suele utilizar para los códigos fuente escritos en Python, por tanto, un programa puede contener uno o más scripts.

Los usuarios de Python, suelen referirse generalmente con este nombre (Python) a ambos, en lenguaje y el intérprete. Es importante tenerlo en cuenta porque en numerosas ocasiones cuando los programadores hablan sobre las versiones de Python hacen mención directa al intérprete y no al lenguaje.

Para interactuar con dicho intérprete a través de la consola solo es necesario instalar un componente llamado *Shell*, que permite que el código Python sea ejecutado directamente haciendo uso del terminal o interfaz de comandos

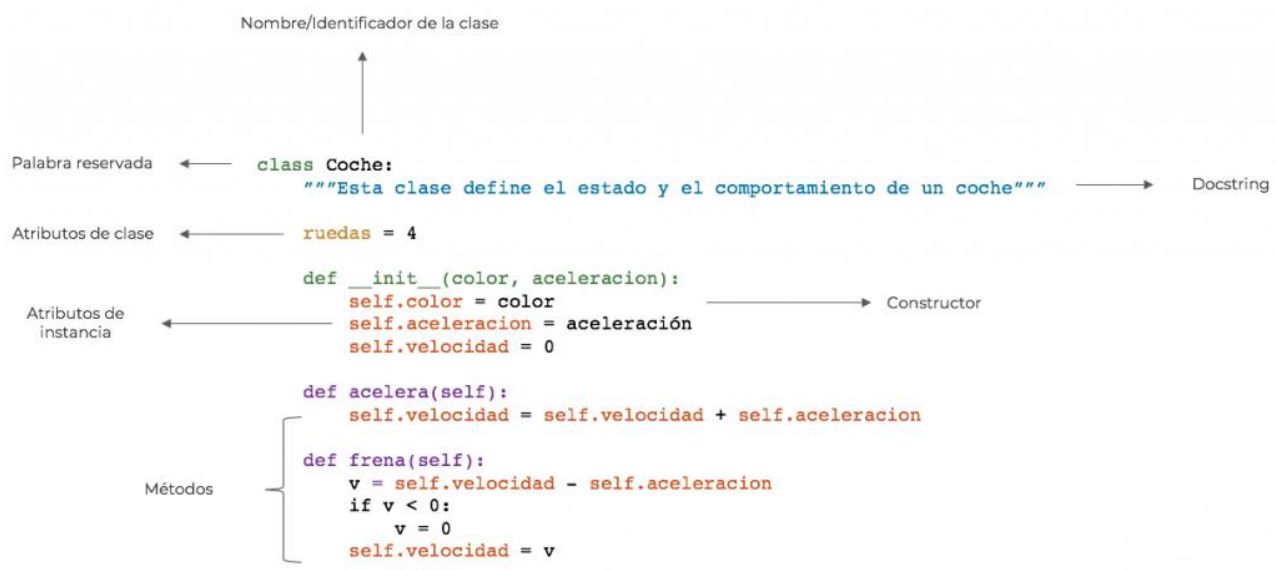


Figura 2-7: Ejemplo de objetos de Python (extraída de [22])

En lo que a la sintaxis del lenguaje se refiere, cabe reseñar su simplicidad, ya que, gracias a ella resulta bastante sencillo escribir código de fácil lectura. Este es uno de los factores más importantes porque no solo facilita el aprendizaje, sino que además hace que mantener el código sea más viable.

A diferencia de otros lenguajes de programación como C++ o Java cuyos usuarios acostumbran a declarar cada variable con su respectivo tipo en concreto, en Python este paso no es necesario, debido a que el tipo de cada variable queda definido en el mismo momento de su asignación. Como resultado, una variable tiene la posibilidad de cambiar de tipo a lo largo del ciclo de vida del programa sin tener que ser declarado de manera explícita.

Python, además de soportar la orientación a objetos, permite usar otras herramientas de programación, como la programación funcional y la imperativa. Actualmente, Python se ha colocado como uno de los lenguajes que más facilidades ofrecen al usuario para enseñar programación orientada a objetos. A esto contribuyen su sintaxis, el soporte para la implementación de herencia sencilla y múltiple y los mecanismos de introspección

Este lenguaje de programación, con el fin de facilitar su uso, incorpora una serie de estructuras de datos de alto nivel, como por ejemplo las listas, los diccionarios, cadenas de texto o strings, tuplas y conjuntos. Por otro lado, su librería por defecto incluye una gran variedad de funciones que pueden ser usadas en diferentes ámbitos. Estas funciones van desde las más

básicas para el manejo de strings, hasta las que pueden utilizarse en programación criptográfica, sin olvidar las de nivel intermedio, como es el caso de las que permiten el manejo de ficheros ZIP o realizar comunicaciones de red a través de diferentes protocolos. Todo ello, por supuesto, sin tener que instalar librerías adicionales.

Otra de las características que destacan de este lenguaje, es su sencillez para interactuar con otros lenguajes. Esto se puede realizar gracias a las extensiones y a los módulos. Este aspecto resulta interesante cuando disponemos de un programa escrito, por ejemplo, en C++ y queremos implementar en nuestro programa alguna función específica de Python. En lugar de tener que reescribir el programa de C++ en Python, simplemente podemos comunicar ambos haciendo uso de esta interfaz con la que cuenta Python.

También debemos mencionar la capacidad que tiene Python de ser ejecutado en diferentes sistemas acondicionados para ello. De todas, la implementación más popular es conocida con el nombre de CPython, escrita C, aunque existen otras equivalentes para otros lenguajes como Jython para Java e IronPython para la plataforma .NET de Microsoft.

Figura 2-8 Logotipo de Jython (extraída de [23])

3 DESARROLLO DEL TFG

3.1 Entorno de trabajo

Atendiendo a los resultados proporcionados en la Figura 3-1, se ha decidido usar Python como lenguaje de programación debido a su facilidad de uso por ser un lenguaje libre y multiplataforma. Además, según varias fuentes, se prevé que sea uno de los lenguajes más usados en el futuro, y más aún en el ámbito de la inteligencia artificial y *machine learning* ya que en la actualidad existe gran número de proyectos basados en Python.

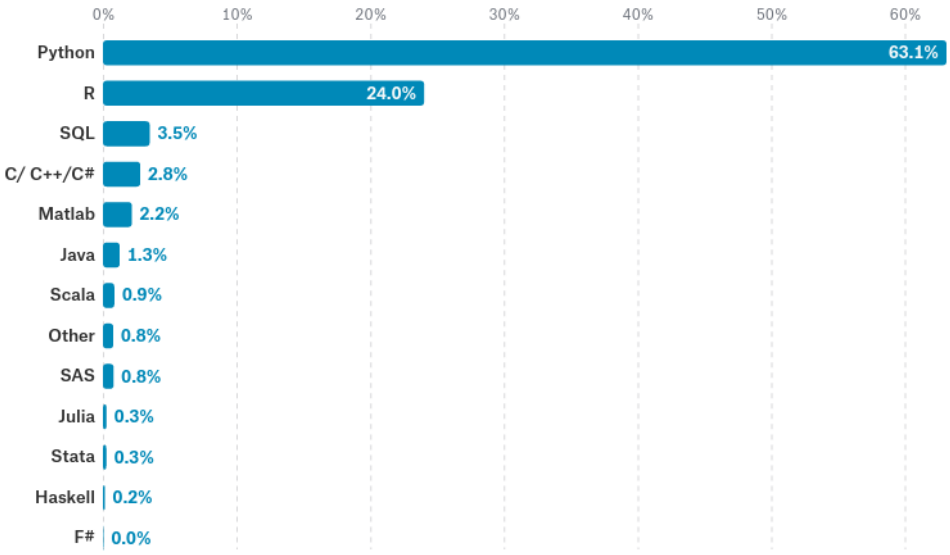


Figura 3-1: Recomendación lenguajes de programación por científicos (extraído de [24])

Es habitual que los usuarios de Windows hagan uso de una máquina virtual Ubuntu para su utilización, pero en mi caso al disponer de un Mac con sistema operativo MacOs Big Sur en su versión 11.1, con el fin de facilitar el proceso, decidí trabajar sobre el mismo.

El primer paso es entonces instalar Python. El sistema operativo mencionado anteriormente cuenta con la versión de Python 2.7, la cual quedará obsoleta y sin soporte próximamente, y es por ello que decidí usar la versión de 3.9.1 en su lugar, una de las más nuevas y compatible con todas las librerías necesarias.

Para ello utilicé la herramienta "*Homebrew*" [25], utilizando para ello el tutorial de la web "*OpenSource"* [26]. También fue necesario instalar "*conda"* haciendo uso de "*miniconda*" que se trata

de un pequeño instalador del mismo que incluye los paquetes imprescindibles para su uso [27] con el fin de poder visualizar algunos gráficos ejecutando el código directamente desde el terminal sin la necesidad de usar "*Jupyter*".

Con el comando **`pyenv versions`** y **`conda -V`** podemos comprobar las versiones instaladas de los mismos. En mi caso, puede observarse en la Figura 3-2

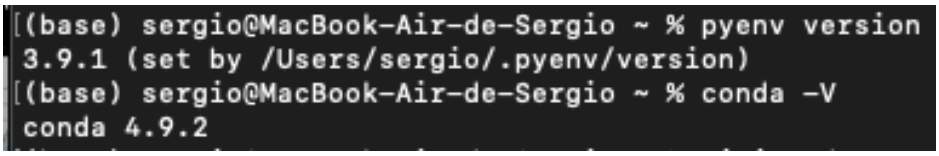


Figura 3-2 Versiones de pyenv y conda (Propia)

Una vez que el sistema está preparado para trabajar con Python, el siguiente paso es instalar un editor de código que nos permita trabajar cómodamente. En esta ocasión el elegido es "*Visual Studio Code*" [28] cuyo logotipo puede verse en la Figura 3-3, desarrollado por Microsoft y que incluye una gran cantidad de plugins que resultan de gran utilidad a la hora de escribir código fuente. Además, desde la AppStore de Apple instalé el editor "*XCode*" que, aunque finalmente no fue necesario su uso, incorpora una serie de características para desarrolladores que mejoran la integración de Python en MacOs.

Figura 3-3 Logo Visual Studio Code (extraída de [29])

3.2 Librerías

El objetivo de este apartado es definir brevemente qué es una librería de Python, así como listar las que han sido necesarias para la consecución de este trabajo.

Entrando en materia, una librería de Python podría definirse como un conjunto de implementaciones funcionales, es decir, un conjunto de módulos cuya agrupación tiene una finalidad específica y el objetivo de implementarlas es facilitar el proceso de creación de los programas [30]. Las librerías que han sido necesarias son, por orden temática:

Funciones matemáticas

- **Re**: Se trata de un módulo que presenta funciones que permiten trabajar con expresiones regulares o cadenas [31].
- **Itertools**: Es un módulo de la librería estándar de Python que incorpora funciones que devuelven objetos iterables, es decir, elementos que pueden recorrerse secuencialmente y utilizarse en procesos iterativos como bucles. Estas funciones fueron creadas con el fin de conseguir una ejecución rápida y eficiente de los algoritmos basados en bucles [32].
- **Numpy**: Es una librería de Python especializada en el cálculo numérico y análisis de datos, en concreto para un gran volumen de datos. Incorpora una nueva clase de objetos, *arrays*, que permiten representar colecciones de datos del mismo tipo en varias dimensiones como si de una matriz se tratara, además de incluir una serie de funciones muy eficientes para su manipulación [33].

Acceso a archivos

- **Pandas**: [34] Librería especializada en el manejo y análisis de datos estructurados. Sus principales características son:

    - Permite leer y escribir ficheros en formato CSV entre otros
    - Permite acceder a los datos de los ficheros mediante sus índices o nombres de files/columnas.
    - Ofrece métodos para reordenar, dividir y combinar conjuntos de datos.

Representaciones gráficas

- **Scikit-learn**: Cuenta con algoritmos de clasificación, regresión, clustering y reducción de dimensionalidad. Además, es compatible con otras librerías tales como Numpy o Matplotlib, mencionadas anteriormente. Scikit-learn se convierte entonces en la herramienta básica para empezar a programar y estructurar los programas de análisis de datos y modelado estadístico [35].
- **Matplotlib**: Se trata de una librería generadora de gráficos 2D, muy polivalente y con una gran cantidad de valores predeterminados [36].

Procesamiento de lenguaje natural

- **Spacy**: Se trata de una librería usada para el procesamiento avanzado de lenguaje natural, ya que se basa en las últimas investigaciones del ámbito y fue diseñada desde un primer momento para ser implementada en programas reales. Incluye modelos estadísticos pre-entrenados y actualmente soporta más de 49 idiomas. Es además de código abierto y publicado bajo la licencia del MIT (Massachusetts Institute of Technology) [37].
- **NLTK**: (Natural Language Toolkit) Al igual que Spacy, se trata de una librería para llevar a cabo procesos de PLN.
- **Gensim**: Nuevamente otra librería para procesos de PLN. El punto fuerte de esta librería es el modelado de temas, en otras palabras, incorpora funciones y algoritmo que son capaces de identificar de forma automática de que trata un conjunto de datos o documentos. Además, como es en el caso de este trabajo, es muy útil para construir representaciones de vectores [38].

Traducción

- **SpeechRecognition**: Librería creada por *Google* que brinda a sus usuarios, de manera gratuita, la posibilidad de convertir el sonido en texto. Puede usar como fuente tanto una pista de audio como un micrófono y además trabaja en varios idiomas.

Todas y cada una de las librerías, excepto las que vienen incluidas de forma predeterminada con Python, necesitan ser descargadas e instaladas[2]. Este proceso puede llevarse a cabo desde el terminal mediante un gestor de paquetes como puede ser el mencionado en el apartado 3.1 Entorno de trabajo '*Homebrew*'.

Una vez descargadas e instaladas, solo es necesario importarlas al archivo donde se está trabajando con el código, normalmente en la cabecera del mismo por comodidad, como podemos ver en la Figura 3-4.

```
import numpy as np

import matplotlib.pyplot as plt
import itertools
```

Figura 3-4 Ejemplo importación de librerías (Propio)

3.3 Aproximaciones llevadas a cabo

Para realizar el presente trabajo, se ha decido que la mejor forma de hacerlo es mediante dos aproximaciones o vías de desarrollo. En la primera, el programa trabaja de inicio a fin en el mismo idioma, es por ello que se necesita de un dataset para cada idioma a analizar. En la segunda aproximación, sin embargo, se buscará la manera de poder usar el programa en un entorno verdaderamente multilingüe para el cual no haga falta contar con un dataset para cada idioma.

En los siguientes apartados se explicarán ambos con detalle.

3.3.1 Aproximación dependiente del idioma

A continuación, tendrá lugar la explicación de la metodología seguida en el caso de realizar una aproximación considerando que este es dependiente del idioma. Por ello, se tratará de descomponer el trabajo en partes, describiendo los problemas encontrados y las soluciones adoptadas, así como el motivo y origen de ser de cada uno de ellos.

Como puede observarse en la Figura 3-5, se ha divido el problema en una secuencia o flujo que se tratarán de desglosar en diferentes apartados con el fin de facilitar la creación del programa y su explicación.

En esta primera aproximación, el programa diseñado trabaja siempre en el mismo idioma, de ahí la consideración de dependencia con él. Cabe resaltar que, en la fase de pruebas, descritas en el capítulo 4, se han usado a modo de ejemplo el inglés y el francés. Para explicar la secuencia de funcionamiento del programa, se ha adjuntado un esquema de la arquitectura del mismo la cual se acompañará de un desarrollo más detallado de cada proceso que tiene lugar.

De manera esquemática, la forma de trabajar del programa es la siguiente:

[2] Para encontrar tutoriales de instalación de paquetes puede acceder a la web https://pypi.org/.

1- En primer lugar, el programa capta el audio mediante 'Reconocimiento de voz' pronunciado en un idioma en concreto y fijo para todo el proceso, con el objetivo de determinar su relación con el ámbito náutico.
2- Posteriormente, haciendo uso de un 'Clasificador', el programa clasifica la frase captada, para determinar si su contenido es de interés o no. Dicho 'Clasificador' se ha entrenado previamente con un dataset en el idioma fijado anteriormente y etiquetado con el fin de aprender y ser capaz de llevar a cabo la clasificación. De esta manera, con este clasificador se hace un primer filtrado gracias al cual podemos diferenciar fácilmente si se trata de una oración relevante para nosotros.
3- El siguiente paso lo realiza el 'Extractor'. Para ello hace uso de un diccionario creado previamente de forma automático gracias a la utilización de fuentes de información como por ejemplo el uso de una enciclopedia online de términos náuticos. Los elementos de la oración dictada se convertirán a sus lemas para posteriormente ser comparados con el diccionario, pudiendo de esta forma mostrar en pantalla los términos coincidentes y por tanto náuticos.

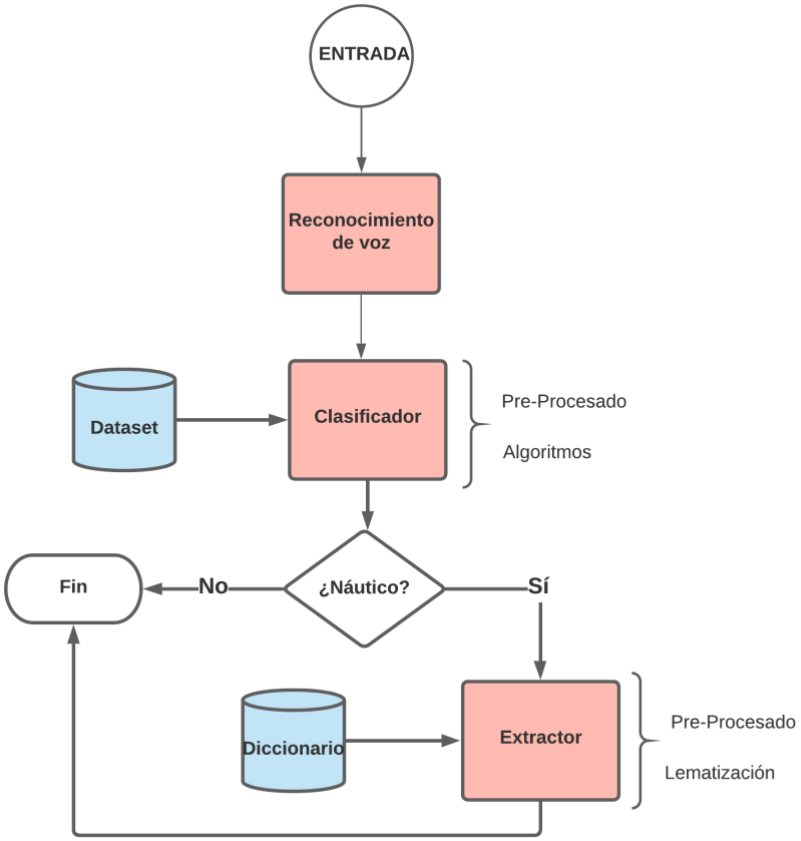


Figura 3-5 Esquema funcionamiento aproximación dependiente de idioma (Propia)

Por último, en la misma imagen pueden observarse unas llaves en ciertos procesos, lo que indica de manera esquemática los requisitos para poder realizar sus funciones. Tanto los procesos como estos requisitos mencionados serán comentados a continuación.

3.3.1.1 Reconocimiento de voz

Una vez que el dataset ya ha sido creado en el idioma definido, se puede empezar a trabajar con Python. El siguiente paso es el de implementar mediante código, una función que permita reconocer la voz para posteriormente transcribirla a texto, siendo la voz el mensaje que por radio nos llegaría

estando en el barco o en la estación radio. En este caso vamos a simular dicho mensaje de radio por una frase dictada al ordenador

Para llevar a cabo este proceso ha sido necesario, instalar la librería "*Speech Recognition*". Esta librería implementa una API [39], en este caso de *Google.* Una API no es más que un conjunto de protocolos que se utilizan para integrar software de terceros ya desarrollados, sin necesidad de ser programados de nuevo. Las API otorgan flexibilidad, simplifican el diseño y brindan oportunidades de diseñar herramientas y productos nuevos. Además, esta librería permite hacer uso del micrófono del ordenador al aginarle el nombre de "*sr.Microphone"* haciendo que sea posible llamarlo con código de manera sencilla. Además, tal y como puede ver en la Figura 3-6, aprovechando dicha posibilidad e implementado un *"switch"* en la que se enumeran todas las entradas de audio, el usuario es capaz de elegir cuál de ellas desea usar para llevar a cabo el reconocimiento de voz.

```
27  #ELEGIR MICROFONO#
28  def switch_mic_input():
29
30      argument = int(input("> Esperando respuesta... "))
31
32      switcher = {}
33      for i, mic_name in enumerate(sr.Microphone.list_microphone_names()):
34          switcher[i] = mic_name;
35
36      if argument not in switcher:
37          print(switcher.get(argument, "\nElección no válida"))
38          switch_mic_input(argument)
39
40      else:
41          print("\nElección válida", switcher.get(argument))
42          return argument
43
```

Figura 3-6 Función switch para elegir micrófono (Propia)

Una vez que el usuario ya ha elegido la entrada de audio que desea, el código referente al reconocimiento de voz se ejecuta automáticamente. El proceso es el que se explica a continuación.

Como puede verse en la Figura 3-7 durante dos segundos se escucha el sonido ambiente para ajustar los niveles de ruido y optimizar el reconocimiento, para posteriormente pasar a modo de escucha de voz durante un tiempo de quince segundos. Ambos temporizadores han sido elegidos en base a lo que se cree que son duraciones razonables y suficientes, teniendo en cuenta que el programa está diseñado para reconocer mensajes de radio de barcos, los cuales suelen ser concisos y breves. Por último, las últimas líneas de código de la imagen muestran en pantalla el audio transcrito y se almacena en la variable "*txt_speech*".

```
44
45  #RECONOCIMIENTO DE VOZ#
46  recognizer = sr.Recognizer()
47  for i, mic_name in enumerate(sr.Microphone.list_microphone_names()):
48      print(" #:", i, "Nombre", mic_name)
49
50  n_mic = switch_mic_input()
51
52  mic=sr.Microphone(device_index=n_mic)
53
54
55  with mic as source:
56
57      print("\n...Esuchando ruido ambiente...")
58      recognizer.adjust_for_ambient_noise(source, duration=2)
59      print("...Grabando durante 15 segundos...")
60      recorded_audio = recognizer.listen(source, timeout=15)
61      print("...Grabación finalizada...")
62
63  try:
64      print("...Reconociendo el audio...\n")
65      txt_speech = recognizer.recognize_google(
66              recorded_audio,
67              language="en-EN"
68          )
69      print(">>Texto Transcrito: {}\n".format(txt_speech))
70
71  except Exception as ex:
72      print(ex)
73
```

Figura 3-7 Reconocimiento de voz (Propia)

Además de los parámetros de tiempo de escucha, la función necesita saber qué idioma va a ser utilizado, en este caso se ha seleccionado el inglés. Más información sobre la API de Google pueden encontrarse en [40].

Es importante reseñar que esta API funciona de modo "*online*", es decir, necesita que el ordenador desde el cual se ejecuta el código esté conectado a Internet. Este hecho tiene asociado una serie de ventajas y desventajas. Por una parte, al ser una API de Google el rendimiento que se obtiene es adecuado pues se trata de un desarrollador fiable. Sin embargo, al ser una herramienta de uso online, su funcionalidad se pierde cuando se carece de conexión a la red.

3.3.1.2 Clasificador

Para la realización de este trabajo, la función de clasificación se realizará haciendo uso de Inteligencia Artificial, más concretamente mediante aprendizaje automático en su modalidad de aprendizaje supervisado.

