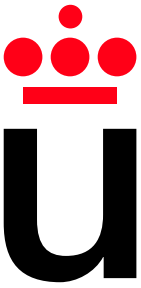

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Máster Ciberseguridad y Privacidad

Curso 2021-2022

Trabajo Fin de Máster

IA/MACHINE LEARNING APLICADA A CIBERSEGURIDAD

Autor: Carlos Luzzatti Vázquez
Tutor: Raul Martín Santamaria

Agradecimientos

Agradecimientos a mi familia y a todos los profesores que he tenido por fomentar mi curiosidad y mi desarrollo personal. También a Carmen Rubio por estar siempre a mi lado y apoyarme en todo momento.

Resumen

¿Cómo la IA y las aplicaciones de Machine Learning pueden ser útiles? Las IA actualmente se encuentran integradas en casi todos los entornos gracias a las distintas soluciones que dan en las diversas áreas de la sociedad [1], por lo cual la ciberseguridad no es una excepción, ya que las amenazas de ciberseguridad informáticas ocasiones millones en daños y cuanto más avanza la informática más aumentan esos daños.

Los algoritmos de Machine Learning o aprendizaje realizan tareas de manera inteligente sin que sea necesario que un humano las deba programar para esa tarea, esto lo realiza una aplicación o un subconjunto de aplicaciones dentro del programa que realiza un modelo matemático dependiendo del modelo que se utilice que proviene de datos de entrenamiento para poder realizar un modelo con el que predecir los resultados cuando se introduzcan nuevos datos automáticos

La IA puede utilizarse para detección de amenazas, contraseñas y autentificación, prevención y detección de phishing, entre otras.

En nuestro trabajo nos hemos centrado en la automatización de la detección de *spam* a través de aproximaciones generadas por IA

El objetivo principal es crear un clasificador de correos de spam utilizando técnicas de aprendizaje *Machine Learning*, utilizaremos varias técnicas de *Machine Learning* Machine Learning para predecir mediante un conjunto de datos o *dataset* cuantos correos son spam y cuantos son correos seguros para así evitar posibles fraudes o infecciones del sistema y realizar la comparativa de las distintas técnicas para comparar el rendimiento de estas dos técnicas, realizaremos esta acción mediante dos maneras diferentes para poder comprobar la exactitud que empleamos en ambos modelos, en uno de ellos utilizaremos un algoritmo que previamente compararemos con los más utilizados y lo elegiremos en razón a su rendimiento y la otra manera será mediante una red neuronal para ver que método puede resultar más efectivo para esta acción, después de este proceso generaremos un fichero con los datos del *spam*.

En este caso realizaremos una comparativa de los modelos más usados, comparando el rendimiento para resolver este problema con el modelo que mejor resultados nos devuelva.

Para comparar los algoritmos más utilizados lo haremos calculando su exactitud y su tiempo de ejecución y una vez analizados podremos valorar cuál de ellos es el más conveniente, nosotros en este caso después de analizarlo hemos optado por la opción *ComplementNB* por su alto rendimiento y eficiencia.

Para la red neuronal hemos optado por utilizar el algoritmo de regresión logística, que es un aprendizaje supervisado, pero contrariamente a su nombre, no es una regresión, sino un método de clasificación. Asume que los datos pueden ser clasificados (separados) por una línea o un plano de dimensiones.

Los datos del *dataset* debemos pre procesarlos para eliminar características especiales y tokenizar las palabras para convertir esos datos en datos que el proceso soporte y pueda realizar las predicciones.

Para realizar este proceso necesitaremos realizar varias tareas dentro de nuestro programa, necesitaremos importarnos las librerías necesarias para realizar todas las acciones, preparar los datos de entrenamiento, seleccionar un modelo o crearlo, entrenar el modelo con los datos de entrenamiento y usar el modelo para obtener las predicciones. Utilizaremos sets de datos de *Kaggle* para poder analizarlos y realizar las predicciones, porque nos ofrece distintos juegos de datos para diferentes propósitos.

Hemos optado por utilizar varios modelos diferentes, principalmente uno basado en redes neuronales, hemos realizado un entrenamiento con los modelos predefinidos *Liblinear*, *SGD*, *SGD con elastic net penalty*,*threshold* y basados en *Naïve Bayes* para escoger el más adecuado que en este caso el modelo que elegimos para tratar los datos finales sería el algoritmo *ComplementNB* basado en *Naïve Bayes* para comprobar cuál es la mejor opción para la clasificación de texto, ya que para algunas tareas puede dar mejores resultados una opción u otra y siempre es favorable analizar cuál es más efectiva para un mejor rendimiento, por lo tanto, en nuestros resultados nos interesaría la comparativa realizada con los distintos métodos para así obtener el mejor resultado posible.