Al tratarse de aprendizaje supervisado será necesario utilizar un dataset etiquetado previamente con el cual poder entrenar el sistema. Pero para ello es necesario llevar a cabo algunos procesos de transformación de los datos de entrada para que el ordenador sea capaz de interpretar el lenguaje. A esto se le conoce como '*preprocesado*' de texto. Por último, gracias a los algoritmos de aprendizaje, la máquina es capaz de llevar a cabo su función de clasificación.

3.3.1.2.1 Preprocesado

Una vez tenemos almacenada la frase que queremos clasificar, hace falta que el clasificador empiece a trabajar y es aquí es cuando entra a jugar el dataset creado anteriormente.

Gracias a la librería `pandas`, se puede acceder al fichero `.csv` que contiene las frases de las películas, así como sus etiquetas (náutico/no náutico), es decir, se está leyendo el dataset y más concretamente las columnas que contienen la información mencionada, tal y como se puede observar en la Figura 3-8.

```
#LEER DATASET#
df = pd.read_csv("/Users/sergio/Desktop/machine_learning_definitvo/archivos_csv/datasetnautico.csv", sep=';')
df = df.set_index("ID")


y = df.target
df.drop("target", axis=1)
df.drop("TIME", axis=1)

```

Figura 3-8 Función para leer el dataset (Propia)

Una vez tenemos acceso al fichero y se han seleccionado las columnas que son de interés o como en este caso, eliminado las que no nos sirven para almacenar exclusivamente las columnas que contienen las oraciones y sus etiquetas, podemos seguir con el proceso.

En Python son muchas las librerías que existen para llevar a cabo dicho preprocesado, y los pasos que se llevan a cabo son muy similares. Normalmente se parte del texto que se quiere procesar, y mediante una serie de funciones se divide dicho texto en fragmentos más pequeños, el texto se divide en párrafos, los párrafos en oraciones y las oraciones en palabras. Posteriormente se realiza una limpieza del texto, se eliminan signos de puntuación innecesarios, palabras de parada o "*stopwords*" como "*The*" en inglés, conjunciones y artículos entre otros, incluidas en un diccionario por defecto dentro de la misma librería. Por último, con las palabras que han resultado, se realiza un proceso de *"lematización*" que básicamente busca transformar las palabras en su raíz de tal forma que trate como una sola, todas las palabras que comparten raíz, simplificando notablemente la cantidad de información a tratar. Podría decirse que transforma todas las palabras y las convierte en su entrada en el diccionario.

Para este programa y, teniendo en cuenta que los algoritmos solo tendrán que analizar frases cortas, se ha llegado a la conclusión de que no es necesario realizar un preprocesamiento tan exhaustivo para obtener un buen rendimiento, e incluso podría llegar a ser contraproducente pues la eliminación de ciertos términos a priori innecesarios, así como tratar a las palabras que comparten raíz como una misma, puede tener un efecto negativo a la hora de analizar frases náuticas, ya que suelen incorporar términos muy específicos y una sola palabra puede cambiar el contexto de una oración por completo. Sin embargo, se ha aprovechado las funciones "`tfidfvectorizer`" y "`countvectorizer`", incluidas en la librería "`scikit-learn`", para implementar un pequeño preprocesamiento. Este preprocesamiento no lo realizan en sí las funciones mencionadas, pero si pueden añadirse parámetros para eliminar por ejemplo las palabras de espera, tal y como se muestra y explica en la Figura 3-9. Además, las funciones mencionadas anteriormente, cuentan con varios parámetros, en este caso "*stopwords*" y "*max_df*". El primero se encarga de eliminar las palabras de parada, como puede ser "and", y el segundo elimina los términos que aparecen con una frecuencia mayor de la señalada. Dichas funciones contienen muchos otros parámetros que pueden resultar útiles a la hora de llevar a cabo el preprocesamiento, todos ellos se pueden ver en su página oficial [41].

```
#VECTORIZACIÓN Y PREPROCESADO#
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)
tfidf_train = tfidf_vectorizer.fit_transform(X_train)
tfidf_test = tfidf_vectorizer.transform(X_test)

#COUNT VECTORIZACION#
count_vectorizer = CountVectorizer(stop_words='english')
count_train = count_vectorizer.fit_transform(X_train)
count_test = count_vectorizer.transform(X_test)
```

Figura 3-9 Funciones vectorización (Propia)

El uso de estas funciones nace de la necesidad de convertir el texto humano en un producto que pueda introducirse en los algoritmos de aprendizaje automático, en este caso ambas funciones transforman el texto en una matriz o vector de números con significado para los algoritmos. Aunque ambas funciones están diseñadas para lo mismo, trabajan de forma diferente. Mientras la función '*countvectorizer*' se limita a contar el número de veces que aparecen los términos o palabras dentro de una oración y crear una matriz con ellos, la función '*tfidfvectorizer*', del inglés *Term Frequency Inverse Document Frequency*, transforma las palabras en una matriz numérica con significado para los algoritmos pero en lugar de haciendo un recuento, usando cálculos matemáticos relativos a la frecuencia de aparición de los términos, más adecuado para este tipo de proyectos en qué términos con poca frecuencia de aparición en una misma oración aparecen en varios documentos diferentes. En la Figura 3-10, se pueden ver las diferencias entre las matrices que crean ambas funciones.

Count Vectorizer

	blue	bright	sky	sun
Doc1	1	0	1	0
Doc2	0	1	0	1

TD-IDF Vectorizer

	blue	bright	sky	sun
Doc1	0.707107	0.000000	0.707107	0.000000
Doc2	0.000000	0.707107	0.000000	0.707107

Figura 3-10 Matrices creadas con CountVectorizer y TfidfVectorizer (extraída de [35])

Una vez llevados a la práctica, se ha observado que la función '*tfidfvectorizer*' ofrece un desempeño más alto a los diferentes algoritmos de aprendizaje automático y por tanto ha sido la elegida para llevar a cabo el desarrollo del programa. El motivo por el cual la función TF-IDF obtiene un mejor rendimiento para la mayoría de los algoritmos es la función que utiliza para determinar la importancia de los términos ya que no solo tiene en cuenta la frecuencia de aparición de un término en un documento, sino que también tiene en cuenta el factor IDF de este, es decir, valora de manera adicional el número de veces que dicho término aparece en el cómputo global de los documentos. Dichos parámetros pueden verse en la Figura 3-11.

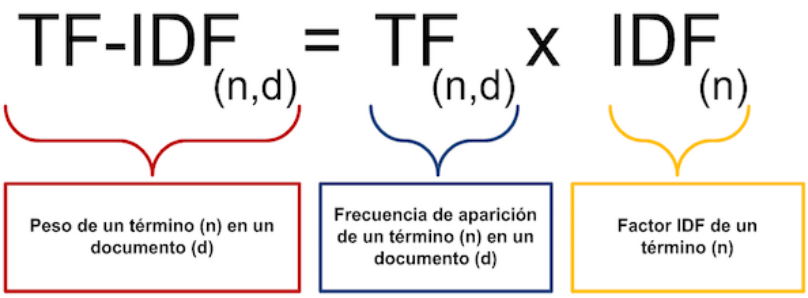


Figura 3-11 Fórmula de la función TFIDF (extraída de [37])

Por último, faltaría que es establecer los parámetros para que los algoritmos de aprendizaje automático aprendan y llevan a cabo su predicción. Para ellos es necesario dividir el conjunto de datos en dos grupos, el primero usado para aprender y el segundo para entrenar, la relación entre ellos suele

ser 80/20. En la imagen se puede observar que la división del texto para aprendizaje y entrenamiento se hace solamente en la columna '*TEXT*' del dataset y que contiene las frases.

```
93    #SEPARAR TRAIN/TEST#
94    X_train, X_test, y_train, y_test = train_test_split(df['TEXT'].astype(str), y, test_size=0.2, random_state=100)
95
```

Figura 3-12 Parámetros aprendizaje y entrenamiento (Propia)

Precisamente, sobre las variables X_train y X_test vamos a llevar a cabo el proceso de vectorización explicado anteriormente, ya que son las variables que guardan las oraciones para el entrenamiento y el test respectivamente tal y como puede verse en la Figura 3-12.

3.3.1.2.2 Algoritmos

Para llevar a cabo el último paso del proceso de clasificación, hace falta hacer uso de al menos un algoritmo de aprendizaje automático de entre todo los que existen. Como se ha explicado en el apartado 2.2 en este trabajo se ha hecho uso de 4 algoritmos diferentes, los cuales son conocidos por desempeñar gratamente a la hora de trabajar con lenguaje. A continuación, mostraré cómo ha sido su implementación mediante código, para en el apartado de resultados comentar cuáles han sido sus rendimientos.

Logistic Regression (LR)

```
def algoritmoLR(tfidf_train, y_train,tfidf_test,y_test,lista):
    model = LogisticRegression(solver='lbfgs')

    model.fit(tfidf_train, y_train)

    y_predict = model.predict(tfidf_test)

    print('Porcentaje de acierto')
    print(str(round(accuracy_score(y_test, y_predict)*100, 3))+'%')
    print(model.predict(lista))
```

Figura 3-13 Implementación del algoritmo LR (Propia)

Multinomial Naive Bayes (MNB)

```
def algoritmoMNB(count_train, y_train,count_test,y_test,lista):

    model = MultinomialNB()

    model.fit(count_train, y_train)
    y_predict = model.predict(count_test)

    print('Porcentaje de acierto')
    print(str(round(accuracy_score(y_test, y_predict)*100, 3))+'%')
    print(model.predict(lista))
```

Figura 3-14 Implementación del algoritmo MNB (Propia)

Passive Agressive Classifier (PAC)

```
def algoritmoPassiveAgressive(tfidf_train, y_train,tfidf_test,y_test,lista):
    linear_clf = PassiveAggressiveClassifier(max_iter=10000)

    linear_clf.fit(tfidf_train, y_train)
    pred = linear_clf.predict(tfidf_test)

    print('Porcentaje de acierto')
    print(str(round(accuracy_score(y_test, pred)*100, 3))+'%')
    print(linear_clf.predict(lista))
```

Figura 3-15 Implementación del algoritmo PAC (Propia)

Random Forest Classifier (RFC)

```
def algoritmoRandomForest(tfidf_train, y_train,tfidf_test,y_test,lista):

    rf = RandomForestClassifier()

    rf.fit(tfidf_train,y_train)
    y_pred = rf.predict(tfidf_test)

    print('Porcentaje de acierto')

    print(str(round(accuracy_score(y_test, y_pred)*100, 3))+'%')
    print(rf.predict(lista))
```

Figura 3-16 Implementación del algoritmo RFC (Propia)

3.3.1.3 Extractor

La función de este proceso es, una vez el texto ha pasado por el proceso de clasificación y este ha resultado favorable, obtener los términos en concreto que hacen que dicha frase sea náutica. Para ello se hará una comparación entre la frase etiquetada y los términos que forman parten de un diccionario náutico creado automáticamente. Por tanto, antes de comentar el modo de extracción, se hace necesario explicar el modo de crear dicho diccionario.

3.3.1.3.1 Creación del diccionario

Para crear el diccionario se valoraron dos caminos a seguir. El primero consistía en crear un archivo de texto y escribir manualmente términos considerados náuticos. A priori parece un camino bastante sencillo, y en efecto así es, pero, por otro lado, es un trabajo algo tedioso y existen limitaciones basadas en nuestro conocimiento de terminología náutica. Es por ello que finalmente se optó por una opción automática y escalable.

Este método consiste en, partiendo de una enciclopedia naval online, o en este caso de un glosario de términos náuticos [42], implementar un algoritmo capaz de interpretar dicho glosario mediante una serie de pasos y procesos y extraer de él los términos náuticos. Por tanto, el punto de partida ha sido el volcado de esa información procedente de Internet a un archivo `.csv` de fácil lectura. Este fichero contendrá dos columnas: la primera para las palabras y la segunda para la definición de las mismas, tal y como se observa en el ejemplo de la Figura 3-17.

	word	description
1	word	description
2	aback	A sail is aback when the wind fills it from the opposite side to the one normally used to move th
3	abaft	Toward the stern, relative to some object (e.g. abaft the cockpit).
4	abaft the beam	Farther aft than the beam: a relative bearing of greater than 90 degrees from the bow (e.g. two
5	abandon ship	An imperative to leave the vessel immediately, usually in the face of some imminent overwhelr
6	abeam	On the beam, a relative bearing at right angles to the ship's keel.
7	able seaman (AB), able-bodied seaman	A merchant seaman qualified to perform all routine duties, or a junior rank in some navies.
8	aboard	On or in a vessel. Synonymous with on board. See also close aboard.
9	about	To change the course of a ship by tacking. Ready about is the order to prepare for tacking.
10	above board	On or above the deck; in plain view; not hiding anything. Pirates would hide their crews below d
11	above-water hull	The hull section of a vessel above the waterline; the visible part of a ship. See also topsides.
12	absentee pennant	A special pennant flown to indicate the absence of a ship's commanding officer, admiral, his ch
13	absolute bearing	The bearing of an object in relation to north, either true bearing, using the geographical or true
14	accommodation ladder	A portable flight of steps down a ship's side.
15	accommodation ship, accommodation hulk	A ship or hulk used as housing, generally when there is a lack of quarters available ashore. An o
16	Act of Pardon or Act of Grace	A letter from a state or power authorising action by a privateer. See also letter of marque.
17	action stations	See battle stations.
18	admiral	A senior naval officer of flag rank. In ascending order of seniority: in the Royal Navy: rear admira
19	admiralty	1. A high naval authority in charge of a state's navy or a major territorial component. In the Roy
20	admiralty law	The body of law that deals with maritime cases. In the UK, it is administered by the Admiralty C

Figura 3-17 Glosario náutico en formato csv (Propia)

En el dataset de la Figura 3-17 puede verse que los términos pertenecientes a la columna "*Word"* son claramente náuticos, pero, además, en la columna "*description*" en donde se encuentran las definiciones de los términos también existen términos náuticos. Sabiendo esto, se pretende incluir dichos términos en el diccionario con el objetivo de obtener un léxico más completo

Por este motivo, se ha llevado a cabo un preprocesamiento algo más exhaustivo del texto, en el cual se han eliminado signos de puntuación, palabras entre corchetes, palabras que contienen números además de las stopwords. De manera adicional se ha llevado a cabo un proceso de lematización con el fin de simplificar aquellas palabras que pudieran tener la misma raíz léxica, como por ejemplo en el caso de "*ships*" que quedaría reflejada como "*ship*" (igual que las entradas de los diccionarios).

Teniendo en cuenta que las palabras que aportan más significado son aquellas que poseen como categoría léxica la de sustantivo o verbo, se ha decidido descartar el resto de categorías para trabajar exclusivamente con las dos anteriores. Esto ha sido posible gracias a la función "*word.pos"* incluída en la librería Spacy, tal y como puede verse en la Figura 3-18.

```
def lemmatizer(text,tag):
    sent = []
    doc = nlp(text)
    for word in doc:
        word.lemma_ = re.sub(re.escape(" él"), re.escape(""), word.lemma_)
        if word.lemma_ !="":
            #print(word.lemma_+"-"+word.pos_)
            if tag =="NOUN" and word.pos_ == "NOUN":
                sent.append(word.lemma_+"-"+"NOUN")
            if tag =="VERB" and word.pos_ == "VERB":
                sent.append(word.lemma_+"-"+"VERB")
    return " ".join(sent)
```

Figura 3-18 Función "lemmatizer" (Propia)

En este punto, disponemos de todos los sustantivos y verbos almacenados en los variables "`sentencesNoun`" y "`sentencesVerb`" que podemos ver en la Figura 3-19.

```
158  sentencesNoun0=[row.split() for row in df_cleanWord['word_lemmatize_noun_clean']]
159  sentencesNoun = [row.split() for row in df_clean['text_lemmatize_noun_clean']]
160  #print(len(sentences))
161  sentencesNoun=sentencesNoun0+sentencesNoun
162  #print(sentences[0])
163  #sentences=sentences.extend(sentences0)
164  print(len(sentencesNoun))
165  sentencesNoun_copy = sentencesNoun
166  sentencesNoun_copy = [val for sublist in sentencesNoun_copy for val in sublist]
167  #print(sentencesNoun_copy)
168  #print(sentences0)
169  #print(sentencesNoun0[0])
170  #print(sentencesNoun[0])
171
172  sentencesVerb0=[row.split() for row in df_cleanWord['word_lemmatize_verb_clean']]
173  sentencesVerb = [row.split() for row in df_clean['text_lemmatize_verb_clean']]
174  sentencesVerb=sentencesVerb0+sentencesVerb
175  print(len(sentencesVerb))
176  sentencesVerb_copy=sentencesVerb
177  sentencesVerb_copy = [val for sublist in sentencesVerb_copy for val in sublist]
178  #print(sentencesVerb0[0])
179  #print(sentencesVerb[0])
```

Figura 3-19 Almacenamiento en nombres y verbos (Propia)

Ambas variables tienen una longitud de 3453, es decir, cuentan con ese número de palabras tal y como puede verse en la Figura 3-20 de la salida del programa.

```
--------SPLIT-----------
3453
3453
```

Figura 3-20 Salida con la longitud de las variables (Propia)

Dichos términos serán usados para crear dos vocabularios, uno de verbos y otro de sustantivos, que posteriormente el algoritmo `Word2Vec` de la librería `gensim` utilizará para llevar a cabo su entrenamiento con el fin de ser capaz de identificar aquellas palabras que por relación semántica son similares entre ellas. El uso de este algoritmo se hace necesario ya que para el entrenamiento las palabras se expresan mediante una representación alternativa conocida con el nombre de *"Vectors Space Models"*. Éstos buscan representar las palabras como vectores en un espacio multidimensional con el propósito de que los términos relacionados entre sí se representen como puntos cercanos. Gracias a este proceso es posible obtener información semántica de las palabras ya que los términos pertenecientes a una misma categoría, como puede ser el caso de "coche", "autobús", "moto" se encontrarán en la misma zona del espacio multidimensional [43]. Como pueda apreciarse en la Figura 3-21, los parámetros quedarían definidos de la siguiente forma.

```
w2v_modelNoun = Word2Vec(min_count=5,
                         window=1,
                         size=100,
                         workers=4)

w2v_modelVerb = Word2Vec(min_count=5,
                         window=1,
                         size=100,
                         workers=4)

# this line of code to prepare the model vocabulary
w2v_modelNoun.build_vocab(sentencesNoun)
w2v_modelVerb.build_vocab(sentencesVerb)
# train word vectors
w2v_modelNoun.train(sentencesNoun, total_examples=w2v_modelNoun.corpus_count, epochs=w2v_modelNoun.epochs)
w2v_modelVerb.train(sentencesNoun, total_examples=w2v_modelVerb.corpus_count, epochs=w2v_modelVerb.epochs)
```

Figura 3-21 Código para la creación de los vocabularios y entrenamiento del algoritmo (Propia)

En parámetro "*min_count*" es de vital importancia a la hora de crear el vocabulario debido a que gracias a él se establece el número de apariciones que debe tener un término para que se incorpore al vocabulario de entrenamiento. En este caso se ha decidido que 5 será el valor para dicho parámetro. El parámetro "*size"* hace referencia al tamaño del vector y "*workers*" hace referencia al número de procesos paralelos que se desea realizar, que habitualmente se fija en un valor próximo al número de núcleo del procesador.

Una vez el algoritmo ha realizado el entrenamiento, además de poder conocer si un término está incluido en el vocabulario, podemos hacer uso de la función "*most_similar"* que, introduciendo como parámetro de búsqueda cualquier término, nos muestra en pantalla los "n-términos" más relacionados con el mismo. Es decir, está mostrando, los términos que tiene mayor relación semántica con, por ejemplo, "*ship*" o "*boat*". Se ha establecido además que solamente muestre los 20 términos más cercanos para que de esta forma nos aseguremos que no se pierde dicha relación.

Por último, solo faltaría volcar dichas coincidencias en un archivo de texto externo al programa para almacenar todos los términos que el programa ha considerado en base a los parámetros establecidos. De esta forma, ya se habrá creado un diccionario con los términos del glosario que tienen un mayor peso semántico referente al ámbito náutico.

```
with open('/Users/sergio/Desktop/machine_learning_definitvo/diccionario_automatico.txt', 'w') as f:
    #word_list=[]
    #word_list.append(' '.join([i for i in df_clean['text_lemmatize_noun_clean']]).split())
    #word_list.append(' '.join([i for i in df_cleanWord['word_lemmatize_noun_clean']]).split())
    #word_list = [val for sublist in word_list for val in sublist]
    #word_list = list(set(word_list))
    word_list_wout_dash = []
    for x in sentencesNoun_copy:
        #word_list_wout_dash.append(x.split('-')[0])
        word_list_wout_dash.append(w2v_modelNoun.wv.most_similar(positive=['ship-NOUN', 'boat-NOUN'], topn=20))
        word_list_wout_dash.append(w2v_modelNoun.wv.most_similar(positive=['navigation-NOUN'], topn=20))
        word_list_wout_dash.append(w2v_modelNoun.wv.most_similar(positive=['crew-NOUN'], topn=20))
        word_list_wout_dash.append(w2v_modelNoun.wv.most_similar(positive=['anchor-NOUN'], topn=20))
        word_list_wout_dash.append(w2v_modelNoun.wv.most_similar(positive=['port-NOUN'], topn=20))


    word_list_wout_dash= [val[0] for sublist in word_list_wout_dash for val in sublist]
    #print(word_list_wout_dash)
    word_list_wout_dash = list(set(word_list_wout_dash))
    word_list_wout_dash2 = []
    for word in word_list_wout_dash:
        word_list_wout_dash2.append(word.split('-')[0])
    f.write('\n'.join(word_list_wout_dash2))
```

Figura 3-22 Almacenamiento de términos coincidentes según parámetros en archivo (Propia)

En la Figura 3-22 puede verse el proceso llevado a cabo para almacenar los términos, así como los parámetros de búsqueda usados, en este caso *"ship", "boat", "navigation", "crew", "anchor",* y *"port"*. En la imagen solo se muestra para el caso de los sustantivos ya que para los verbos el proceso ha sido exactamente el mismo.

De manera adicional, en el Anexo III: Extensión del diccionario léxico, se ha adjunto una imagen del diccionario obtenido con el programa y en el Anexo IV: Clustering de términos náuticos, el clustering de los datos.

3.3.1.3.2 Comparación de términos coincidentes

Para que el proceso tenga éxito, hace falta llevar a cabo un preprocesado, similar al explicado en el apartado 3.3.1.2.1, con la salvedad de que además es necesario realizar un lematizado de los términos de la oración con el fin de que coincidan con los términos que conforman el diccionario. Esto es de capital importancia ya que en el diccionario aparecen los lemas de los términos y, por tanto, el programa al comparar la oración dictada con los términos del diccionario, necesita que en estos sean exactamente iguales.

Una vez disponemos del diccionario con términos náuticos, ya solo queda comparar dichos términos con la frase escuchada por el programa para, además si se trata de un diálogo náutico o no, saber exactamente qué términos hacen que dicha oración lo sea.

Para obtener unos resultados fiables, teniendo en cuenta que dicho diccionario contiene únicamente el lema de las palabras, se ha estimado necesario realizar un proceso análogo al anterior en lo que a lematización se refiere de tal forma que el programa sea capaz de comparar únicamente lemas entre sí.