Índice de contenidos

1 Introducción y contexto

Machine Learning o aprendizaje automático se está convirtiendo en una herramienta muy potente para garantizar la seguridad de un sistema, todo esto se debe a la capacidad de aprender a buscar de forma automática dentro de una gigantesca cantidad de datos los modelos de comportamiento y así poder detectar cualquier dato anómalo que sea indicativo de una amenaza para poder detectar las amenazas de forma automática, así se puede actuar inmediatamente implementando las soluciones de seguridad informática, el motor de la máquina en este caso sería el algoritmo que aprende revisando los datos y aprendiendo a predecir los futuros comportamientos, esto implica que los sistemas se van a ir mejorando de forma independiente a lo largo del tiempo que permanezca el algoritmo trabajando sin necesidad de intervención humana.

Machine Learning es un algoritmo que puede aprender entrenando con datos que ha recibido, estos algoritmos se pueden definir con redes neuronales definiendo las neuronas y las capas o bien podemos importar un modelo desde *Keras* para facilitarlo, también es posible crear una red neuronal he importarla en formato JSON.

El problema que abordaremos en este TFM es la predicción de *spam* a través de IA, se realizara con el mismo conjunto de datos distintas predicciones con varios modelos y se valorara el que resulte tener más eficacia en comparación con los demás, ya que debido a la cantidad de correos que recibimos y a la tendencia a la digitalización, cada vez vamos a recibir más correos tanto de anuncios como pueden tratarse de documentos importantes a escala de usuario o en las grandes empresas que recibir correos spam puede ser un problema importante, por eso la predicción de spam a través de una inteligencia artificial es sumamente importante, ya que nos evitaría recibir correos basura.

1.1. Que es una red neuronal

Las neuronas artificiales intentan imitar el comportamiento de una neurona cerebral, se compone de unas ramificaciones y un núcleo o nodo, existirán ramificaciones de entrada al nodo que se procesara y genera una información de salida que se trasmite por las ramificaciones de salida hacia otras neuronas.

Para este experimento vamos a utilizar una red neuronal que va a recibir una serie de datos

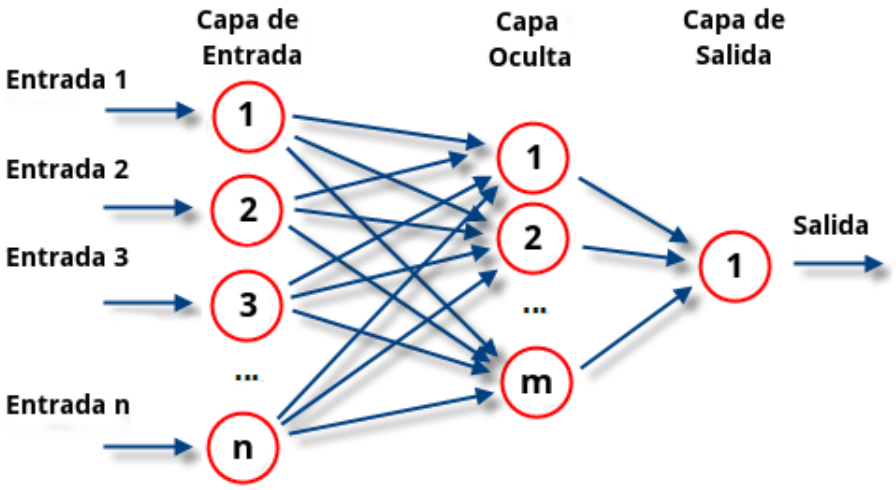

Figura 1.1: Funcionamiento red neuronal [3]

con los cuales va a entrenar para así aprender que correos son *spam* mientras recibe nueva información para relacionarla entre sí y de esta manera, puede aprender de manera similar que un cerebro humano.

Con este proceso se crean patrones de comportamiento que se obtienen a partir del análisis de un conjunto de datos para que las máquinas aprendan solas a base de los anteriores resultados [2], por lo que a mayor volumen de datos disponibles nuestro algoritmo sería más complejo.

1.2. Capas y neuronas

Las capas se pueden definir como los diferentes niveles de lo que dispone la red neuronal, a su vez estos se componen de uno o varios conjuntos de neuronas de las que sus entradas dependen de una capa anterior, a menos que sea la primera capa que depende de los datos de entrada, de la primera capa las salidas serían la entrada de la próxima capa, la neurona se puede considerar la unidad funcional de los modelos de redes. En cada neurona pueden ocurrir dos tipos de operaciones:La suma ponderada de las entradas y también se aplica la función de activación [4].

La primera capa se conoce como la capa de entrada que deberá tener tantas neuronas como los tipos de datos que queremos analizar, en nuestro caso como queremos analizar la columna donde muestra el texto del mensaje.