Si ejecutamos el programa en modo prueba poniendo como frase a analizar "*I like ships*", claramente el término que convierte la oración en náutica es "*ships*". Sin embargo, nuestro diccionario solamente contiene el singular de la misma, es decir, su lema, y es por ello que, a pesar de etiquetarla como náutica no extrae ningún término náutico. Es por ello que el proceso de lematizar la frase anterior es fundamental para la correcta extracción de términos.

```
def econtrar_terminos_nauticos():
    #Código para encontar las palabras
    nlp = spacy.load('en')
    doc = nlp(l[0]) #TODO CAMBIAR EN EL FUTURO POR TRANSLATED
    for token in doc:
        print(token, token.lemma, token.lemma_)

    with open('diccionario_automatico.txt') as f:
        content = f.readlines()

        naval_terms = [x.strip() for x in content]

    # Import the PhraseMatcher and initialize it
    matcher = PhraseMatcher(nlp.vocab, attr='LEMMA')

    # Create pattern Doc objects and add them to the matcher
    # This is the faster version of: [nlp(country) for country in COUNTRIES]
    patterns = list(nlp.pipe(naval_terms))
    matcher.add('naval_terms', None, *patterns)

    # Call the matcher on the test document and print the result
    print('Términos Náuticos encontrados:')
    matches = matcher(doc)
    for match_id, start, end in matches:
        span = doc[start:end]
        print(span.text)
```

Figura 3-23 Código para encontrar términos coincidentes (Propia)

En la Figura 3-23 se muestra el código que permite encontrar los términos coincidentes en el cual se puede apreciar el `attr='LEMMA'` que establece el patrón de búsqueda sea los lemas y no la palabra completa.

3.3.2 Aproximación Independiente de idioma

Una vez realizada la primera versión del programa llevada a cabo mediante la aproximación explicada en el apartado anterior, y sabiendo que para el funcionamiento del mismo es necesario disponer de un dataset etiquetado en el idioma específico en el que se desea trabajar, nos surge la siguiente pregunta:

¿Qué ocurre si queremos clasificar una frase en un idioma para el cual no disponemos de un dataset?

Es aquí entonces cuando se plantea una posible solución, que no es otra que añadir un proceso de traducción a continuación del reconocimiento de voz con el fin de traducir el mensaje al idioma de trabajo del programa. Haciendo este procedimiento, solamente es necesario disponer de un sistema traductor del idioma de partida al idioma final, y de un único dataset en el mismo idioma final, en este caso inglés.

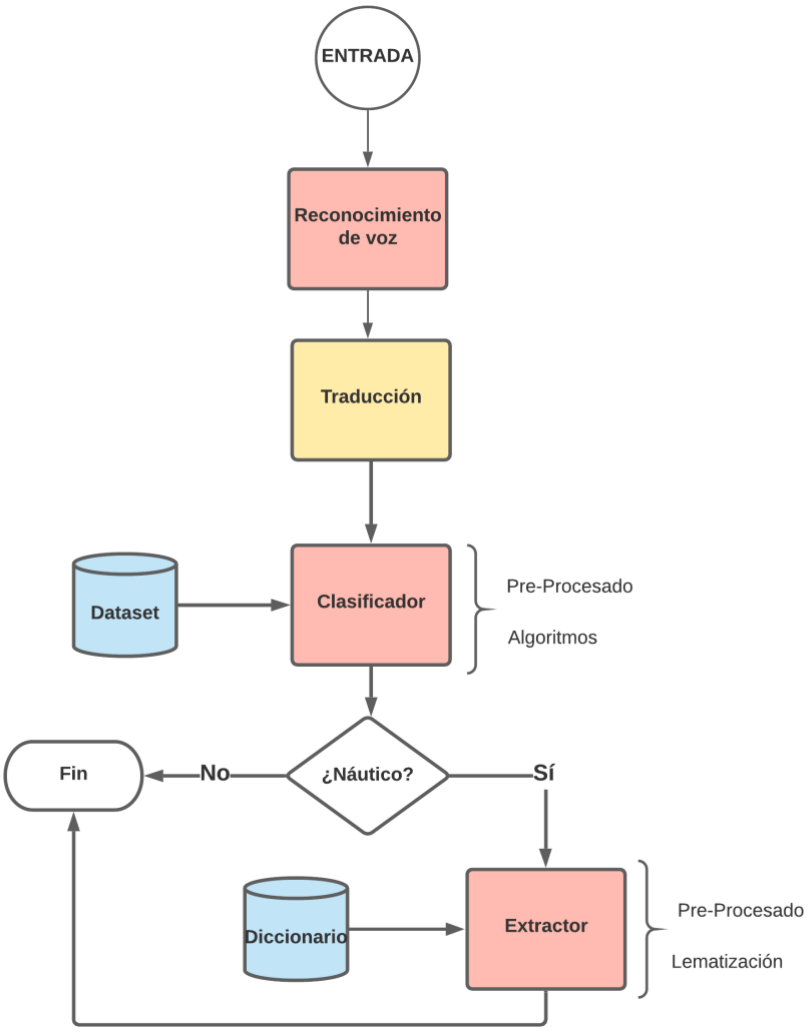


Figura 3-24 Esquema funcionamiento aproximación independiente de idioma (Propia)

Más adelante, en el apartado de pruebas y resultados, se mostrarán más en detalle las dificultades que puede entrañar el proceso de crear un dataset válido para idiomas los cuales no se conocen ni se disponen da datos suficientes.

Debido a que la segunda aproximación se compone de los mismos elementos que la primera aproximación en lo que al funcionamiento se refiere, con la excepción del módulo de traducción, solo se explicará en que consiste dicho módulo.

Traducción

Para llevar a cabo este proceso, se ha decidido usar la API de Google "*google_trans_new*". Al tratarse de una API, su uso simplifica enormemente la implementación de esta funcionalidad. Además, al tratarse de un desarrollador tan potente y fiable como lo es Google, se puede garantizar un buen comportamiento en la traducción de textos.

```
##traduccion a ingles
translator = google_translator()
translated = translator.translate(txt_speech, lang_tgt='en')
print(">>Texto en ingles: {}\n".format(translated))


translator2 = google_translator()
translated2 = translator2.translate(txt_speech, lang_tgt='es')
print(">>Texto en español: {}\n".format(translated2))
```

Figura 3-25 Código para la traducción (Propia)

Además, como puede verse en la Figura 3-25 se ha decido traducir la frase dictada, guardada con la variable "*txt_speech*", tal y como se ha explicado en el apartado 3.3.1.1, a dos idiomas. El primero es el inglés y sobre el cual el programa va a trabajar pues el dataset de entrenamiento fue creado en dicho idioma. De manera adicional muestra en pantalla la frase traducida al español, de manera informativa, ya que es nuestro idioma nativo.

4 RESULTADOS EXPERIMENTALES

4.1 El dataset de prueba

En este primer apartado se comentarán el modo de obtención de los datos, así como un análisis visual de los datos para llevar a cabo las pruebas. Pero antes de entrar a comentar ambos aspectos, se hace necesario explicar las fuentes de las que se obtuvieron esos datos.

4.1.1 Plataformas de películas

IMDB

Este término hace referencia, en inglés, a *Internet Movie Database.* Se trata de una plataforma web que contiene información de todo lo que a películas y series se refiere. Es una de las páginas más populares y usadas a nivel mundial para buscar información sobre el mundo cinematográfico [44].

Figura 4-1 Logotipo de la web IMDB (extraída de [45])

Según datos de la misma web [45], esta recibe más de 200 millones de visitantes al mes y su base de datos contiene información relacionada con más de 3 millones de películas. Para el caso en concreto de este TFG, se hizo uso de esta web con el fin de encontrar películas relacionadas con el ámbito naval. En la Figura 4-2 puede verse el primer resultado para la búsqueda de películas con la palabra clave “naval”.

Figura 4-2 Resultado en IMDB para películas navales según su popularidad (extraída de [45])

SUBSCENE

Por otro lado, *subscene* es una web relacionada también con el mundo cinematográfico pero su finalidad es completamente diferente a la anterior. Se trata de una web que cuenta con uno de las mayores y mejores bases de datos de subtítulos de películas. Cuenta con soporte para gran cantidad de idiomas y sus servidores almacenan los diálogos de prácticamente todas las filmografías. En la Figura 4-3 puede apreciarse una captura de la página principal.

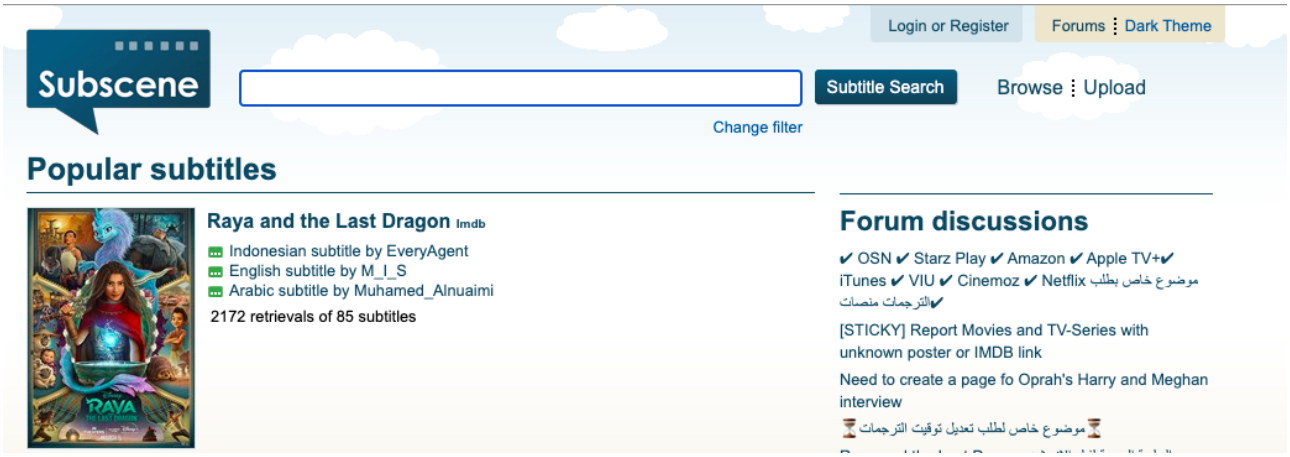


Figura 4-3 Página principal de la web Subscene (extraída de [46])

Como se ha comentado anteriormente, la web cuenta con un gran número de idiomas de subtítulos disponibles. De hecho, en la misma página se puede hacer un filtrado precisamente por idioma. En la Figura 4-4 Figura 4-4 Idiomas más populares (izquierda) e idiomas alternativos (derecha) (extraída de [46])pueden apreciarse dos columnas, la de la izquierda que contiene los idiomas más populares de la plataforma, y en la derecha otros lenguajes quizás no tan habituales con los que trabaja.

Figura 4-4 Idiomas más populares (izquierda) e idiomas alternativos (derecha) (extraída de [46])

4.1.2 Mecanismo de elaboración del dataset de pruebas

Los procesos de aprendizaje automático o *machine learning* necesitan de un conjunto de datos para poder desarrollar su función con éxito. Un dataset puede definirse por tanto como un conjunto de datos estructurados, los cuales pueden relacionarse entre sí. Es por ello que, para poder hacer pruebas con el sistema desarrollado en este TFG, se hace necesario la utilización de datos útiles para lograr este fin. Se es consciente que este dataset se podía haber creado de más modos, pero se ha elegido este por simplicidad. De esta forma, la creación de este dataset servirá, por un lado, para entrenar el sistema clasificador y, por el otro, para ser utilizado a modo de prueba para la extracción de la terminología náutica.

En una primera instancia, se realizaron varias búsquedas en Internet con el fin de encontrar datasets de uso libre y que pudieran servir como base para el programa. Tras la exhaustiva búsqueda, no se consiguió encontrar ninguno que cumpliese con las características necesarias, es decir, que contuviera frases relacionadas con el ámbito náutico, y/o estuviera disponible en varios idiomas. Fue por ello que se decidió optar por la creación de uno propio. Acudiendo a la web "*IMDb*" [45], utilizando el filtro de búsqueda por género, se listaron las películas navales en orden decreciente de puntuación para posteriormente hacer uso de "*Subscene*" [46] y descargar el archivo de subtítulos de las mismas en inglés y francés.

De manera adicional y aprovechando que los subtítulos de películas en diferentes idiomas pueden tratarse como textos paralelos (véase la Figura 4-5), es decir, para una misma película los diálogos en ingles deben corresponderse con los diálogos en otro idioma mediante el instante de tiempo en que aparecen, el etiquetado de los diálogos en una lengua concreta (como por ejemplo el inglés) puedan extrapolarse a otro idioma sobre el cual no tenemos la misma habilidad. Para este trabajo se han creado mediante texto paralelos dos datasets totalmente idénticos, uno en inglés y otro en francés, pero, ¿por qué es necesario tener datasets en distintos idiomas? Esta pregunta es fácil de responder y es que, si no disponemos de un módulo de traducción como en la segunda aproximación, 3.3.2, el programa de

aprendizaje automático no es capaz de clasificar frases en un idioma diferente al que ha sido entrenado. En otras palabras, el idioma del dataset y de la frase que se quiere etiquetar debe ser el mismo.

TEXT	target	TEXT
We are here to sharpen our skills as a team.	no nautico	Nous sommes là pour affûter nos compétences en équipe.
I'm excited to see what we learn.	no nautico	Je suis ravi de voir ce que nous apprenons.
Be safe out there and look out for each other and keep	no nautico	Soyez en sécurité, faites attention les uns aux autres et continuez à charger.
I will personally ...	no nautico	Je vais personnellement ...
Yo Saunders, ever been in a department run by some k	no nautico	Yo Saunders, a déjà été dans un département dirigé par une sorte de combo mutant Donald Trump / Mike Tyson.
I swear you said Donald Trump. Wanna calrify?	no nautico	Je jure que vous avez dit Donald Trump. Voulez-vous calmer?
I think I heard a Myke Tyson as well.	no nautico	Je pense que j'ai aussi entendu un Myke Tyson.
If you did,	no nautico	Si vous l'avez fait,
it was only in reference to the fact the you both projec	no nautico	c'était seulement en référence au fait que vous projetiez tous les deux une grande intensité physique, monsieur.
That's flattering.	no nautico	C'est flatteur.
Hopper! A helo headed for the Samson. Make sure you	no nautico	Hopper! Un hélicoptère se dirigea vers le Samson. Assurez-vous que votre cul est dessus.
- Why? - I don't know why. Just make sure your on it!	no nautico	- Pourquoi? - Je ne sais pas pourquoi. Assurez-vous simplement que vous y êtes!
Damn!	no nautico	Mince!
- You've gotta make some calls. - Some calls?	no nautico	Tu dois passer des appels. - des appels?
Who do I call to teach you humility?	no nautico	Qui dois-je appeler pour vous apprendre l'humilité?
I'm sorry, I dont' have that number.	no nautico	Je suis désolé, je n'ai pas ce numéro.
I just don't get it man. You got so much potential.	no nautico	Je ne comprends juste pas mec. Vous avez tellement de potentiel.
I'm sorry you have to deal with this.	no nautico	Je suis désolé que vous ayez à gérer cela.
And I'm sorry I let you down.	no nautico	Et je suis désolé de vous avoir laissé tomber.
Stand up nice and tall. I want you to focus on that gree	no nautico	Tenez-vous bien debout. Je veux que vous vous concentriez sur ce marqueur vert juste au milieu de votre dos.
Where do you think I'm paying attention to?	no nautico	Où pensez-vous que je fais attention?
- Keep your feet together. - My feet can't get any closer	no nautico	Gardez vos pieds ensemble. - Mes pieds ne peuvent pas se rapprocher.

Figura 4-5 Muestra de textos paralelos de ambos datasets (Propia)

El siguiente paso, es convertir y editar los archivos contenedores de los subtítulos de su extensión "*.srt*" a la extensión válida para el dataset y legible por el programa, es decir, "*.csv*". Este tipo de archivo es además manipulable con "*Excel*", lo que hace que la tarea de edición sea notablemente más sencilla. Los dataset suelen estar estructurados en filas y columnas, y es por ello que en este caso se ha usado el símbolo punto y coma ";" como delimitador de los mismos, de forma que Excel lo use como separador. A continuación, mediante dos imágenes mostraré como muestra un editor de texto plano como '*TextEdit*" un archivo .csv y como lo trata Excel para mostrarlo con su correspondiente estructura. Para llevar a cabo la transformación de archivos en formato "*.srt" descargado de subscene* a "*.csv"*, se ha usado la plataforma [47].

```
ID;TIME;TEXT;target;;;
1;00:01:10:14;Today really marks the first stage of an unprecedented technological advancement.;no ;;;
2;00:01:15:21;We have begun to identify a so-called goldilocks planet,;no ;;;
3;00:01:20:07;these planets that share a similar relationship with their sun as we have with ours;no ;;;
```

Figura 4-6 Archivo csv abierto con TextEdit (Propia)

Como puede verse en la Figura 4-6 en la parte superior aparecen escritos los nombres de las diferentes columnas, así como el separador de columnas ";" y de fila ";;;". La columna "ID" hace referencia a la numeración de la fila, la comuna "TIME" se refiere al momento en que la oración aparece en la película, la columna "TEXT" contiene las oraciones y la columna "target" la etiqueta que discrimina entre frases con terminología náutica o no.

Gracias a estos parámetros, Excel es capaz de entender la estructura del archivo, y con Python podemos acceder a las columnas en concreto que nos interesen, "TEXT" y "target" en este caso.

	A	B	C	D
1	ID	TIME	TEXT	target
2	201	00:23:44:19	We are here to sharpen our skills as a team.	no nautico
3	202	00:24:07:24	I'm excited to see what we learn.	no nautico
4	203	00:24:13:04	Be safe out there and look out for each other and keep charging.	no nautico
5	204	00:24:22:12	I will personally ...	no nautico
6	205	00:24:25:06	Yo Saunders, ever been in a department run by some kind of Dona	no nautico
7	206	00:24:33:11	I swear you said Donald Trump. Wanna calrify?	no nautico
8	207	00:24:38:14	I think I heard a Myke Tyson as well.	no nautico
9	208	00:24:40:19	If you did,	no nautico
10	209	00:24:41:12	it was only in reference to the fact the you both project great phy	no nautico
11	210	00:24:47:08	That's flattering.	no nautico
12	211	00:24:48:22	Hopper! A helo headed for the Samson. Make sure your ass is on i	no nautico
13	212	00:24:52:14	- Why? - I don't know why. Just make sure your on it!	no nautico
14	213	00:24:58:15	Damn!	no nautico
15	214	00:25:41:10	- You've gotta make some calls. - Some calls?	no nautico
16	215	00:25:44:01	Who do I call to teach you humility?	no nautico
17	216	00:25:48:06	I'm sorry, I dont' have that number.	no nautico
18	217	00:25:53:19	I just don't get it man. You got so much potential.	no nautico
19	218	00:26:00:21	I'm sorry you have to deal with this.	no nautico
20	219	00:26:08:01	And I'm sorry I let you down.	no nautico
21	220	00:26:58:07	Stand up nice and tall. I want you to focus on that green marker ri	no nautico
22	221	00:27:01:21	Where do you think I'm paying attention to?	no nautico
23	222	00:27:03:13	- Keep your feet together. - My feet can't get any closer.	no nautico
24	223	00:27:06:08	- Stand up tall! - Alright, I've had enough of this.	no nautico
25	224	00:27:08:15	- Keep going. Just a little more. - Turn the damn thing off!	no nautico
26	225	00:27:11:16	- Come on, we're just about there. - Turn it off now!	no nautico
27	226	00:27:15:03	I'm your new physical therapist.	no nautico
28	227	00:27:21:14	- I'm sensing alot of anger. - Very persceptive of you.	no nautico
29	228	00:27:26:20	- Is there anything besides anger in there, Mick? - Not much.	no nautico
30	229	00:27:35:17	Your last therapist says that you lost the will to fight. Is that accu	no nautico
31	230	00:27:42:05	I lost my fight when I lost my legs.	no nautico
32	231	00:27:50:22	You realise you're still the same man that won the "Golden Glove	no nautico
33	232	00:27:54:08	at 22, Bronze Star in Afganistan?	no nautico
34	233	00:27:58:02	Same man.	no nautico
35	234	00:28:07:09	That's all I've ever known.	no nautico
36	235	00:28:13:07	- Alright. Let's go! We're gonna take a walk. - No we're not!	no nautico
37	236	00:28:19:11	Legs on!	no nautico

Figura 4-7 Archivo csv del dataset en inglés abierto con Excel (Propia)

A diferencia de la imagen anterior, en la Figura 4-7 pueden observarse las mismas tres primeras filas, pero esta vez separadas por sus correspondientes columnas. Finalmente, el trabajo de etiquetado sobre cada una de las frases se ha realizado de forma manual.

4.1.3 El dataset resultado

Ambos datasets usados para las pruebas son idénticos en cuanto a cantidad y este puede ser considerado como balanceado. El dataset importado finalmente cuenta con un total de 3366 oraciones de las cuales 1941 han sido etiquetadas como “no náuticas” y 1425 como “náuticas”, en porcentajes sería un 57% frente a un 43%.

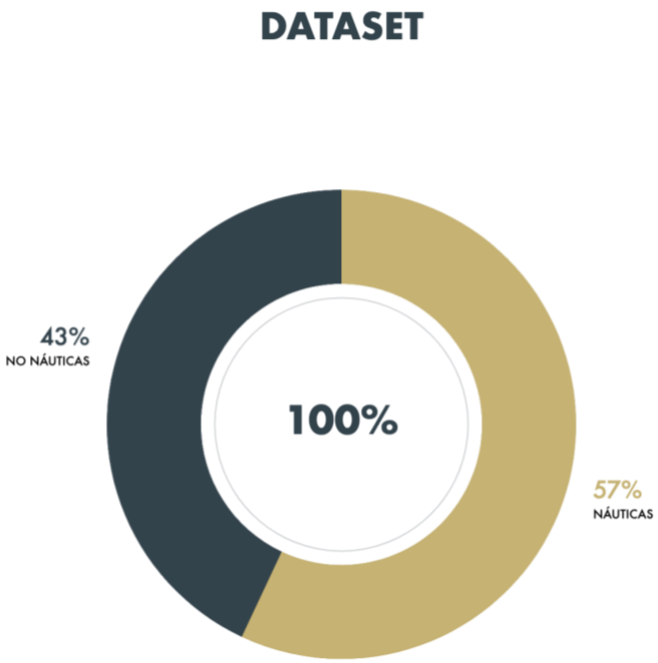


Figura 4-8 Porcentajes de etiquetas dataset (Propia)

4.2 Resultados

4.2.1 Métricas de evaluación de la clasificación

Para evaluar los resultados se van a aplicar 4 métricas disponibles en la librería *sklearn*. Estas métricas se calculan a partir de los valores obtenidos en una matriz de confusión generada por cada algoritmo. En una matriz de confusión se muestran en la diagonal principal, los aciertos que ha tenido el algoritmo de aprendizaje automático, es decir, los positivos etiquetados como positivos y los negativos etiquetados como negativos, mientras que en la otra diagonal se muestras los falsos positivos y negativos. Para facilitar la explicación hare uso de la siguiente tabla en donde: TN (True Negative), TP (True Positive), FP (False Positive), FN (False Negative).

		predicción	
		0	1
realidad	0	TN	FP
	1	FN	TP

Figura 4-9 Matriz de confusión (extraída de [48])

Gracias al uso de esta matriz, se puede valorar que algoritmo funciona mejor, en el caso de este proyecto se utilizarán las siguientes métricas [48].

Precision (Precisión)

Con esta métrica se puede obtener la calidad del modelo de aprendizaje automático en tareas de clasificación, se utiliza la siguiente fórmula:

$$Precisión = \frac{TP}{TP + FP} \tag{4.1}$$

Recall (Exhaustividad)

Esta métrica no informa de la cantidad de positivos que el algoritmo es capaz de identificar, ya que tiene en cuenta la cantidad de falsos negativos, usa la siguiente fórmula:

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

F Score (Valor F)

Esta métrica se usa con el fin de combinar las dos anteriores en una sola, hace que se más sencillo comparar el rendimiento combinado de precisión y exhaustividad. Hay varios valores de F, pero para este proyecto se ha calculado el F1 en el que se tanto la precisión como el recall tienen la misma importancia y es por ello que se calcula la media armónica. La fórmula es la siguiente:

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{4.3}$$

Accuracy (Exactitud)

La exactitud se encarga de medir el número de casos, o mejor dicho el porcentaje de aciertos del programa, es una de las métricas más usadas. Ésta es poco eficiente en el caso de tener datos no balanceados, sin embargo, el dataset usado para este programa se considera que sí lo está. Usa la siguiente fórmula:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.4}$$

De forma adicional, en los casos en los que el conjunto de datos tiene más de 2 etiquetas, es conveniente medir cada una de las métricas descritas anteriormente para cada una de las clases y posteriormente hacer una media. Este proceso se lleva a cabo mediante el parámetro '*average'* pero no es necesario para este proyecto al contar con 2 clases.

4.2.2 Valoración de la evaluación de la clasificación

Para llevar a cabo la valoración de los resultados, se compararán las métricas seleccionadas para cada uno de los algoritmos tanto para el dataset en inglés como para el texto paralelo en francés.

4.2.2.1 Accuracy (Exactitud)

A continuación, se mostrará un gráfico, Figura 4-10, en donde se compara el parámetro accuracy (exactitud) de los diferentes algoritmos. Las barras de color verde representan el inglés y las de color azul el francés.

En la leyenda de dichas barras se puede observar el nombre de cada algoritmo en donde LR es para *Logistic Regression*, MNB para *Multinomial Naive Bayes*, PAC para *Passive Agressive Classifier* y RFC para *Random Forest Classifier*.

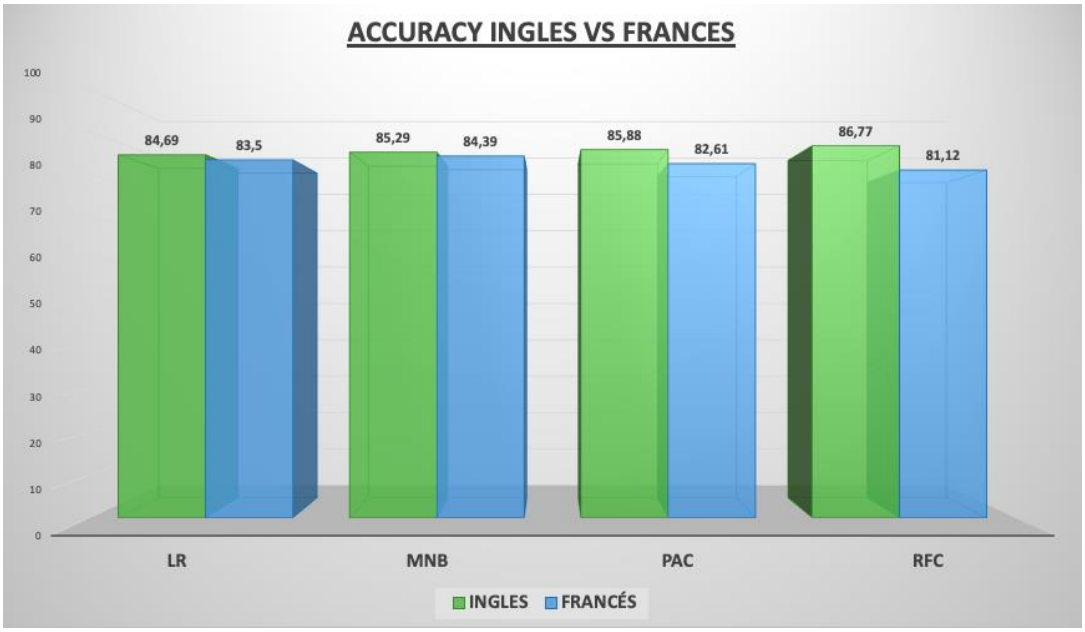


Figura 4-10 Comparación parámetro Accuracy para cada idioma (Propia)

Gracias a la gráfica, pueden sacarse dos conclusiones fácilmente. La primera es que el programa de aprendizaje automático obtiene mejores resultados para el inglés que para el francés (con exactamente los mismos parámetros de entrenamiento), aunque ambos resultados son claramente próximos; y la segunda es que de los cuatro algoritmos que se han probado, el que mejor puntuación ha obtenido ha sido el Random Forest Classifier para el inglés con un porcentaje del 86,77% y el Multinomial Naive Bayes para el francés con un porcentaje del 84,39%. A pesar de ello sería interesante reseñar que el resto de algoritmos han hecho un gran trabajo en lo que a clasificación se refiere para ambos idiomas.

4.2.2.2 Precisión, Recall y F1

Para facilitar la visualización de los resultados de estos parámetros se ha decidido plasmarlos en la siguiente tabla, en la que, de manera análoga a la anterior, se compararán entre sí los rendimientos para los dos idiomas.

	INGLÉS			FRANCÉS		
	PRECISION	**RECALL**	**F1**	**PRECISION**	**RECALL**	**F1**
LR	89,15	70,26	**78,58**	81,6	75,83	78,61
MNB	81,48	81,78	81,63	81,78	78,43	**80,07**
PAC	82,22	82,52	82,37	75,85	82,89	79,21
RFC	86,88	78,81	**82,65**	74,82	79,55	**77,11**
LR: Logistic Regression **MNB**: Multinomial Naive Bayes **PAC**: Passive Aggressive Classifier **RFC**: Random Forest Classifier						

Tabla 4-1Comparación parámetros secundarios (Propia)

En la Tabla 4-1, los números representan porcentajes de acierto y además para cada idioma se ha resaltado en verde y rojo los algoritmos con mejor y peor rendimiento respectivamente. Se ha seleccionado el medidor F1 como el óptimo de entre los 3 anteriores ya que tiene en cuenta la precisión y el recall. De la misma manera que se pudo ver para el caso del parámetro Accuracy, para el inglés los resultados vuelven a ser superiores que para el francés y además también se cumple para el idioma inglés que el algoritmo Random Forest Classifier es que obtuvo más alto porcentaje. Sin embargo, para el caso del francés, el Random Forest Classifier obtuvo el peor resultado mientras que el Multinomial Naive Bayes el más alto.

4.2.2.3 Matrices de confusión

Por último, para finalizar el estudio de los resultados, se van a mostrar las diferentes matrices de confusión obtenidas. En total han sido ocho (8), una para cada algoritmo y para cada idioma (inglés y francés). Como habíamos explicado anteriormente, en la matriz de confusión se pueden ver el total de frases que han sido etiquetadas correctamente y cuales han sido etiquetadas como falsos positivos o negativos. En forma de recordatorio, del total de los 3366 diálogos disponibles en los datasets, un 80% ha sido usado para entrenar y el 20% para realizar el test, es por ello que el sumatorio de los números de las matrices resulta 672.

Para el caso de este trabajo, el eje horizontal corresponde a las predicciones y el eje vertical corresponde a los valores reales. A continuación, se adjuntan todas las matrices de confusión.

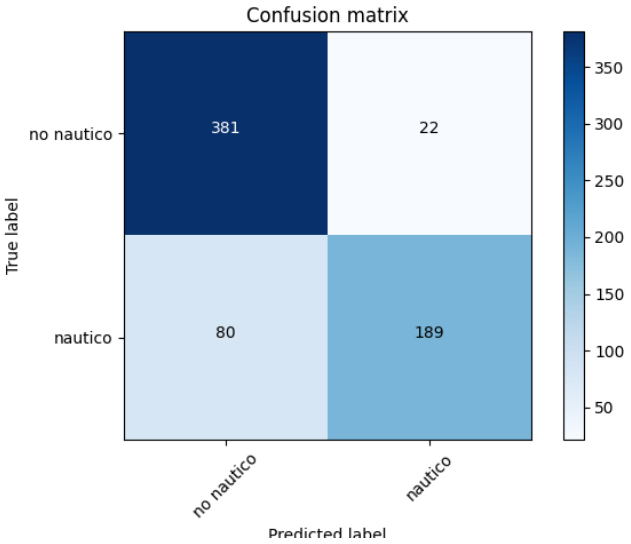


Figura 4-11 Matriz confusión Logistic Regression idioma inglés (Propia)

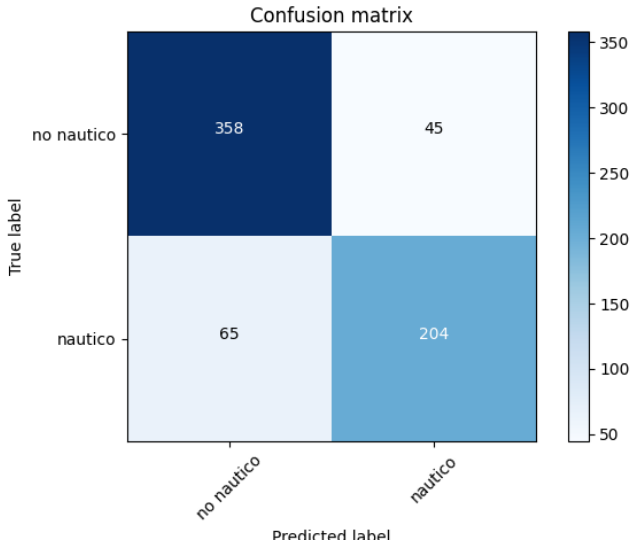


Figura 4-12 Matriz de confusión Logistic Regression idioma francés (Propia).

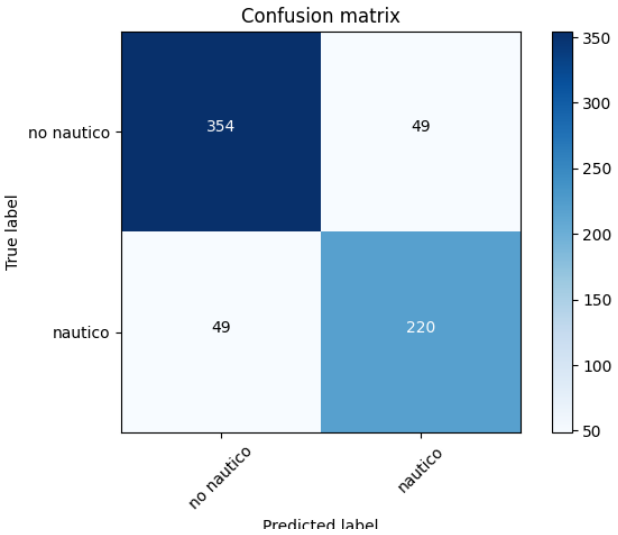


Figura 4-13 Matriz de confusión Multinomial Naive Bayes idioma inglés (Propia)

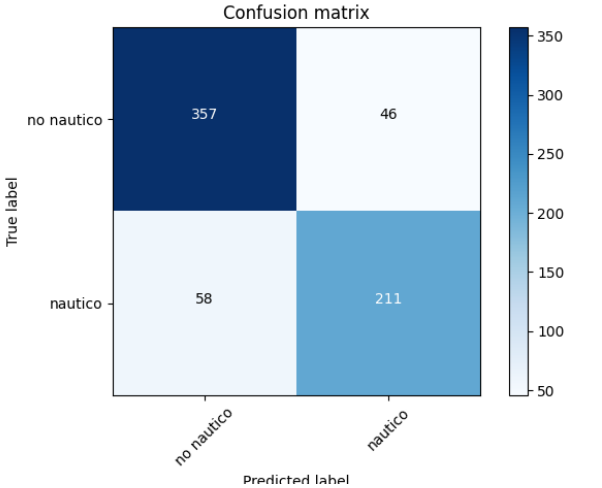


Figura 4-14 Matriz de confusión Multinomial Naive Bayes idioma francés (Propia)

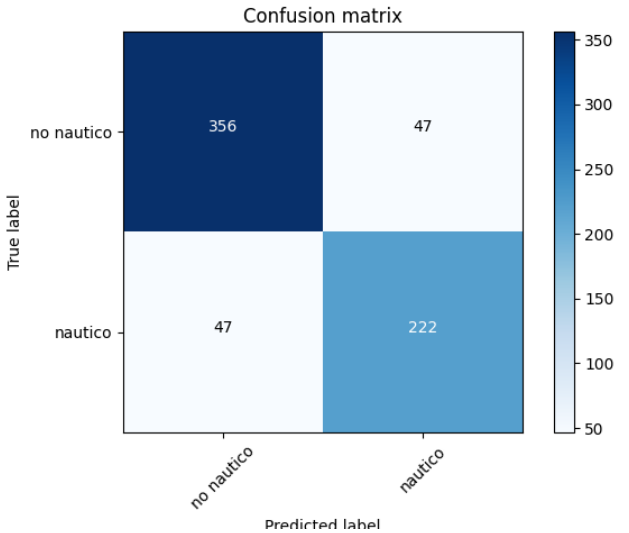


Figura 4-15 Matriz de confusión Passive Agressive Classifier Idioma Inglés (Propia)

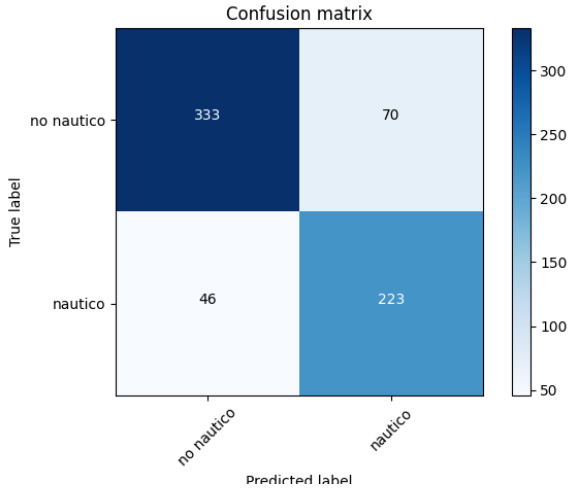


Figura 4-16 Matriz de confusión Multinomial Naive Bayes Idioma Francés (Propia)

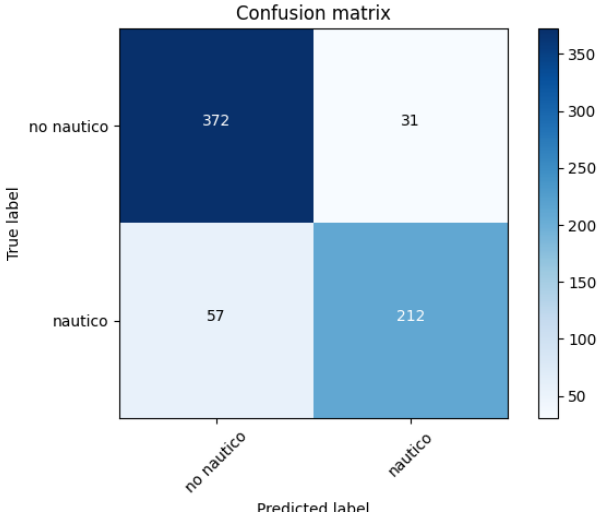


Figura 4-17 Matriz de confusión Random Forest Classifier Idioma Inglés (Propia)

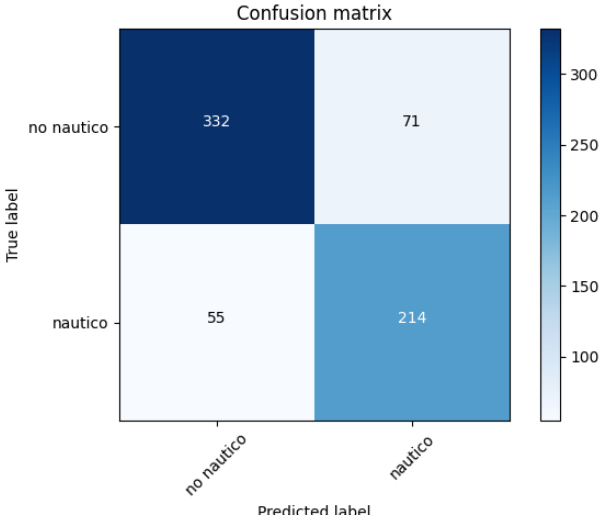


Figura 4-18 Matriz de confusión Random Forest Classifier Idioma Francés (Propia)

4.2.3 Evaluación de la extracción

A diferencia de los algoritmos de *Machine Learning* que han podido valorarse numéricamente haciendo uso de las métricas expuestas anteriormente, la evaluación de la calidad y precisión de la extracción de términos náuticos no ha podido ser tan exhaustiva. Esto ha sido debido principalmente al breve periodo de tiempo disponible ya que, para poder obtener un conjunto de datos con la entidad suficiente y requerida para extraer conclusiones válidas, el programa necesita un ciclo de entrenamiento más amplio con el cual poder acumular resultados y almacenarlos en un registro.

Sin embargo, para demostrar el funcionamiento del código, en este apartado se van a realizar varios experimentos consecutivos de complejidad creciente, más concretamente tres en los cuales el programa desarrolla un funcionamiento correcto y uno adicional con el fin de encontrar carencias derivadas, precisamente, de la fase inicial en la que se encuentra este trabajo.

4.2.3.1 Prueba empírica 1 usando la primera aproximación

Esta primera prueba, se llevará a cabo con el programa entrenado en inglés. La oración dictada es la siguiente: *"Good morning Sir, this is a fishing vessel sailing on your coast".* En la Figura 4-19, se puede ver que para la prueba se ha seleccionado la entrada de audio "*Built-in Microphone*" que corresponde con el elemento "0" de la lista. A continuación, el programa ha grabado la voz y en la línea de "Texto Transcrito" muestra el audio pasado a texto.

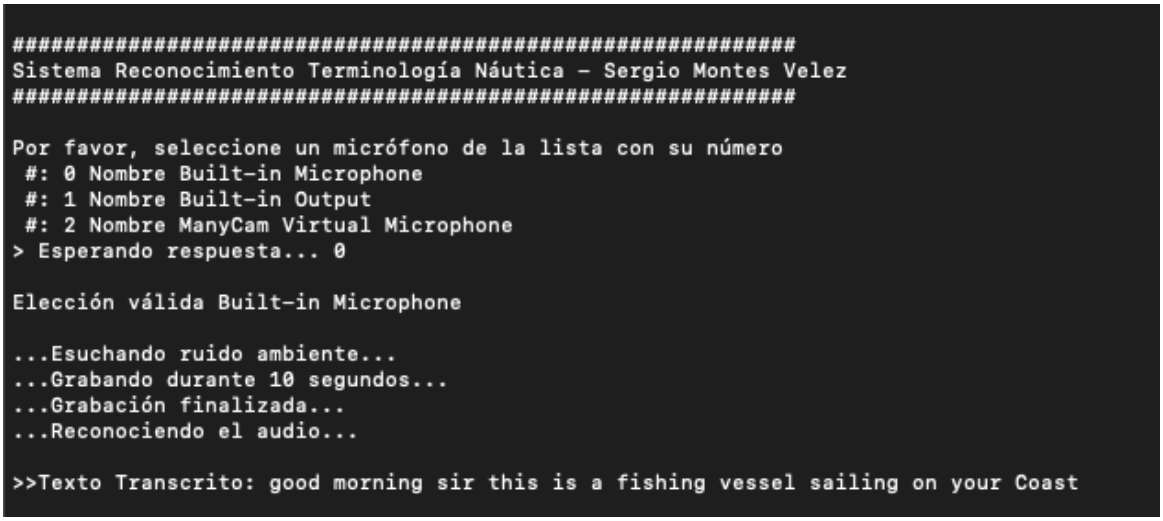


Figura 4-19 Apartado 1 del programa desarrollado (Propia)

El siguiente paso es ejecutar el algoritmo para llevar a cabo la clasificación de la oración. En este caso el seleccionado ha sido el *Random Forest Classifier*, pues como se ha demostrado en el apartado 4.2.2 es el que se ha obtenido un mejor rendimiento con el idioma inglés. Como puede verse en la Figura 4-20, el algoritmo ha desarrollado un rendimiento del 86,778% y ha etiquetado la frase como "náutica".

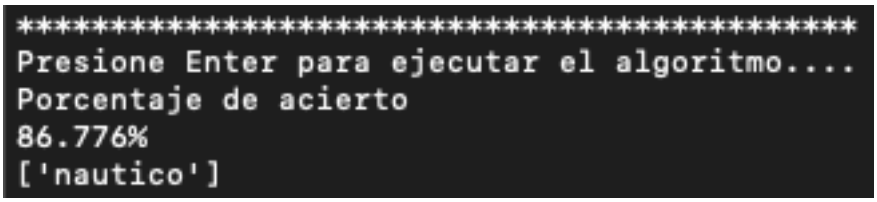


Figura 4-20 Apartado 2 del programa desarrollado (Propia)

Por último, se va a proceder a la extracción de terminología náutica de la oración dictada. Para ello usaremos el diccionario creado de forma automática a partir del glosario de términos náuticos, descrito en el apartado 3.3.1.3.1. En la Figura 4-21 se pueden encontrar los términos náuticos que el programa ha sido capaz de extraer.

Figura 4-21 Apartado 3 programa desarrollado (Propia)

4.2.3.2 Prueba empírica 2 usando la primera aproximación

En esta ocasión, se indicará una oración en inglés más compleja que en el experimento anterior. El resto del proceso será exactamente el mismo. En esta prueba, el mensaje radio simulado ha sido: *"This is an English Navy Warship asking for help. Our ships have been hit on the portside and the decks are flooding. The crew is safe"*. En la Figura 4-22 se puede observar el texto transcrito, en la Figura 4-23 la etiqueta "náutico" y por último en la Figura 4-24 los términos náuticos extraídos.

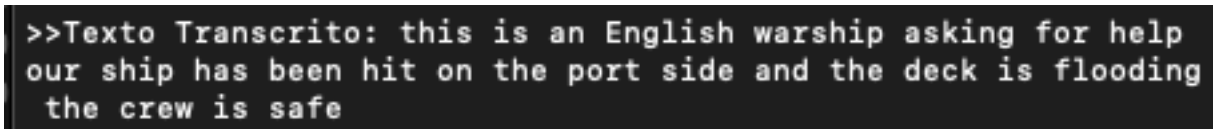


Figura 4-22 Texto transcrito (Propia)

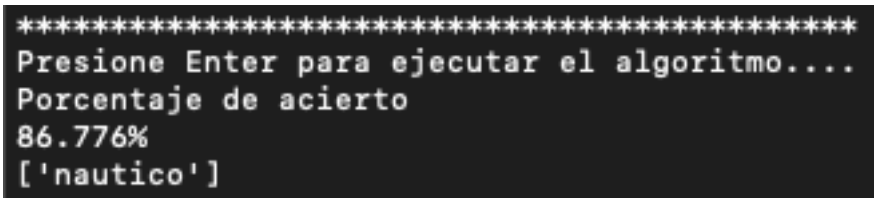


Figura 4-23 Etiqueta y porcentaje de acierto (Propia)

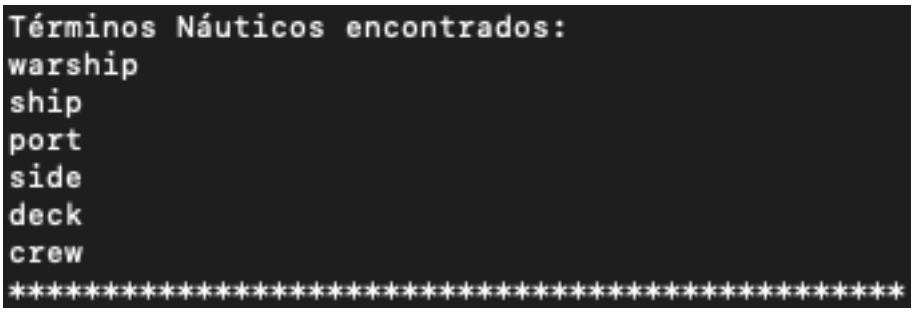


Figura 4-24 Términos náuticos encontrados (Propia)

Como puede observarse en la Figura 4-24, a pesar de haber dictado los términos "*ships"* y "*decks"* en plural, gracias a la lematización el programa ha sido capaz de obtener dichos términos como su entrada en el diccionario

4.2.3.3 Prueba empírica 3 usando la primera aproximación

Para acabar con las pruebas empíricas usando la primera aproximación, se va a introducir una frase con la intención de que, aun siendo náutica, no la etiquete como tal. Gracias a procesos como este se podrán implementar mejoras en el funcionamiento de los algoritmos, así como en el dataset usado para entrenarlos. La frase elegida ha sido: "*We had an accident due to the storm. The keel hit a rock and we are stranded"*. Se ha omitido el proceso de reconocimiento de voz para agilizar la prueba y, como se puede ver en la Figura 4-25, los algoritmos no lo han clasificado como náutica.

```
**********************************************
Presione Enter para ejecutar el algoritmo....
Porcentaje de acierto
86.776%
['no nautico']
```

Figura 4-25 Etiqueta de la frase (Propia)

4.2.3.4 Prueba empírica 4 usando la segunda aproximación

En esta ocasión, el procedimiento vuelve a ser el mismo, pero la prueba va a realizarse sobre la aproximación independiente del idioma en la que se añadía un módulo de traducción. La oración dictada esta vez será en español: *"Buenos días, soy el Capitán del barco. Nuestro ancla y timón se han estropeado y no podemos fondear. Si la maniobra lo permite vamos a acercarnos a la posición más cercana a las fuerzas amigas."*. En la Figura 4-26 se muestra de manera adicional al texto transcrito, la traducción al inglés del mismo ya que como se explicó en el apartado 0 dicha traducción se hace necesaria al trabajar con el dataset en inglés. En la Figura 4-27 el porcentaje de acierto y la predicción del algoritmo y por último en la Figura 4-28 los términos náuticos encontrados.

```
>>Texto Transcrito: buenos días soy el capitán del barco nuestro ancla y
 timón se han estropeado y no podemos fondear si la maniobra lo permite
vamos a acercarnos a la posición más cercana a las fuerzas amigas

>>Texto en ingles: Good morning I am the captain of the ship our anchor
and rudder have spoiled and we can not anchor if the maneuver allows us
we are going to approach the closest position to the friendly forces
```

Figura 4-26 Texto transcrito y traducido (Propia)

```
**********************************************
Presione Enter para ejecutar el algoritmo....
Porcentaje de acierto
86.776%
['nautico']
```

Figura 4-27 Etiqueta y porcentaje de acierto (Propia)

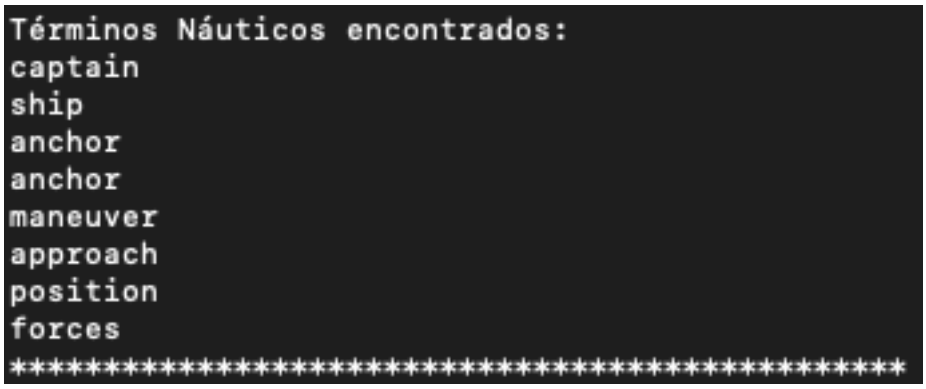


Figura 4-28 Términos náuticos encontrados (Propia)

4.2.3.5 Prueba empírica 5 usando la segunda aproximación

Para finalizar con los experimentos, y con el fin de encontrar deficiencias al programa, en esta prueba empírica se va a intentar encontrar alguna frase náutica que los algoritmos no sean capaces de etiquetar como tal. Para ello, se va a omitir el proceso de transcribir el audio a texto y se trabajará directamente sobre el texto traducido al inglés. La frase elegida ha sido: *"Durante la regata tuvimos que cambiar el aparejo para aprovechar los vientos del norte"* que, como puede verse en la Figura 4-29 ha sido etiquetada como "no náutica".

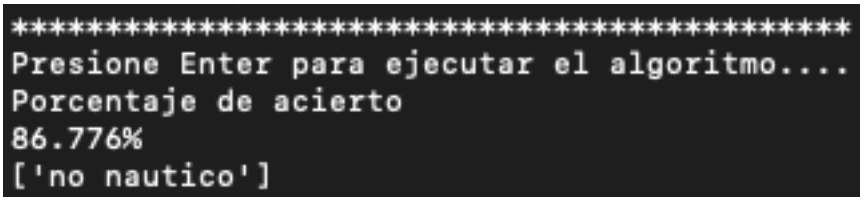


Figura 4-29 Etiqueta y porcentaje de acierto (Propia)

Además, como puedo verse en la Figura 4-30 con el diccionario creado automáticamente explicado en el apartado 3.3.1.3.1, el programa ha determinado que tanto "*change*" como "*wind*" son términos náuticos, resultado que difiere de la realidad.

Figura 4-30 Términos náuticos encontrados (Propia)

Como se puede observar en las Figura 4-29 y Figura 4-30, a pesar de que la frase dictada inicialmente se hizo en español, debido a que el programa trabaja sobre la traducción de la oración transcrita y que, tanto el dataset como el diccionario han sido creados para el inglés, los términos que extrae aparecen en dicho idioma. Lo ideal sería que, a partir de estos términos y en base a las relaciones existentes entre ellas, se pudieran extrapolar los resultados para que dicha información obtenida se mostrase directamente sobre el mensaje en el idioma original, es decir, español.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

Habiendo establecido los objetivos en el apartado 1.2 y tras haber valorado el desarrollo y resultados del presente TFG, puede afirmarse que se con este proyecto se ha conseguido satisfacer las necesidades.

Se ha comprobado que, a pesar de haber contado con algunas dificultades como la inexistencia de un dataset adecuado a las necesidades del trabajo, se ha logrado desarrollar dos aproximaciones a una metodología de trabajo y, por consiguiente, un programa que la aplique. En cualquiera de los casos, el objetivo de filtrar los mensajes vía radio con el fin de liberar a los operadores en tierra puede darse por cumplimentado.

Así, por un lado, la primera aproximación del sistema es capaz de escuchar audio e introducirlo en un algoritmo de aprendizaje supervisado en un idioma concreto. Es por ello que se ha hecho necesario la creación de un dataset multilingüe, en el que al menos uno de los idiomas ahí presentes tiene que ser conocido por la persona que los iba a etiquetar. Concretamente se ha probado el sistema con dos idiomas (inglés y francés), y estos son los que han permitido entrenar los algoritmos en ambos casos. El algoritmo resultante para el inglés ha logrado un rendimiento 86%, mientras que el del francés se aproximaba al 80%.

Por otro lado, la segunda aproximación desarrollada es independiente del idioma, gracias a la incorporación de un módulo de traducción, en la que solamente es necesario disponer de un programa entrenado en inglés para lidiar con la problemática de recibir mensajes en algún idioma para el cual no se dispusiera de dataset, como puede ser el caso de idiomas arábigos. De este modo, el sistema sería capaz de etiquetar dicha oración en función de lo aprendido en inglés, y luego realizaría la extracción de los términos relevantes en lo que a náutica se refiere. Con la creación de estas dos aproximaciones se puede afirmar que el sistema es capaz de trabajar en entornos multilingües.

De manera adicional, también se ha cumplido con el objetivo de extracción de terminología náutica gracias al módulo de extracción que, de forma automática crea un diccionario náutico a partir de un glosario obtenido de Internet, para posteriormente volver a filtrar el mensaje recibido y ser capaz de identificar aquellos términos que convierten el mensaje en información relevante.

5.2 Líneas futuras

A pesar de haber alcanzado los objetivos propuestos, todo proceso es susceptible a cambios y mejoras, y este TFG no es una excepción.

La base del éxito de los procesos de aprendizaje automático no es otra que los datasets usados para su entrenamiento. Por este motivo, es de vital importancia que estos sean extensos y completos (deseablemente balanceados). Como línea futura se plantea dar continuidad al proceso de ampliación de datasets (incrementando el número de oraciones), tanto para inglés y francés como para otros idiomas menos comunes.

Además, uno de los puntos fuertes de disponer de un sistema de aprendizaje automático es la retroalimentación. Actualmente el diccionario náutico cuenta con un léxico limitado y poco preciso, pero, gracias al uso de inteligencia artificial, puede aumentar su entidad y calidad de manera exponencial. Es por ello que como línea futura también se plantea la posibilidad de que el mismo sistema a medida que analiza mensajes, sea capaz de extraer por si solo más léxico náutico a partir de los mensajes que analiza para posteriormente volcarlo en el diccionario así como la perfección del sistema de extracción de terminología náutica para asegurarse que el diccionario contiene exclusivamente el léxico deseado ya que, como se ha comprobado en el apartado 4.2.3.5, el diccionario actualmente no cuenta con términos exclusivamente náuticos.

Por último, es indudable que los procesos de traducción pueden dar lugar a errores y por tanto la fiabilidad de la aproximación independiente de idioma puede verse comprometida. Como solución a este problema, se plantea una línea futura en la que, mediante algoritmos de PLN el sistema sea capaz de extrapolar los resultados obtenidos para ser usados directamente en otro idioma sin la necesidad de ninguna traducción, como se ha visto en el apartado 4.2.3.5.

Para llevar a cabo este proceso se ha pensado en una posibilidad. Esta consiste en hacer uso de la relación existente entre las palabras pertenecientes a una misma oración, es decir, si para el inglés el programa ha detectado como náutico un término que es un sustantivo y a su vez está modificado por dos adjetivos dentro de un sintagma nominal, en el idioma al se desea extrapolar los resultados, dicho término náutico es probable que se encuentre en sintagma que desempeñe la misma función (además de que pueda estar siendo modificado a su vez por adjetivos). Desde el punto de vista de la sintaxis, se trataría de conseguir extrapolar los resultados a través de árboles sintácticos.

6 BIBLIOGRAFÍA

En esta sección figurarán todas las referencias, sean recursos web, libros, artículos, etc., incluyendo la información de autores, título de la obra, nombre de la publicación, año, edición y enlace más fecha de último acceso en el caso de referencias a recursos online.

[1] Gradiant, «La digitalización de las comunicaciones marítimas,» vol. 1, p. 103, 2019.

[2] VesselFinder, «VesselFinder,» [En línea]. Available: https://www.vesselfinder.com/es. [Último acceso: 1 Marzo 2021].

[3] J. Salama, «Salama,» [En línea]. Available: https://www.salama.es/noticia/ver/id/79/titulo/el-estrecho-de-gibraltar-la-ruta-maritima-mas-transitada-del-mundo.html#:~:text=El%20Estrecho%20de%20Gibraltar%20se,del%2010%25%20del%20tr%C3%A1fico%20internacional.. [Último acceso: 1 Marzo 2021].

[4] G. Nail y A. Platonov, «IV International Research Conference "Information Technologies in Science, Management, Social Sphere and Medicine",» de Advances in Computer Science Research (ACSR), volume 72, 2017.

[5] A. Bocanegra-Valle, «The Language of Seafaring: Standardized Conventions and Discursive Features in Speech Communications,» International Journal of English Studies (Universidad de Murcia), Cádiz, 2010.

[6] Mehdi Allahyari, Saied Safaei, Seyedamin Pouriyeh, Elizabeth D. Trippe, Krys Kochut, Mehdi Assefi y Juan B. Gutierrez, «KDD Bigdas,» de A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques, Halifax, Canada, 2017.

[7] M. Delgado, «La inteligencia Artificial Realidad de un mito moderno,» de Discurso de apertura de universidad de Granada, Granada, 1996.

[8] L. Rouhiainen, Inteligencia Artificial 101 cosas que debes saber hoy sobre nuestro futuro, Barcelona: Alienta, 2018.

[9] L. Rouhiainen, «Amazon,» Amazon, [En línea]. Available: https://www.amazon.es/Artificial-Intelligence-Things-Today-Future/dp/1982048808. [Último acceso: 25 Febrero 2021].

[10] Aprende IA, «AprendeIA,» [En línea]. Available: https://aprendeia.com/aprendizaje-supervisado-logistic-regression/#:~:text=La%20regresi%C3%B3n%20log%C3%ADstica%20o%20Logistic,de%20una%20variable%20dependiente%20categ%C3%B3rica.&text=Permite%20decir%20que%20la%20presencia,resultado%20dado%20un%20porcent. [Último acceso: 2 Marzo 2021].

[11] E. Anguiano-Hernández, «Naive Bayes Multinomial para Clasificaci´on de Texto Usando un Esquema de Pesado por Clase».

[12] A. Kesh, «Geek for geeks,» 17 Julio 2021. [En línea]. Available: https://www.geeksforgeeks.org/passive-aggressive-classifiers/#:~:text=The%20Passive%2DAggressive%20algorithms%20are,even%20intermediate%20Machine%20Learning%20enthusiasts.&text=We%20can%20simply%20say%20that,then%20throw%20away%20the%20example.. [Último acceso: 2 Marzo 2021].

[13] J. A. Rodrigo, «Ciencia de datos,» [En línea]. Available: https://www.cienciadedatos.net/documentos/py08_random_forest_python.html. [Último acceso: 2 Marzo 2021].

[14] A. t. (. report. [En línea]. Available: http://www.hutchinsweb.me.uk/MTNI-14-1996.pdf. [Último acceso: 6 Marzo 2021].

[15] Aprende machine learning, «Aprende machine learning,» 27 Diciembre 2018. [En línea]. Available: https://www.aprendemachinelearning.com/procesamiento-del-lenguaje-natural-nlp/. [Último acceso: 20 Febrero 2021].

[16] Macarena, «Inteligencia Analitica,» 3 Octubre 2016. [En línea]. Available: https://inteligencia-analitica.com/procesamiento-lenguaje-natural/. [Último acceso: 6 Marzo 2021].

[17] N. L. Processing, «Michael R. Williams,» [En línea]. Available: https://dl.acm.org/doi/pdf/10.5555/1074100.1074630. [Último acceso: 6 Marzo 2021].

[18] M. L. Mendoza, «Open Webinars,» 16 Julio 2020. [En línea]. Available: https://openwebinars.net. [Último acceso: 20 Enero 2021].

[19] E. IT, «Educacion IT,» 24 Septiembre 2018. [En línea]. Available: https://blog.educacionit.com. [Último acceso: 20 Enero 2021].

[20] A. F. Montoro, Python 3 al descubierto, Madrid: RC Libros, 2012.

[21] «Cosas de devs,» [En línea]. Available: https://cosasdedevs.com/python/. [Último acceso: 28 Enero 2021].

[22] J2LOGO, «J2logo,» [En línea]. Available: https://j2logo.com/python/tutorial/programacion-orientada-a-objetos/. [Último acceso: 24 Febrero 2021].

[23] Jython, «Jython,» [En línea]. Available: https://www.jython.org/. [Último acceso: 3 Marzo 2021].

[24] J. Huerta, «Paradigma Digital,» 2018. [En línea]. Available: https://www.paradigmadigital.com/dev/es-python-el-lenguaje-del-futuro/. [Último acceso: 1 Marzo 2021].

[25] HomeBrew, «HomeBrew,» Homebrew, [En línea]. Available: https://brew.sh/.

[26] M. Broberg, «Open Source,» 1 Diciembre 2020. [En línea]. Available: https://opensource.com/article/19/5/python-3-default-mac.

[27] Conda, «Conda,» [En línea]. Available: https://docs.conda.io/en/latest/miniconda.html.

[28] Visual Studio, «Visual Studio,» Visual Studio, [En línea]. Available: https://code.visualstudio.com/. [Último acceso: 6 Marzo 2021].

[29] Marc, «Sobrebits,» 7 Noviembre 2018. [En línea]. Available: https://sobrebits.com/primeros-pasos-con-visual-studio-code-para-powershell/. [Último acceso: 15 Febrero 2021].

[30] Mariano, «Pythones,» [En línea]. Available: https://pythones.net/importar-modulos-en-python/. [Último acceso: 15 Febrero 2021].

[31] Pherkad, «Python para impacientes,» 9 Febrero 2014. [En línea]. Available: https://python-para-impacientes.blogspot.com/2014/02/expresiones-regulares.html. [Último acceso: 6 Marzo 2021].

[32] A. S. Lamadrid y A. Suarez Jimenez, «Python 3 para impacientes,» 26 Agosto 2015. [En línea]. Available: https://python-para-impacientes.blogspot.com/2015/08/bucles-eficientes-con-itertools.html. [Último acceso: 15 Febrero 2021].

[33] Aprende con Alf, «Aprende con Alf,» 4 Octubre 2020. [En línea]. Available: https://aprendeconalf.es/docencia/python/manual/numpy/. [Último acceso: 15 Febrero 2021].

[34] Aprende con Alf, «Aprende con Alf,» 4 Octubre 2020. [En línea]. Available: https://aprendeconalf.es/docencia/python/manual/pandas/. [Último acceso: 15 Febrero 2021].

[35] Universidad de Alcalá, «Master Data Scientist,» [En línea]. Available: https://www.master-data-scientist.com/scikit-learn-data-science/. [Último acceso: 15 Febrero 2021].

[36] Aprende IA, «Aprende IA,» [En línea]. Available: https://aprendeia.com/libreria-pandas-de-matplotlib-tutorial/. [Último acceso: 15 Febrero 2021].

[37] G. Fosado, «Medium,» 24 Julio 2019. [En línea]. Available: https://medium.com/@yeralway1/primeros-pasos-en-nlp-con-spacy-un-vistazo-general-734686843a57. [Último acceso: 6 Marzo 2021].

[38] J. M. Heras, «IA Artificial,» 10 Octubre 2020. [En línea]. Available: https://www.iartificial.net/librerias-de-python-para-machine-learning/#gensim. [Último acceso: 6 Marzo 2021].

[39] Red Hat, «Red Hat,» [En línea]. Available: https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces. [Último acceso: 20 Febrero 2021].

[40] D. Amos, «Real Python,» [En línea]. Available: https://realpython.com/python-speech-recognition/. [Último acceso: 20 Febrero 2021].

[41] Scikit-learn, «Scikit-learn,» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

[Último acceso: 20 Febrero 2021].

[42] Wikipedia, «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Glossary_of_nautical_terms. [Último acceso: 6 Marzo 2021].

[43] G. R. d. Villa, «Medium,» 18 Mayo 2018. [En línea]. Available: https://gruizdevilla.medium.com/introducción-a-word2vec-skip-gram-model-4800f72c871f. [Último acceso: 9 Marzo 2021].

[44] L. Castro, «About Español,» 1 Noviembre 2019. [En línea]. Available: https://www.aboutespanol.com/imdb-la-base-de-datos-del-mundo-del-cine-y-television-157980. [Último acceso: 8 Marzo 2021].

[45] IMDb, «IMDb,» [En línea]. Available: imdb.com.

[46] Subscene, «Subscene,» [En línea]. Available: https://subscene.com/.

[47] «Burner,» [En línea]. Available: https://burner.com/srt-to-csv/. [Último acceso: 10 Marzo 2021].

[48] J. M. Heras, «I Artificial,» 9 Octubre 2020. [En línea]. Available: https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/. [Último acceso: 3 Marzo 2021].

[49] M. Vyas, «Secjuice,» 23 Junio 2019. [En línea]. Available: https://www.secjuice.com. [Último acceso: 20 Enero 2021].

[50] V. T. Aranda, «Historia y evolución de los lenguages de programación,» ACTA, Vols. %1 de %21888-6051, nº 34, pp. 85-95, 2004.

[51] IBM, «IBM,» IBM, [En línea]. Available: ibm.com. [Último acceso: 23 Enero 2021].

[52] «Computer History,» [En línea]. Available: computerhistory.org. [Último acceso: 23 Enero 2021].

[53] C. R. Martin, «Stack Over Flow,» 20 Abril 2020. [En línea]. Available: https://stackoverflow.blog/2020/04/20/brush-up-your-cobol-why-is-a-60-year-old-language-suddenly-in-demand/. [Último acceso: 26 Enero 2021].

[54] Vintage Computer, «Vintage Computer,» 19 Enero 2012. [En línea]. Available: https://vintagecomputer.com/mits-altair-8800.html. [Último acceso: 26 Enero 2021].

[55] P. P. G. Abenza, Comenzando a programar con JAVA, Universidad Miguel Hernández de Elche, 2015.

[56] R. Sedgewick y K. Wayne, «IntroCs,» 30 Octubre 2019. [En línea]. Available: https://introcs.cs.princeton.edu/java/11cheatsheet/. [Último acceso: 26 Enero 2021].

[57] F. S. Caparrini, «Cs,» Cs, 14 Diciembre 2020. [En línea]. Available: http://www.cs.us.es/~fsancho/?e=77. [Último acceso: 13 Febrero 2021].

[58] James Gosling y Henry McGilton, «Oracle,» Oracle, Mayo 1996. [En línea]. Available: https://www.oracle.com/java/technologies/language-environment.html. [Último acceso: 26 Enero 2021].

[59] S. Pramanick, «GeeksforGeeks,» 28 Octubre 2019. [En línea]. Available: https://www.geeksforgeeks.org/python2-vs-python3-syntax-and-performance-

comparison/. [Último acceso: 28 Enero 2021].

[60] P. G. Bejerano, «El diario.es,» 6 Febrero 2014. [En línea]. Available: https://www.eldiario.es/turing/criptografia/alan-turing-enigma-codigo_1_5038272.html. [Último acceso: 25 Febrero 2021].

[61] D. Datta, «Digit,» 15 Abril 2020. [En línea]. Available: https://www.digit.in/features/tech/digit-mag-the-origins-of-artificial-intelligence-53402.html. [Último acceso: 25 Febrero 2021].

[62] N. Delam, «Medium,» 22 Agosto 2016. [En línea]. Available: https://medium.com/@nazanindelam/single-layer-artificial-neural-networks-a91cf3752a86. [Último acceso: 27 Febrero 2021].

[63] «Web de La Moncloa,» [En línea]. Available: http://www.lamoncloa.gob.es. [Último acceso: 13 enero 2015].

[64] J. Rodríguez y V. Fernández, Cómo redactar el estado del arte de un trabajo, Editorial Genios, 2010.

[65] P. Martínez y A. García, Cómo escribir una buena memoria de TFG, Publicaciones del 2000, 2013.

[66] A. Pérez, Cómo escribir una bibliografía, Nuevas publicaciones.

[67] Edube, «Edube,» [En línea]. Available: https://edube.org/learn/programming-essentials-in-python-part-1-spanish/python-una-herramienta-no-un-reptil. [Último acceso: 28 Enero 2021].

[68] ABC, «ABC,» [En línea]. Available: https://www.abc.es/historia/abci-profetica-tumba-papa-maldito-pacto-diablo-201512300154_noticia.html. [Último acceso: 25 Febrero 2021].

[69] J. González, «Data Centric,» 2 Enero 2019. [En línea]. Available: https://www.datacentric.es/blog/business/procesamiento-lenguaje-natural-revolucion-futuro/. [Último acceso: 1 Marzo 2021].

[70] M. Chaudhary, «Medium,» 24 Abril 2020. [En línea]. Available: https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a. [Último acceso: 2 Marzo 2021].

[71] Google, «AnunciosGoogle,» [En línea]. Available: https://www.anunciosgoogle.net/que-es-el-tf-idf/. [Último acceso: 2 Marzo 2021].

[72] KeepCoding, «Keep Coding,» [En línea]. Available: https://keepcoding.io/blog/que-son-datasets/. [Último acceso: 18 Febrero 2021].

[73] Neptuno, «Neptuno,» [En línea]. Available: https://www.neptuno.es/que-es-un-sistema-ais/. [Último acceso: 3 Marzo 2021].

ANEXO I: HISTORIA Y EVOLUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

En las primeras etapas de la informática, dicha comunicación o lenguaje se llevaba a cabo haciendo uso del código binario, ya que era la forma de trabajar que tenían los procesadores, y en ocasiones recibía el nombre de lenguaje de máquina o código máquina.

Pero la programación en dicho lenguaje resultaba muy difícil, poco eficiente y tediosa, debido a que los datos, archivos e instrucciones se debían introducir en código binario, además era obligatorio conocer las posiciones en memoria donde se almacenaban los datos. Todo ello daba lugar a un gran número de errores y a largos tiempos de depuración.

Por esta causa y motivados por la mejora y el desarrollo, a principios de los años 50 se inventó una notación simbólica, *ASSEMBLY* (Figura A1- 1) (o código de ensamblaje) que usa un conjunto de abreviaturas más sencillas y fáciles de recordar que el código binario, para representar las operaciones: ADD (sumar), STORE (copiar), etc.

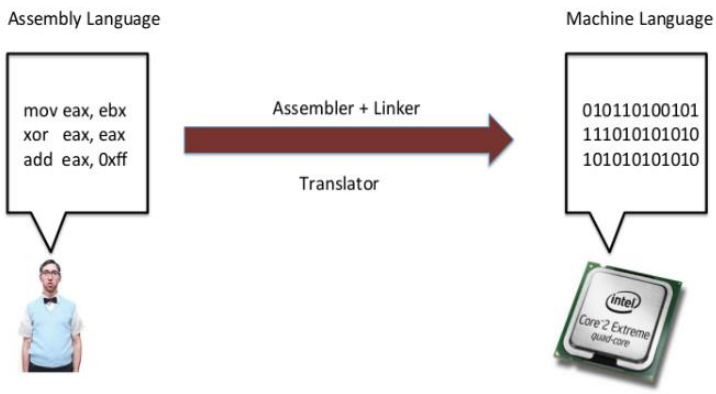


Figura A1- 1 Assembly language (extraída de [49])

Inicialmente, la traducción del código de maquina a código de ensamblaje se llevaba a cabo de manera manual, pero no tardaron en darse cuenta de que los ordenadores eran capaces de encargarse de dicha tarea, y así fue como se desarrolló un programa capaz de traducir estos lenguajes, y lo llamaron ASSEMBLER (ensamblador).

A medida que los ordenadores fueron implementándose en el ámbito empresarial y académico, convirtiéndose en una herramienta cada vez más empleada, los arcaicos y complicados lenguajes de programación fueron evolucionando hacia otros más sencillos de aprender y más cómodos para el usuario. Estos lenguajes, conocidos por el nombre de 'Lenguajes de alto nivel', tienen una arquitectura y sintaxis que se asemeja notablemente a la forma de comunicarse de los humanos. Algunos de los mas conocidos son 'BASIC', 'PASCAL', 'C', etc.

A pesar de que son múltiples, si no centenares, los lenguajes de programación existentes actualmente, solamente unos cuantos son ampliamente empleados.

Antes de comentar los principales lenguajes de programación, hay que definir una característica común a todos ellos: las órdenes y comandos dados en cualquier tipo de lenguaje siempre deben traducirse a aquel código binario que los ordenadores son capaces de interpretar en su unidad central. Este trabajo o labor de traducción se realiza a través de un intérprete o compilador.

El intérprete informático podría asemejarse a un intérprete encargado de traducir, a tiempo real, una conversación entre dos hablantes de distinto idioma. Traduce instrucción a instrucción para

mejorar la interactividad, la depuración y puesta a punto del programa, la ejecución inmediata de una orden entre otros.

Sin embargo, existe otra forma de traducción, la escrita, y evidentemente presenta diferencias con respecto a la *traducción oral*. Por ejemplo, no tendría sentido tener un intérprete para cada ciudadano si lo que se pretende es leer un artículo en otro idioma, la forma más eficaz de abordar el problema sería realizando una copia de ese mismo articulo, pero en otro idioma.

El equivalente informático de esta forma de trabajar, es el compilador. A diferencia de un intérprete, un compilador traduce todo el programa al mismo tiempo, dejándolo preparado para ser ejecutado. De esta manera se consigue un mayor y mejor flujo en la ejecución y, además, se liberan recursos de la memoria, ya que el programa, una vez compilado, no necesita que el traductor esté ejecutándose en segundo plano, como sucedería en el caso de usar un intérprete.

Pero no todo son ventajas en los lenguajes compilados, ya que la depuración resulta más cómoda y eficiente con un intérprete. Esto se debe a que un compilador no informa de los posibles errores hasta que llega el momento de compilar el programa. Además, cada vez que se realice alguna modificación en el código, es necesario volver a compilarlo (traducirlo) de nuevo.

Los primeros lenguajes de programación de alto nivel [50]

FORTRAN

A principios de la década de los 50, John Backus trabajaba con SSEC (*Selective Sequence Electronic Calculator),* uno de los primeros ordenadores de IBM[3], y desarrolló el programa SPEEDCODING para él. Utilizando éste como base, comenzó la creación, en 1954, de un lenguaje de programación para implementarle más prestaciones al modelo IBM 704, que saldría al mercado próximamente.

Figura A1- 2 IBM 704 (extraída de [51])

En el año 1956 finalizó el compilador FORTRAN (FORmula TRANslator) y pudo incluirlo en el IBM 704 (Figura A1- 2) junto con su manual de 51 páginas (Figura A1- 3).

FORTRAN, como su nombre indica, estaba (y actualmente también) destinado a la resolución de problemas científico-técnicos, algo que resultaba relativamente sencillo de aprender, eso si, si se dominaba la notación matemática.

A pesar de que ha ido perfeccionándose a lo largo de los años, la realidad es que se ha visto superado por numerosos lenguajes, principalmente debido a que sus programas carecen de una buena

[3] International Business Machines Corporation. Reconocida empresa multinacional estadounidense de tecnología y consultoría, con sede en Nueva York.

estructura y resultan difíciles para los usuarios. A pesar de ello sigue usándose en el ámbito universitario, por ejemplo, al disponer de una gran biblioteca de subrutinas y funciones que ha ido creciendo y mejorando en sus más de treinta años de vida.

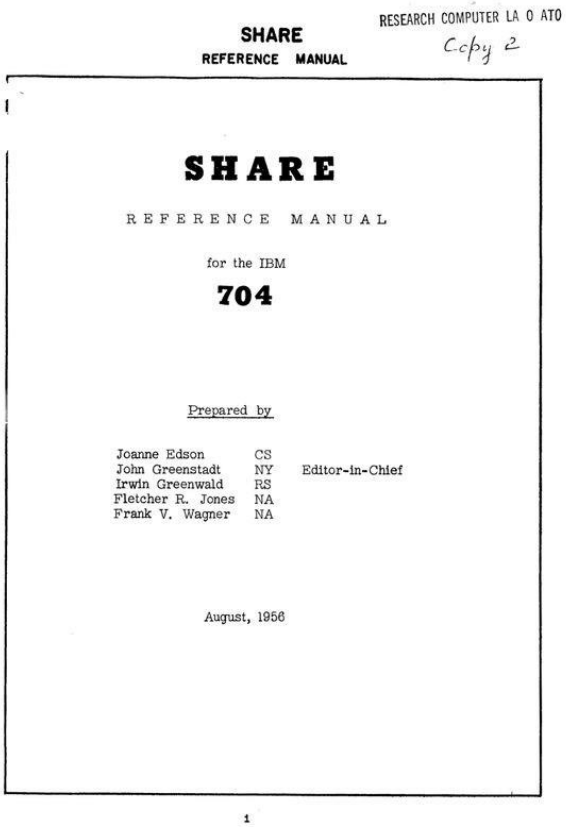


Figura A1- 3 Manual IBM 704 (extraída de [52])

COBOL

En los años 50, el Departamento de Defensa de los Estados Unidos presentaba un alto nivel de interés en los lenguajes de programación, principalmente por dos motivos: los programas de la época no podían llevarse de un ordenador a otro y además resultaban complejos de leer y modificar.

Para solucionar estas dificultades, impartió una conferencia sobre lenguajes, en 1959, y en la que participaron las grandes empresas del sector, entre ellas IBM. En consecuencia, a esa conferencia, fue posible el desarrollo de COBOL (COmmon Business Oriented Language), un lenguaje destinado a funciones administrativas, portable y de fácil lectura. Su primera versión nació al año siguiente y con el paso del tiempo implementaron nuevas actualizaciones.

Debido a que uno de los objetivos era la buena legibilidad, COBOL presenta una sintaxis muy parecida al inglés, cuya terminología aparece continuamente: verbos, párrafos, frases…). Por ello, los programas se estructuraban en cuatro divisiones, que se subdividían en secciones y éstas en párrafos, que constaban de frases e instrucciones. En la Figura A1- 4 se puede observar una plantilla COBOL.

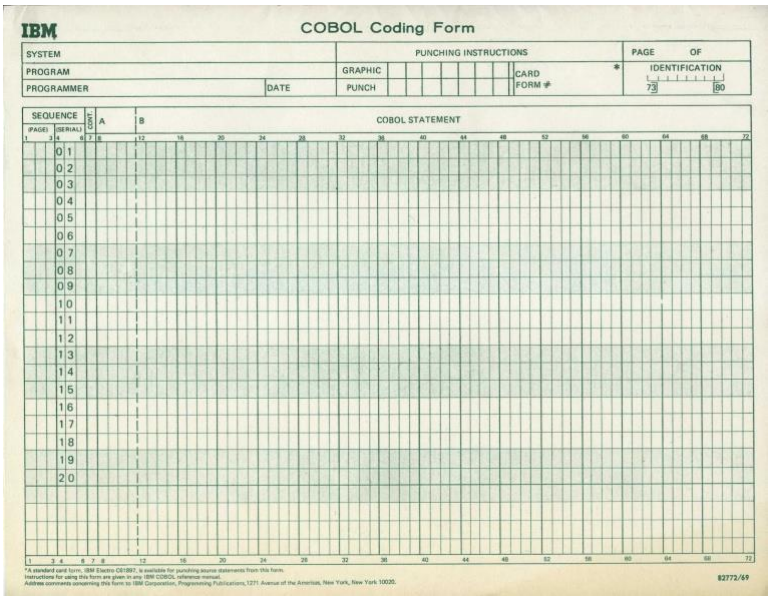


Figura A1- 4 COBOL coding form (extraída de [53])

Actualmente, COBOL se utiliza de manera casi exclusiva en algunos grandes sistemas informáticos (principalmente entidades bancarias) con el objetivo de mantener el código existente y no tener que desarrollar ninguno nuevo.

BASIC

John G. Kemeny y Thomas E. Kurtz, siendo profesores del Dartmouth College, en 1964 desarrollaron un nuevo lenguaje que permitía a sus estudiantes un tener un acercamiento a los sistemas de tiempo compartido. Ese lenguaje lo bautizaron con el nombre de BASIC debido a su sencillez y se utilizó tanto en tareas de gestión como en aplicaciones científicas. Gran parte de su popularidad fue debido a que era un lenguaje sencillo de aprender y, además, su intérprete ocupaba poca memoria. Es por ello que cuando se desarrolló el primer ordenador personal (Altair de MITS Figura A1- 5), se desarrolló un BASIC para él, y de ello se encargó Microsoft.

Figura A1- 5 ALTAIR MITS 8800 (extraída de [54])

Posteriormente, la misma empresa, adaptó su BASIC a los productos de Apple (sus microordenadores) y, más importante, al PC de IBM.

Otro punto positivo de este lenguaje de programación, es que ha sabido adaptarse a los cambios con el paso del tiempo. Aunque las primeras versiones resultaban un tanto complejas, las actuales presentan bastante estructuración y son compiladas. Un ejemplo del avance de BASIC es el Visual BASIC, también de Microsoft.

JAVA

Fue desarrollado por la empresa Sun Microsystems [55] a principios de los años 90, y presentado de manera oficial en mayo de 1995 en la conferencia SunWorld. Recientemente, Sun Microsystems fue adquirida por Oracle Corporation, actual dueña de Java.

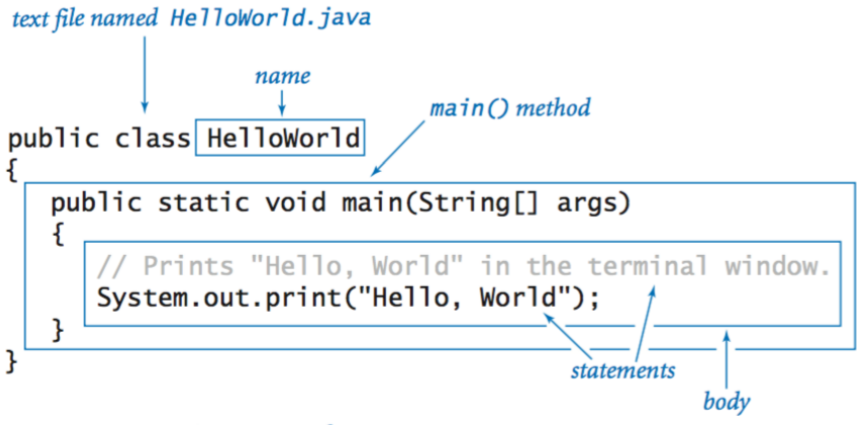


Figura A1- 6 'Hello World' en Java (extraída de [56])

El origen de Java estuvo motivado por la necesidad de crear alguna tecnología que permitiera desarrollar programas y aplicaciones capaces de ejecutarse en diferentes ordenadores y plataformas, es decir, en un entorno distribuido y heterogéneo. La idea inicial fue la de desarrollar un sistema operativo portable y ligero, en tiempo-real y distribuido, pero decidieron que lo óptimo sería crear un lenguaje de programación, además, con el crecimiento masivo de Internet, vieron nuevas oportunidades al cambiar el modo de distribuir los programas. En el documento *'The Java Language Environment'* [57] se detallan los objetivos fijados al comienzo del desarrollo del proyecto Java y se muestran sus características principales, que son: familiar, sencillo, multiplataforma, alto rendimiento, robusto, orientado a objetos, distribuido y concurrente. En la Figura A1- 6 se puede ver un ejemplo de programa en Java.

PYTHON

El origen de este lenguaje tuvo lugar a principios de los años 90. Un investigador holandés, Guido van Rossum (Figura A1- 7), que trabajaba en el centro de investigación CWI (Centrum Wiskunde & Informatica) de Ámsterdam, recibió la asignación de un proyecto para desarrollar un sistema operativo distribuido. En aquella época, el CWI usaba un lenguaje de programación conocido con el nombre de ABC, pero en esta ocasión, Guido decidió no usar este lenguaje e intentar crear otro nuevo que fuera capaz de superar las limitaciones y los inconvenientes con los que se había topado al trabajar con ABC. Esta fue, en consecuencia, la primera motivación que posteriormente daría lugar al nacimiento de Python.

Figura A1- 7 Creador de Python (extraída de [58])

La primera versión de Python vio la luz en 1991 pero no fue hasta 1994, tres años más tarde, cuando decidieron publicar la versión 1.0. Al principio el CWI liberó el intérprete del lenguaje bajo una licencia *open source* propia, pero en septiembre de 2000 y, a la vez que se publica la versión 1.6, tomaron la decisión de cambiar la licencia por una que fuera compatible con la GPL (*GNU General Public License*). Esta nueva licencia recibió el nombre de *Python Software Foundation License* y, a diferencia de la GPL, se trata de una licencia no *copyleft*. Eso significaba que era posible modificar el código fuente y el posible desarrollo de un código derivado sin la obligatoriedad de hacerlo *open source.*

Hasta la fecha solo se han lanzado tres versiones principales, teniendo cada una de ellas diversas actualizaciones. En lo que respecta a la versión 2, la última en ser liberada fue la 2.7, en julio de 2010. La versión 3 cuenta con la actualización 3.9.1, publicada en diciembre de 2020. Ambas versiones son mantenidas por separado. Esto significa que, tanto la 2.7 como la 3.9.1 se consideran estables, pero, lógicamente, correspondientes a diferentes versiones, pero algunas plataformas están empezando a dejar de soporte a la 2.7 debido a que puede estar obsoleta, sin embargo, entre ambas versiones existen diferencias que las hacen incompatibles. En la Figura A1- 8 se puede ver una comparación entre las versiones 2 y 3 de Python.

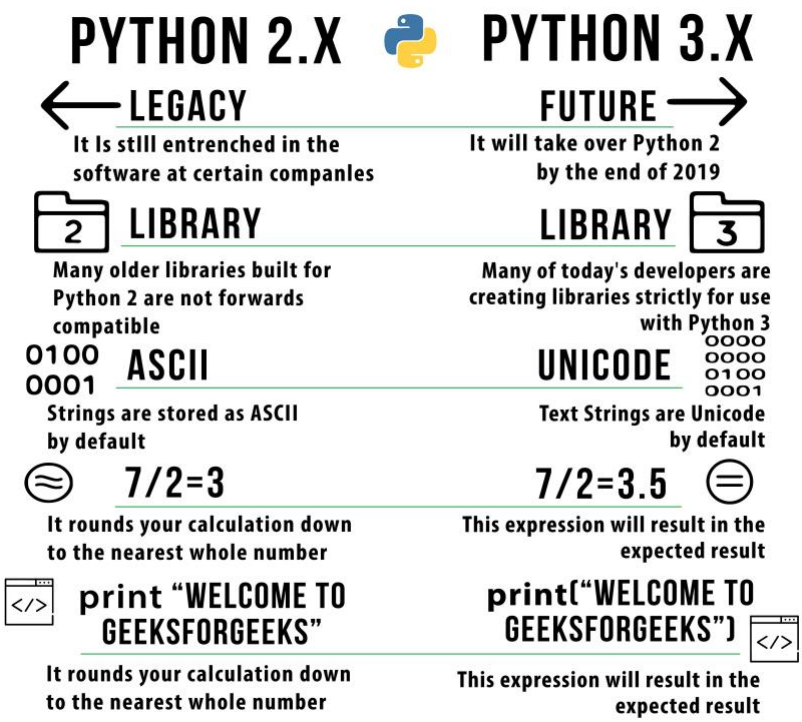


Figura A1- 8 Python2 vs Python3 (extraída de [59])

Entre las características de las primeras versiones de Python cabe reseñar el soporte de la orientación a objetos, el manejo de excepciones y el soporte de estructuras de datos de alto nivel, como, las listas y los diccionarios. Además, desde su desarrollo inicial, se tuvo en cuenta que el código escrito en este lenguaje fuera sencillo de leer y de aprender, sin que esto suponga renunciar a características y funcionalidades avanzadas.

Guido van Rossum decidió darle este nombre (Python) en honor a la serie de televisión *Monty Python´s Flying Circus*, de la cual era fan. Esta es una serie cómica protagonizada por el grupo de humoristas *Monty Python*, famoso por películas como *La vida de Brian* o *El sentido de la vida*. Desde el principio de su diseño, se intentó que Python fuera un lenguaje que resultara divertido de utilizar, de ahí que en el nombre influyera la mencionada serie cómica.

El desarrollo y promoción de Python se gestiona a través de una organización, sin ánimo de lucro, llamada *Python Software Foundation*. Entre las actividades que realiza esta organización destacan el desarrollo y distribución oficial de Python, la gestión de la propiedad intelectual del código y documentos realizados, así como la organización de conferencias y eventos dedicados a poner en contacto a todas aquellas personas interesadas en este lenguaje de programación.

ANEXO II: HISTORIA DE LA IA

[7] A diferencia de lo que se puede llegar a pensar, la IA tiene una historia igual de amplia que interesante. Algunos filósofos de sobrenombre han dedicado una parte de su tiempo a investigar cómo trabajan las máquinas y a especular sobre cómo podrían llegar a replicar la inteligencia humana.

Mucho antes de que existiera un campo de conocimiento llamado Inteligencia Artificial, antes incluso de que existieran los propios ordenadores, ya se planteaba la posibilidad de crear algún tipo de inteligencia externa al cuerpo humano.

Podemos remontarnos a la antigua Grecia, en la que Hefaistos, hijo de Hera, construía entes con apariencia humana, uno de los cuales, Talos, guardaba y defendía Creta. Pigmalión llegó a construir su propia mujer a partir de marfil, a la cual posteriormente Afrodita dio vida.

En el medievo, se dice que el papa Silvestre II (Figura A2- 1) fue capaz de construir una cabeza parlante que disponía de un vocabulario limitado y del poder de la premonición, haciendo predicciones futuras a preguntas que se le planteaban.

Figura A2- 1 Papa Silvestre II (extraída de [60])

Los mitos que han proporcionado tanto la ciencia ficción como la especulación científica, en el pasado y en la actualidad, contemplan la construcción de máquinas inteligentes como un logro alcanzable, algo con lo que hay que tener cautela pues como cita la célebre frase *lo que un hombre puede concebir otro podrá realizarlo.*

Merece la pena destacar que, la ciencia le debe a la propia ciencia ficción uno de los términos más conocidos en este ámbito, la palabra robot. Ésta, en checo significa "trabajador forzado" y aparece por primera vez en una de las novelas del autor Karel Kapec. Con este término el autor pretendía referirse a máquinas semi-inteligentes que realizaban trabajos de minería.

En la II Guerra Mundial los británicos y los americanos fueron los pioneros en usar máquinas para tareas complejas como cálculos numéricos avanzados o interpretación de claves criptográficas, actividades que anteriormente habían sido realizadas en su totalidad por la inteligencia del ser humano. De hecho, tanto ha evolucionado el concepto de inteligencia desde entonces, que actualmente las calculadoras están consideradas como realizadoras de procesos mecánicos y no "inteligentes" como se concebía antaño. Una de aquellas máquinas "inteligentes", más concretamente la diseñada por el brillante matemático Alan Turing, fue capaz de interpretar la clave que usaban los alemanes para cifrar sus comunicaciones, fue capaz de descifrar "Enigma" (Figura A2- 2).

Figura A2- 2 Maquina Enigma (extraída de [61])

En 1950 Turing escribió un artículo titulado "Computing Machines and Intelligence" por el cual se le reconoce como el padre de la inteligencia artificial. Dicho artículo comenzaba con una pregunta que hoy día continúa siendo debatida, "¿Pueden pensar las máquinas?".

Turing predijo que habría gran número de opositores a la proposición de que una máquina pudiera pensar por si sola, tanto es así que en el mismo artículo la respondió él mismo. Propuso una prueba en forma de juego con el fin de intentar esclarecer la pregunta y le bautizó con el nombre de "juego de imitación". Consiste en una persona conversando con un interlocutor al que no ve y al mismo tiempo la persona debe tratar de adivinar si está hablando con otro ser humano o una máquina. Si dicha máquina pudiera hacernos creer que estamos hablando con una persona, la teoría de que un ordenador es inteligente podría sustentarse.

Sin embargo, el nacimiento oficial de la Inteligencia Artificial se considera que fue en el año 1956 en la ciudad universitaria de Hannover (EE.UU.) donde se sitúa el Dartmouth College. En dicha universidad se celebró una conferencia a la que acudieron alrededor de una docena de científicos de diferentes campos tales como neurología, matemáticas, psicología e ingeniería eléctrica. La causa que motivó la conexión de un grupo tan diverso fue su herramienta de trabajo, el ordenador. Todos ellos usaban el ordenador de una forma parecida, intentaban simular diferentes aspectos de la inteligencia humana.

Una nueva rama de la informática y la ciencia germinó en aquella conferencia, combinándose diferentes áreas de investigación en un campo unificado. No fueron capaces de llegar a un acuerdo para bautizar a esta nueva rama, sin embargo, el nombre sugerido por John McCarthy fue "Inteligencia Artificial". En la Figura A2- 3 se muestra a los asistentes de la conferencia.

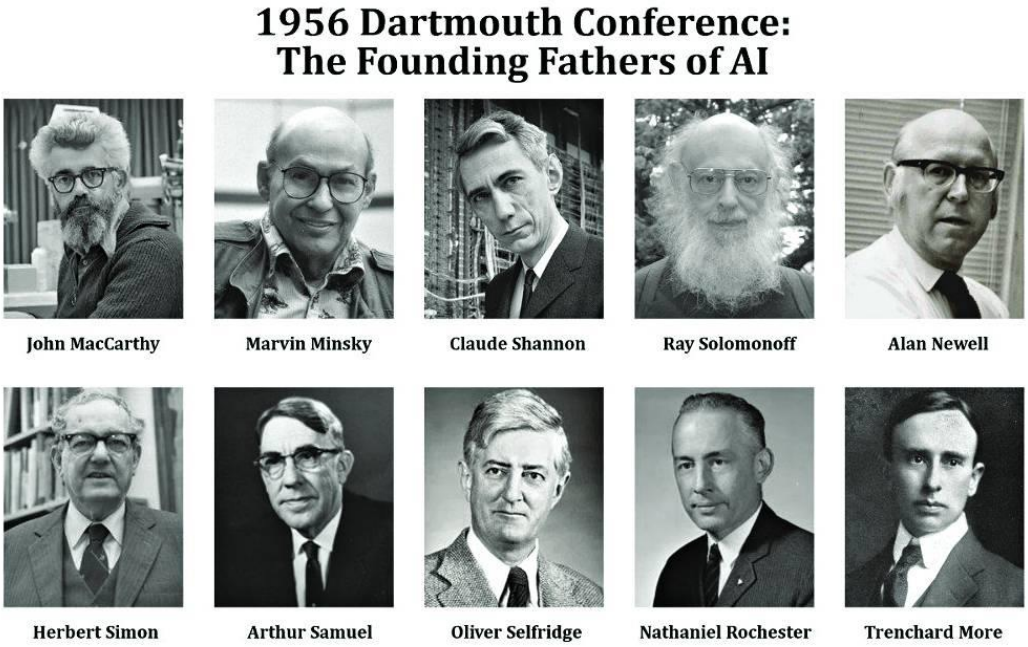
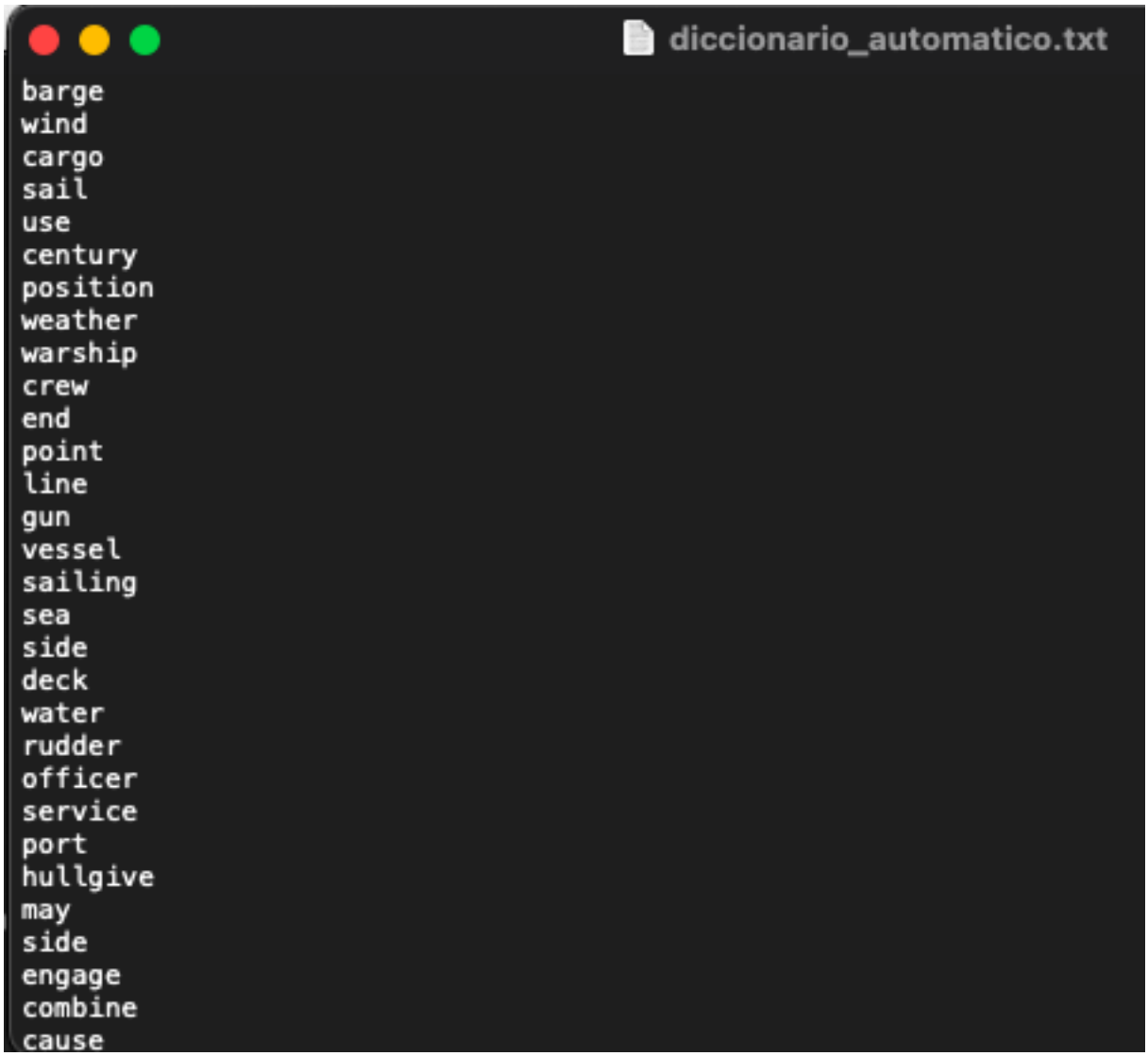


Figura A2- 3 Asistentes Conferencia Dartmouth 1956 (extraída de [62])

ANEXO III: EXTENSIÓN DEL DICCIONARIO LÉXICO

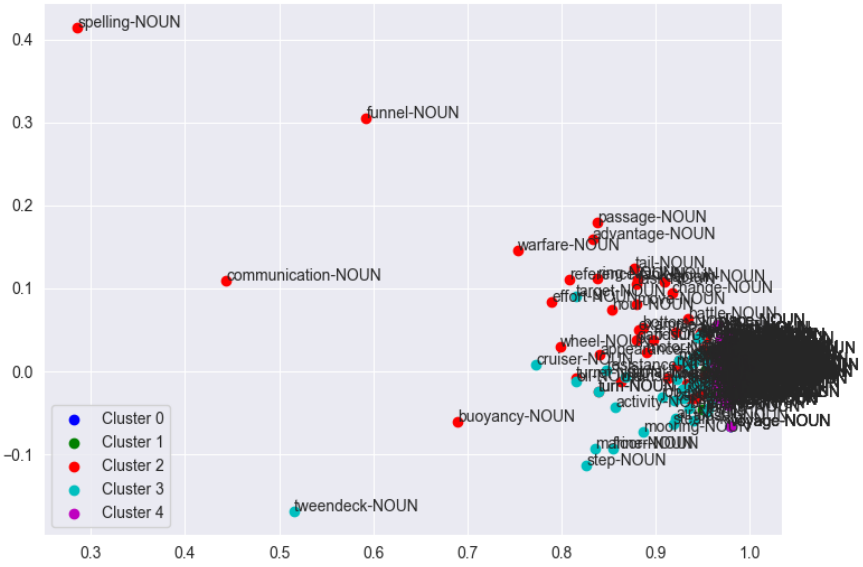
A continuación, se adjunta una imagen de una fracción del diccionario léxico obtenido para mostrar algunos de los términos que el programa ha sido capaz de extraer.

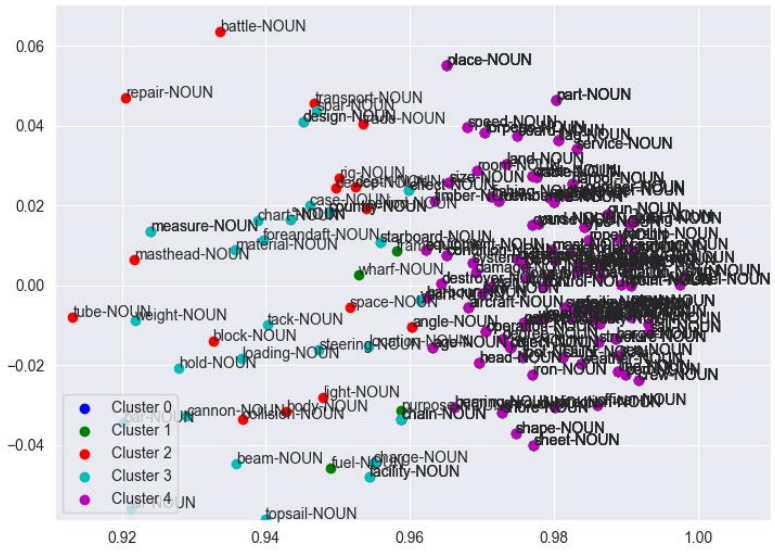


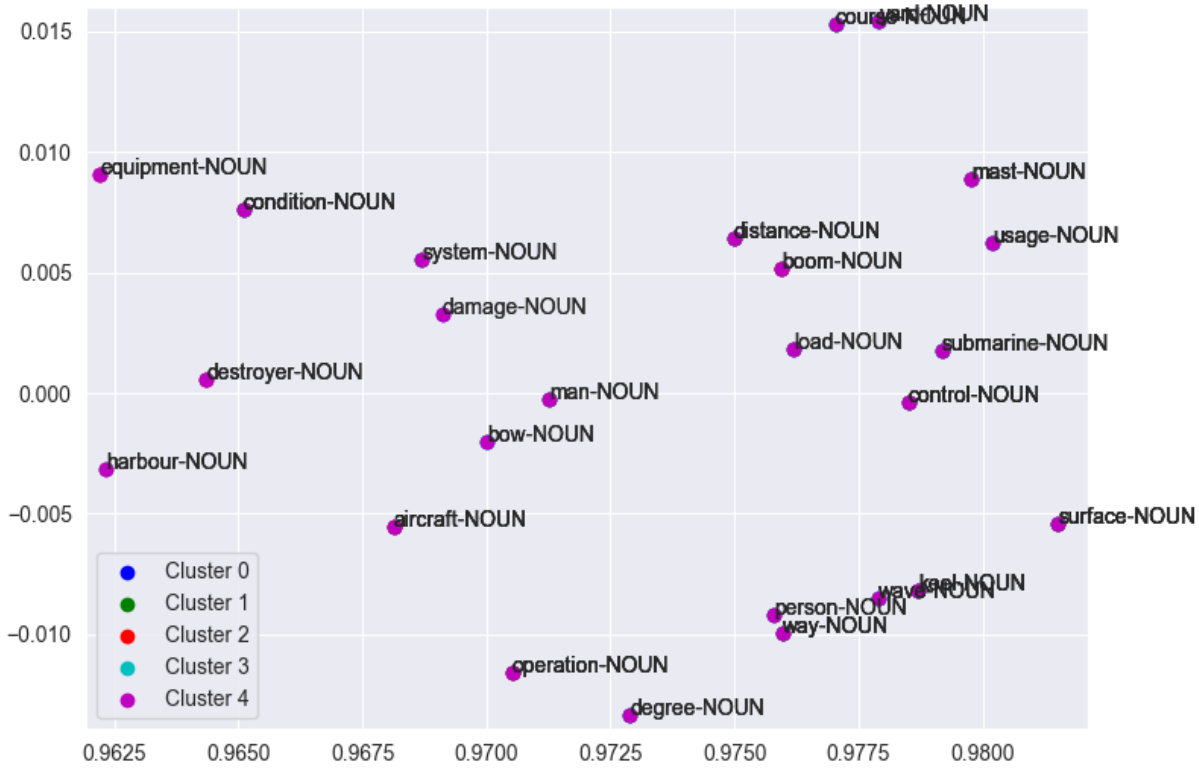
ANEXO IV: CLUSTERING DE TÉRMINOS NÁUTICOS

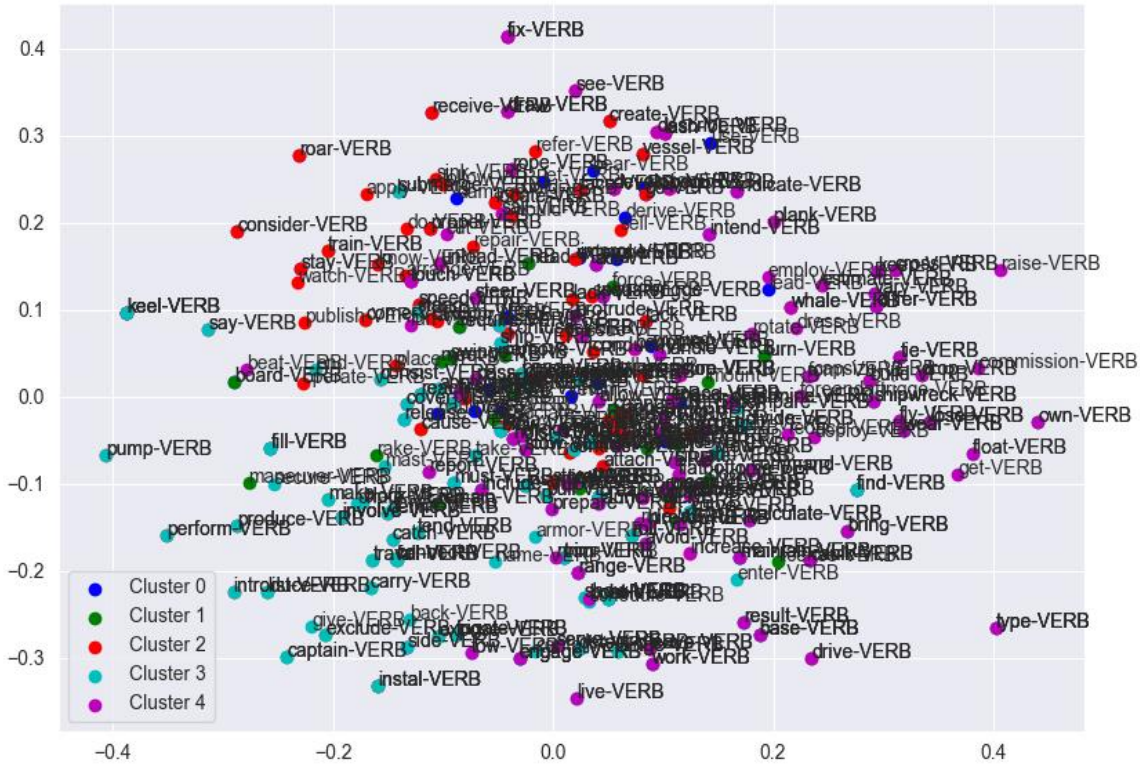
De forma breve, el clustering consiste en agrupar de manera automática mediante algún algoritmo de aprendizaje automático los datos que un programa procesa. Mediante código se puede establecer el número de clusters o grupos que queremos que se creen, por ejemplo, si al algoritmo le arrojamos un conjunto de datos de población y establecemos 2 clusters, lo más probable es que el programa haga un grupo para "hombres" y otro grupo para "mujeres". Si establecemos 4 clusters, podría crear las mismas categorías anteriores más 2 extra, que puedes ser "niños" y "niñas", por ejemplo.

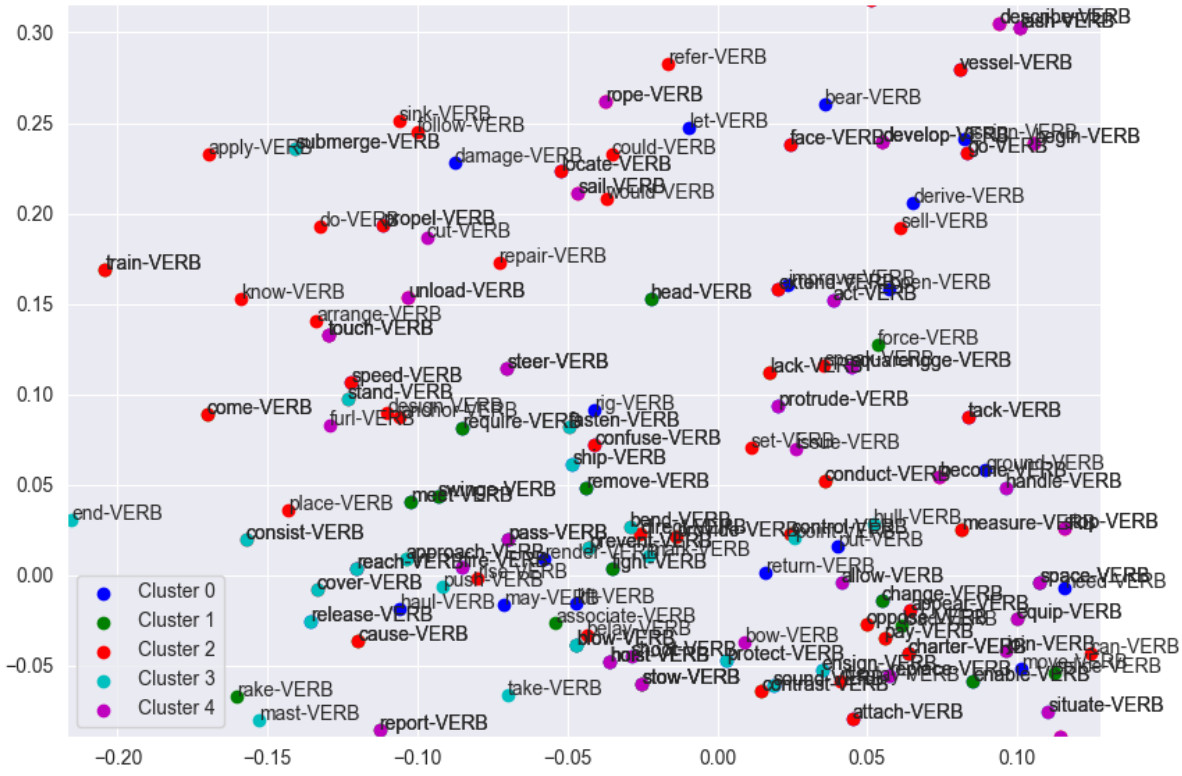
A continuación, se adjuntan los clustering obtenidos con el algoritmo usado para crear el diccionario de léxico náutico. Se han establecido 5 clusters para sustantivos y otros 5 para verbos.

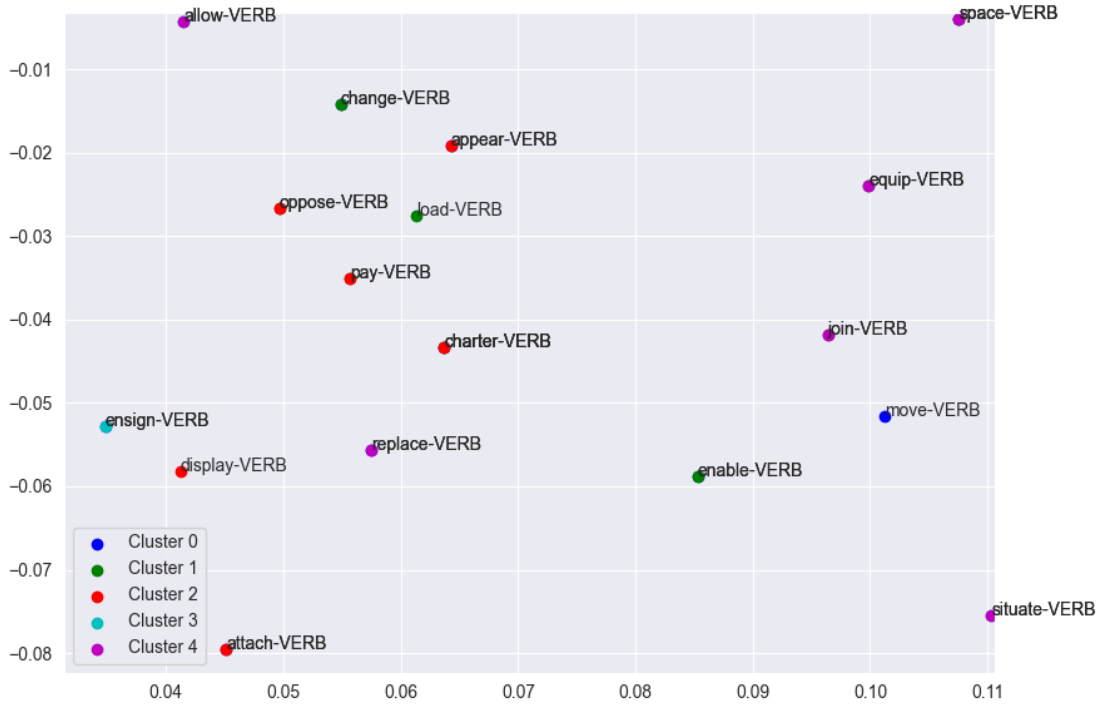





0.015
0.010
0.005
0.000
−0.005
−0.010
0.9625
0.9650
0.9675
0.9700
0.9725
0.9750
0.9775
0.9800
equipment-NOUN
condition-NOUN
system-NOUN
damage-NOUN
destroyer-NOUN
harbour-NOUN
aircraft-NOUN
operation-NOUN
degree-NOUN
man-NOUN
bow-NOUN
person-NOUN
way-NOUN
mast-NOUN
distance-NOUN
usage-NOUN
boom-NOUN
load-NOUN
submarine-NOUN
control-NOUN
surface-NOUN
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4


0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.2
0.0
0.2
0.4
fix-VERB
see-VERB
create-VERB
roar-VERB
refer-VERB
vessel-VERB
consider-VERB
train-VERB
repair-VERB
plank-VERB
intend-VERB
raise-VERB
keel-VERB
say-VERB
whale-VERB
pump-VERB
fill-VERB
float-VERB
get-VERB
find-VERB
perform-VERB
bring-VERB
carry-VERB
enter-VERB
result-VERB
type-VERB
drive-VERB
captain-VERB
instal-VERB
live-VERB
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4


refer-VERB
vessel-VERB
rope-VERB
bear-VERB
let-VERB
sink-VERB
follow-VERB
submerge-VERB
apply-VERB
damage-VERB
could-VERB
face-VERB
develop-VERB
begin-VERB
go-VERB
locate-VERB
sail-VERB
would-VERB
derive-VERB
sell-VERB
do-VERB
propel-VERB
cut-VERB
repair-VERB
train-VERB
know-VERB
unload-VERB
head-VERB
act-VERB
arrange-VERB
touch-VERB
force-VERB
steer-VERB
speed-VERB
stand-VERB
come-VERB
rig-VERB
protrude-VERB
lack-VERB
furl-VERB
require-VERB
fasten-VERB
confuse-VERB
set-VERB
issue-VERB
ship-VERB
remove-VERB
conduct-VERB
become-VERB
handle-VERB
end-VERB
place-VERB
swing-VERB
consist-VERB
pass-VERB
measure-VERB
approach-VERB
fight-VERB
return-VERB
allow-VERB
cover-VERB
push-VERB
change-VERB
haul-VERB
release-VERB
associate-VERB
equip-VERB
cause-VERB
bow-VERB
pay-VERB
charter-VERB
join-VERB
protect-VERB
hoist-VERB
stow-VERB
enable-VERB
rake-VERB
take-VERB
attach-VERB
situate-VERB
mast-VERB
report-VERB
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10


allow-VERB
space-VERB
change-VERB
appear-VERB
equip-VERB
oppose-VERB
load-VERB
pay-VERB
charter-VERB
join-VERB
move-VERB
ensign-VERB
replace-VERB
display-VERB
enable-VERB
situate-VERB
attach-VERB
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
−0.01
−0.02
−0.03
−0.04
−0.05
−0.06
−0.07
−0.08
0.04
0.05
0.06
0.07
0.08
0.09
0.10
0.11

ANEXO V: CÓDIGO

```
1
2   import pandas as pd
3   import speech_recognition as sr
4   from sklearn.model_selection import train_test_split
5   from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
6   from algoritmos import algoritmoRandomForest, algoritmoPassiveAgressive, algoritmoMNB, algoritmoLR
7   from google_trans_new import google_translator
8   import spacy
9   from spacy.lang.en import English
10  from spacy.lang.es import Spanish
11  from spacy.matcher import PhraseMatcher
12  import numpy as np
13  np.random.seed(1)
14
15  txt_speech=''
16
17
18  print("\n\n#############################################################")
19  print("Sistema Reconocimiento Terminología Náutica - Sergio Montes Velez")
20  print("#############################################################\n")
21  print("Por favor, seleccione un micrófono de la lista con su número")
22
```

Figura A5- 1 Librerías 1

```
24  def switch_mic_input():
25
26      argument = int(input("> Esperando respuesta... "))
27
28      switcher = {}
29      for i, mic_name in enumerate(sr.Microphone.list_microphone_names()):
30          switcher[i] = mic_name;
31
32      if argument not in switcher:
33          print(switcher.get(argument, "\nElección no válida"))
34          switch_mic_input(argument)
35
36      else:
37          print("\nElección válida", switcher.get(argument))
38          return argument
```

Figura A5- 2 Switch para elegir entrada de audio

```
recognizer = sr.Recognizer()
for i, mic_name in enumerate(sr.Microphone.list_microphone_names()):
    print(" #:", i, "Nombre", mic_name)

n_mic = switch_mic_input()

mic=sr.Microphone(device_index=n_mic)


with mic as source:

    print("\n...Esuchando ruido ambiente...")
    recognizer.adjust_for_ambient_noise(source, duration=2)
    print("...Grabando durante 15 segundos...")
    recorded_audio = recognizer.listen(source, timeout=15)
    print("...Grabación finalizada...")

try:
    print("...Reconociendo el audio...\n")
    txt_speech = recognizer.recognize_google(
            recorded_audio,
            language="en-EN"
        )
    print(">>Texto Transcrito: {}\n".format(txt_speech))

except Exception as ex:
    print(ex)


translator = google_translator()
translated = translator.translate(txt_speech, lang_tgt='en')
print(">>Texto en ingles: {}\n".format(translated))
translator2 = google_translator()
translated2 = translator2.translate(txt_speech, lang_tgt='es')
print(">>Texto en español: {}\n".format(translated2))
```

Figura A5- 3 Funciones reconocimiento de voz y traducción

```
df = pd.read_csv("/Users/sergio/Desktop/machine_learning_definitvo/archivos_csv/datasetnautico_en.csv", sep=';')
df = df.set_index("ID")


y = df.target
df.drop("target", axis=1)
df.drop("TIME", axis=1)


X_train, X_test, y_train, y_test = train_test_split(df['TEXT'].astype(str), y, test_size=0.2, random_state=1)


tfidf_vectorizer = TfidfVectorizer(stop_words='english', lowercase=True, max_df=0.7)
tfidf_train = tfidf_vectorizer.fit_transform(X_train)
tfidf_test = tfidf_vectorizer.transform(X_test)


count_vectorizer = CountVectorizer(stop_words='english')
count_train = count_vectorizer.fit_transform(X_train)
count_test = count_vectorizer.transform(X_test)

l=[translated]
lista=tfidf_vectorizer.transform(l)

```

Figura A5- 4 Parámetros entrenamiento algoritmos aprendizaje automático

```
algoritmoLR(tfidf_train, y_train,tfidf_test,y_test,lista)
algoritmoMNB(tfidf_train, y_train,tfidf_test,y_test,lista)
algoritmoPassiveAgressive(tfidf_train, y_train,tfidf_test,y_test,lista)
algoritmoRandomForest(tfidf_train, y_train,tfidf_test,y_test,lista)
```

Figura A5- 5 Llamada a las funciones que contienen los algoritmos

```
print("\n***********************************************")
input("Presione Enter para econtrar términos náuticos....")
def econtrar_terminos_nauticos():

    nlp = spacy.load('en')
    doc = nlp(l[0])
    for token in doc:
        print(token, token.lemma, token.lemma_)

    with open('diccionario_automatico.txt') as f:
        content = f.readlines()

        naval_terms = [x.strip() for x in content]

    # Import the PhraseMatcher and initialize it
    matcher = PhraseMatcher(nlp.vocab, attr='LEMMA')

    # Create pattern Doc objects and add them to the matcher
    # This is the faster version of: [nlp(country) for country in COUNTRIES]
    patterns = list(nlp.pipe(naval_terms))
    matcher.add('naval_terms', None, *patterns)

    # Call the matcher on the test document and print the result
    print('Términos Náuticos encontrados:')
    matches = matcher(doc)
    for match_id, start, end in matches:
        span = doc[start:end]
        print(span.text)

econtrar_terminos_nauticos()
print("***********************************************")
```

Figura A5- 6 Código para encontrar términos náuticos en los mensajes

```
import numpy as np
import matplotlib.pyplot as plt
import itertools
from itertools import product
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support
import sklearn.metrics as metrics
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.ensemble import RandomForestClassifier
```

Figura A5- 7 Librerías 2

```python
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.BrBG):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
        print("**************************************************")
    else:
        print('Confusion matrix, without normalization')
        print("**************************************************")

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Figura A5- 8 Función matriz de confusión

```python
def algoritmoLR(tfidf_train, y_train,tfidf_test,y_test,lista):
    model = LogisticRegression(solver='lbfgs')

    model.fit(tfidf_train, y_train)

    y_predict = model.predict(tfidf_test)

    print('Porcentaje de acierto')
    print(str(round(accuracy_score(y_test, y_predict)*100, 3))+'%')
    print(model.predict(lista))
    cm = metrics.confusion_matrix(y_test, y_predict, labels=['no nautico', 'nautico'])
    print((precision_recall_fscore_support(y_test, y_predict, labels=['nautico'], average='weighted')))
    plot_confusion_matrix(cm, classes=['no nautico', 'nautico'])
    plt.show()
```

Figura A5- 9 Algoritmo Logistic Regression

```
67  def algoritmoMNB(count_train, y_train,count_test,y_test,lista):
68
69      model = MultinomialNB()
70
71      model.fit(count_train, y_train)
72      y_predict = model.predict(count_test)
73
74      print('Porcentaje de acierto')
75      print(str(round(accuracy_score(y_test, y_predict)*100, 3))+'%')
76      print(model.predict(lista))
77      cm = metrics.confusion_matrix(y_test, y_predict, labels=['no nautico', 'nautico'])
78      print((precision_recall_fscore_support(y_test, y_predict, labels=['nautico'], average='weighted')))
79      plot_confusion_matrix(cm, classes=['no nautico', 'nautico'])
80      plt.show()
```

Figura A5- 10 Algoritmo Multinomial Naive Bayes

```
82  def algoritmoPassiveAgressive(tfidf_train, y_train,tfidf_test,y_test,lista):
83      linear_clf = PassiveAggressiveClassifier(max_iter=10000)
84
85      linear_clf.fit(tfidf_train, y_train)
86      pred = linear_clf.predict(tfidf_test)
87
88      print('Porcentaje de acierto')
89      print(str(round(accuracy_score(y_test, pred)*100, 3))+'%')
90      print(linear_clf.predict(lista))
91      cm = metrics.confusion_matrix(y_test, pred, labels=['no nautico', 'nautico'])
92      print((precision_recall_fscore_support(y_test, pred, labels=['nautico'], average='weighted')))
93      plot_confusion_matrix(cm, classes=['no nautico', 'nautico'])
94      plt.show()
```

Figura A5- 11 Algoritmo Passive Agressive Classifier

```
96  def algoritmoRandomForest(tfidf_train, y_train,tfidf_test,y_test,lista):
97
98      rf = RandomForestClassifier()
99
100     rf.fit(tfidf_train,y_train)
101     y_pred = rf.predict(tfidf_test)
102
103     print('Porcentaje de acierto')
104
105     print(str(round(accuracy_score(y_test, y_pred)*100, 3))+'%')
106     print(rf.predict(lista))
107     cm = metrics.confusion_matrix(y_test, y_pred, labels=['no nautico', 'nautico'])
108     print(precision_recall_fscore_support(y_test, y_pred, labels=['nautico'], average='weighted'))
109     plot_confusion_matrix(cm, classes=['no nautico', 'nautico'])
110     plt.show()
```

Figura A5- 12 Algoritmo Random Forest Classifier

```python
import re, string
import pandas as pd
import numpy as np
from time import time
import re, itertools, random
from collections import defaultdict
import spacy
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.decomposition import TruncatedSVD
from nltk.corpus import stopwords
STOPWORDS = set(stopwords.words('english'))
from gensim.models import Word2Vec
import matplotlib.pyplot as plt
#matplotlib inline
import seaborn as sns
sns.set_style("darkgrid")
from scipy.spatial.distance import cdist
# uncomment for first run, before put on Internet kernel (settings)
#!python -m spacy download en_core_web_sm
#!python -m spacy link en_core_news_md en_md
nlp = spacy.load('en_core_web_sm', disable=['ner', 'parser']) # disabling Named Entity Recognition for speed
```

Figura A5- 13 Librerías 3

```python
def tsne_plot(model, perplexity=10, n_iter=1000):
    "Create TSNE model and plot it"
    labels = []
    tokens = []

    i = 0
    for word in sorted(model.wv.vocab.keys(), reverse=True):
        tokens.append(model[word])
        labels.append(word)
        i += 1
        if i >= 499:
            break

    tsne_model = TSNE(n_components=2, init='pca', random_state=0, perplexity=perplexity, n_iter=n_iter)
    new_values = tsne_model.fit_transform(tokens)

    x = []
    y = []
    for value in new_values:
        x.append(value[0])
        y.append(value[1])

    x_min, x_max = np.min(new_values, 0), np.max(new_values, 0)
    X = (new_values - x_min) / (x_max - x_min)
    shown_images = np.array([[1., 1.]])  # just something big

    plt.figure(figsize=(20, 20))
    for i in range(len(x)):
        dist = np.sum((X[i] - shown_images) ** 2, 1)
        '''if np.min(dist) < 1e-3:
            # don't show points that are too close
            continue'''
        plt.scatter(x[i], y[i])
        plt.annotate(labels[i],
                     xy=(x[i], y[i]),
                     xytext=(3, 2),
                     textcoords='offset points',
                     ha='right',
                     va='bottom')
    plt.show()
```

Figura A5- 14 Función crear modelo TSNE

```python
def clean_text(text):
    '''Make text lowercase, remove text in square brackets, remove punctuation and remove words containing numbers.'''
    if text is None:
        return ''
    tildes = [("á", "a"), ("é", "e"), ("í", "i"), ("ó", "o"), ("ú", "u")]
    text = str(text).replace("nan",'').lower()
    text = re.sub(r'\[.*?\]', '', text)
    text = re.sub(r'[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub(r'\w*\d\w*', '', text)
    # Quitar tildes
    for j in range(len(tildes)):
        #print(text)
        text = re.sub(re.escape(tildes[j][0]), re.escape(tildes[j][1]), text)
    # Remove a sentence if it is only one word long
    if len(text) >= 2:
        return ' '.join(word for word in text.split() if word not in STOPWORDS)
```

Figura A5- 15 Función para preprocesado del glosario de términos náuticos

```python
def lemmatizer(text,tag):
    sent = []
    doc = nlp(text)
    for word in doc:
        word.lemma_ = re.sub(re.escape(" él"), re.escape(""), word.lemma_)
        if word.lemma_ !="":
            #print(word.lemma_+"-"+word.pos_)
            if tag =="NOUN" and word.pos_ == "NOUN":
                sent.append(word.lemma_+"-"+"NOUN")
            if tag =="VERB" and word.pos_ == "VERB":
                sent.append(word.lemma_+"-"+"VERB")
    return " ".join(sent)
```

Figura A5- 16 Función para lematizar el glosario de términos náuticos

```python
df = pd.read_csv("/Users/sergio/Desktop/extraccion_lexico/lexico-en.csv")

df_cleanWord = pd.DataFrame(df.word.apply(lambda x: clean_text(x)))
df_cleanWord = df_cleanWord.dropna()
df_cleanWord = df_cleanWord.reset_index(drop=True)

df_clean = pd.DataFrame(df.description.apply(lambda x: clean_text(x)))
df_clean = df_clean.dropna()
df_clean = df_clean.reset_index(drop=True)

#print("--------------------")
#print(df_cleanWord.head(10))
#print(df_clean.head(10))

print("--------------------")
df_cleanWord["word_lemmatize_noun"] = df_cleanWord.apply(lambda x: lemmatizer(x['word'],"NOUN"), axis=1)
df_clean["text_lemmatize_noun"] = df_clean.apply(lambda x: lemmatizer(x['description'],"NOUN"), axis=1)
df_cleanWord["word_lemmatize_verb"] = df_cleanWord.apply(lambda x: lemmatizer(x['word'],"VERB"), axis=1)
df_clean["text_lemmatize_verb"] = df_clean.apply(lambda x: lemmatizer(x['description'],"VERB"), axis=1)
#print(df_cleanWord.head(10))
#print(df_clean.head(10))

print("--------------------")
df_cleanWord['word_lemmatize_noun_clean'] = df_cleanWord['word_lemmatize_noun'].str.replace('-PRON-', '')
df_cleanWord['word_lemmatize_verb_clean'] = df_cleanWord['word_lemmatize_verb'].str.replace('-PRON-', '')
df_clean['text_lemmatize_noun_clean'] = df_clean['text_lemmatize_noun'].str.replace('-PRON-', '')
df_clean['text_lemmatize_verb_clean'] = df_clean['text_lemmatize_verb'].str.replace('-PRON-', '')
#print(df_cleanWord.head(30))
print(df_clean['text_lemmatize_noun_clean'].head(10))
print(df_clean['text_lemmatize_verb_clean'].head(10))
```

Figura A5- 17 Código para leer y extraer datos del glosario de términos náuticos

```
print("--------SPLIT-----------")
sentencesNoun0=[row.split() for row in df_cleanWord['word_lemmatize_noun_clean']]
sentencesNoun = [row.split() for row in df_clean['text_lemmatize_noun_clean']]
#print(len(sentences))
sentencesNoun=sentencesNoun0+sentencesNoun
#print(sentences[0])
#sentences=sentences.extend(sentences0)
print(len(sentencesNoun))
sentencesNoun_copy = sentencesNoun
sentencesNoun_copy = [val for sublist in sentencesNoun_copy for val in sublist]
#print(sentencesNoun_copy)
#print(sentences0)
#print(sentencesNoun0[0])
#print(sentencesNoun[0])

sentencesVerb0=[row.split() for row in df_cleanWord['word_lemmatize_verb_clean']]
sentencesVerb = [row.split() for row in df_clean['text_lemmatize_verb_clean']]
sentencesVerb=sentencesVerb0+sentencesVerb
print(len(sentencesVerb))
sentencesVerb_copy=sentencesVerb
sentencesVerb_copy = [val for sublist in sentencesVerb_copy for val in sublist]
#print(sentencesVerb0[0])
#print(sentencesVerb[0])
```

Figura A5- 18 Código para juntar sustantivos y verbos

```
print("------WORD FREQUUENCIES-------------")
word_freqNoun = defaultdict(int)
for sent in sentencesNoun:
    for i in sent:
        if i==0:
            word_freqNoun[i] += 10
        else:
            word_freqNoun[i] += 1


print(len(word_freqNoun))

word_freqVerb = defaultdict(int)
for sent in sentencesVerb:
    for i in sent:
        if i==0:
            word_freqVerb[i] += 10
        else:
            word_freqVerb[i] += 1


print(len(word_freqVerb))
print("--------SORTED WORD FREQUENCIES-----------")

print(sorted(word_freqNoun, key=word_freqNoun.get, reverse=True)[:5])
print(sorted(word_freqVerb, key=word_freqVerb.get, reverse=True)[:15])
```

Figura A5- 19 Frecuencia de los términos

```
# min_count: minimum number of occurrences of a word in the corpus to be included in the model.
# window: the maximum distance between the current and predicted word within a sentence.
# size: the dimensionality of the feature vectors
# workers: I know kaggle system is having 4 cores without gpu and 2 with gpu,
print("--------w2v_model----------")
w2v_modelNoun = Word2Vec(min_count=5,
                         window=1,
                         size=100,
                         workers=4)

w2v_modelVerb = Word2Vec(min_count=5,
                         window=1,
                         size=100,
                         workers=4)

# this line of code to prepare the model vocabulary
w2v_modelNoun.build_vocab(sentencesNoun)
w2v_modelVerb.build_vocab(sentencesVerb)
# train word vectors
w2v_modelNoun.train(sentencesNoun, total_examples=w2v_modelNoun.corpus_count, epochs=w2v_modelNoun.epochs)
w2v_modelVerb.train(sentencesNoun, total_examples=w2v_modelVerb.corpus_count, epochs=w2v_modelVerb.epochs)
print(len(w2v_modelNoun.wv.vocab))
print(len(w2v_modelVerb.wv.vocab))
#print(w2v_modelNoun.wv.vocab)

# As we do not plan to train the model any further,
# we are calling init_sims(), which will make the model much more memory-efficient
w2v_modelNoun.init_sims(replace=True)
w2v_modelVerb.init_sims(replace=True)
```

Figura A5- 20 Parámetros para crear vocabulario y entrenar algoritmo W2V

```
with open('/Users/sergio/Desktop/machine_learning_definitvo/diccionario_automatico.txt', 'w') as f:
    #word_list=[]
    #word_list.append(' '.join([i for i in df_clean['text_lemmatize_noun_clean']]).split())
    #word_list.append(' '.join([i for i in df_cleanWord['word_lemmatize_noun_clean']]).split())
    #word_list = [val for sublist in word_list for val in sublist]
    #word_list = list(set(word_list))
    word_list_wout_dash = []
    for x in sentencesNoun_copy:
        #word_list_wout_dash.append(x.split('-')[0])
        word_list_wout_dash.append(w2v_modelNoun.wv.most_similar(positive=['ship-NOUN', 'boat-NOUN'], topn=20))
        word_list_wout_dash.append(w2v_modelNoun.wv.most_similar(positive=['navigation-NOUN'], topn=20))
        word_list_wout_dash.append(w2v_modelNoun.wv.most_similar(positive=['crew-NOUN'], topn=20))
        word_list_wout_dash.append(w2v_modelNoun.wv.most_similar(positive=['anchor-NOUN'], topn=20))
        word_list_wout_dash.append(w2v_modelNoun.wv.most_similar(positive=['port-NOUN'], topn=20))


    word_list_wout_dash= [val[0] for sublist in word_list_wout_dash for val in sublist]
    #print(word_list_wout_dash)
    word_list_wout_dash = list(set(word_list_wout_dash))
    word_list_wout_dash2 = []
    for word in word_list_wout_dash:
        word_list_wout_dash2.append(word.split('-')[0])
    f.write('\n'.join(word_list_wout_dash2))
```

Figura A5- 21 Introducir sustantivos en diccionario de léxico náutico

```
255     word_list_wout_dash = []
256     for x in sentencesVerb_copy:
257     #     #word_list_wout_dash.append(x.split('-')[0])
258         word_list_wout_dash.append(w2v_modelVerb.wv.most_similar(positive=['sail-VERB'], topn=20))
259         word_list_wout_dash.append(w2v_modelVerb.wv.most_similar(positive=['sink-VERB'], topn=20))
260         word_list_wout_dash.append(w2v_modelVerb.wv.most_similar(positive=['anchor-VERB'], topn=20))
261         word_list_wout_dash.append(w2v_modelVerb.wv.most_similar(positive=['dock-VERB'], topn=20))
262 
263     word_list_wout_dash = [val[0] for sublist in word_list_wout_dash for val in sublist]
264     word_list_wout_dash = list(set(word_list_wout_dash))
265     word_list_wout_dash2 = []
266     for word in word_list_wout_dash:
267         word_list_wout_dash2.append(word.split('-')[0])
268     f.write('\n'.join(word_list_wout_dash2))
```

Figura A5- 22 Introducir verbos en diccionario de léxico náutico

```
270 print("--------PLOT----------")
271 # Use t-SNE to represent high-dimensional data
272 # and the underlying relationships between vectors in a lower-dimensional space.
273 
274 #tsne_plot(w2v_modelNoun,40,5000)
275 tsne_plot(w2v_modelVerb,40,5000)
```

Figura A5- 23 Mostrar modelo TSNE

```
print("---CLUSTERING----------")
# First get the embeddings into a matrix
embedding_size=100
embeddingsNoun = np.zeros((len(w2v_modelNoun.wv.index2word), embedding_size))
for i in range(0, len(w2v_modelNoun.wv.index2word)):
    w = w2v_modelNoun.wv.index2word[i]
    embeddingsNoun[i] = w2v_modelNoun.wv[w]

embeddingsVerb = np.zeros((len(w2v_modelVerb.wv.index2word), embedding_size))
for i in range(0, len(w2v_modelVerb.wv.index2word)):
    w = w2v_modelVerb.wv.index2word[i]
    embeddingsVerb[i] = w2v_modelVerb.wv[w]

svd = TruncatedSVD(n_components=2, algorithm='randomized', n_iter=500, random_state=101)
embeddings_2d_projectionNoun = svd.fit_transform(embeddingsNoun)
embeddings_2d_projectionVerb = svd.fit_transform(embeddingsVerb)

# Train a K-means cluster model with 6 clusters
n_clusters = 5
embedding_cluster_modelNoun = KMeans(n_clusters=n_clusters, random_state=0).fit(embeddingsNoun)
embedding_cluster_modelVerb = KMeans(n_clusters=n_clusters, random_state=0).fit(embeddingsVerb)

centroid_embedding_nearest_wordsNoun = []
for centroid_embedding in embedding_cluster_modelNoun.cluster_centers_:
    centroid_embedding_nearest_wordsNoun.append(
        np.argsort([i[0] for i in cdist(embeddingsNoun, np.array([centroid_embedding]), "euclidean")])[0:100]
    )

centroid_embedding_nearest_wordsVerb = []
for centroid_embedding in embedding_cluster_modelVerb.cluster_centers_:
    centroid_embedding_nearest_wordsVerb.append(
        np.argsort([i[0] for i in cdist(embeddingsVerb, np.array([centroid_embedding]), "euclidean")])[0:100]
    )

plt.figure(figsize=(10,10))
colors = itertools.cycle(["b","g","r","c","m","y","k","w"])
c = 0
for word_indices in centroid_embedding_nearest_wordsNoun:
    clr = next(colors)
    plt.scatter(
        embeddings_2d_projectionNoun[word_indices,0],
        embeddings_2d_projectionNoun[word_indices,1],
        color=clr,
        label="Cluster " + str(c)
    )
    for ix in word_indices:
        x, y = embeddings_2d_projectionNoun[ix,:]
        plt.annotate(w2v_modelNoun.wv.index2word[ix], (x, y))
    c+=1
plt.legend(loc='lower left')
plt.show()
plt.figure(figsize=(10,10))
colors = itertools.cycle(["b","g","r","c","m","y","k","w"])
c = 0
for word_indices in centroid_embedding_nearest_wordsVerb:
    clr = next(colors)
    plt.scatter(
        embeddings_2d_projectionVerb[word_indices,0],
        embeddings_2d_projectionVerb[word_indices,1],
        color=clr,
        label="Cluster " + str(c)
    )
    for ix in word_indices:
        x, y = embeddings_2d_projectionVerb[ix,:]
        plt.annotate(w2v_modelVerb.wv.index2word[ix], (x, y))
    c+=1
plt.legend(loc='lower left')
plt.show()
```

Figura A5- 24 Clustering