Las capas ocultas recibe los valores de la capa de entrada, que son ponderados por los pesos, las unidades se conectan con fuerzas de conexión variables (o ponderaciones), dentro de estas capas se define el numero de neuronas que necesitamos, cuanto más neuronas tenga cada capa más tardara el modelo en entrenar, pero aumenta la precisión por eso el número de neuronas de cada capa se puede ajustar a base de prueba y error .

1.3. Entrenamiento del modelo

El proceso de entrenamiento de una red neuronal se realiza ajustando el valor de los pesos y bías para que así las predicciones que se vayan a generar, reduzca al máximo el menor error posible. Por este motivo, el modelo de red neuronal es capaz de identificar qué predictores tienen mayor eficacia y la forma en la que están relacionados entre ellos y con las variables de respuesta [5].

Alcanzar una forma de óptima de implementarla ha necesitado la combinación de varios métodos matemáticos, más concretamente, el algoritmo de retropropagación *(backpropagation)* y la optimización por descenso de gradiente *(gradient descent)* [5].

Backpropagation es el algoritmo más importante dentro de una red neuronal, ya que permite calcular la influencia que tiene cada uno de los pesos y las bías de la red en sus predicciones. Para lograrlo, se utiliza la regla de la cadena *(chain rule)* para calcular el gradiente *(gradient)*, que es el vector creado por las derivadas parciales de una función [5].

En el caso de las redes neuronales, la derivada parcial del error respecto a una variable, ya sea peso o bías mide cuanta carga ha tenido esa variable en el error cometido [5]. Por esto, se puede reconocer cuáles son los pesos de la red hay que modificar para poder mejorar los resultados. El siguiente paso que hace falta, es decidir cuánto y cómo modificarlos (optimización).

Descenso de gradiente (gradient descent) es un algoritmo de optimización iterativo de primer orden que permite reducir una función haciendo modificaciones de sus parámetros en la dirección del valor negativo de su gradiente. En lo que respecta a las redes neuronales, el descenso de gradiente permite ir modificando los pesos y bías del modelo para reducir así reducir al máximo el error [5].

Dependiendo del número iteraciones es posible que pueda llegar a ser computacionalmente costoso calcular el error de modelo en todas las observaciones de entrenamiento, para esto existe una alternativa para gradient descent que se llama gradiente estocástico *(stochastic gradient descent, SGD)*. Este método divide el conjunto de entrenamiento en lotes (pueden ser minibatch o batch) y actualizar las variables de la red con cada uno. De esta manera, no es necesaria evaluar cada observación para modificar los parámetros, se pueden ir actualizando de forma sucesiva.

Una ronda completa de iteraciones sobre todos los batch se llama época. El número de épocas con las que se entrena una red equivale al número de veces que la red ve cada ejemplo de entrenamiento [6].

La mayoría de algoritmos de aprendizaje automático también necesitan entrenamiento, es decir, estos algoritmos se utilizan en los modelos, ya que pueden utilizar lo aprendido en el pasado para realizar predicciones, pero para que esto sea posible es necesario entrenarlo con modelos de datos donde ya se conozca el resultado para que al recibir nuevos datos en los que no conozca el resultado pueda capaz de predecirlo y ajustar el modelo en consecuencia.

1.4. Inteligencia artificial

En estos momentos estamos viviendo el mayor crecimiento dentro de la tecnología de inteligencia artificial, ya que actualmente se puede encontrar este tipo de tecnología en asistentes de dispositivos (Alexa), en nuestro motor de búsqueda de internet, el acceso a edificios como puede ser monitorización de la temperatura corporal por cámaras, en los algoritmos de los navegadores de búsqueda y de optimización, etc.

La inteligencia artificial nacida en 1956 sufrió un desarrollo complicado con periodos de

grandes avances seguidos por momentos de poco crecimiento. Actualmente, vivimos un increíble auge en el crecimiento de esta tecnología debido a que los equipos son cada vez más potentes y cada vez son más necesarios en nuestro día a día, por lo que pueden procesar una gran cantidad de datos *('Big data')* en menor tiempo, aumento de los procesamientos gráficos(GPU) porque cada vez son más potentes y el procesamiento tensorial o TPU desarrollado por Google precisamente para las inteligencias artificiales.

Otras de las aplicaciones que nos resulta bastante interesante en el ámbito de la seguridad es la Biométrica, ya que permite interacciones entre humanos y máquinas, sean de una manera más fácil y natural, en nuestro caso se puede incrementar la dificultad de acceso a datos o incluso a la propia instalación, esta opción es bastante interesante ya que sería posible bloquear un dispositivo o restringir el acceso a ellos mismos si no es la persona que tiene los permisos para ello, pudiendo reducir la probabilidad de los ciberataques.

1.5. Machine Learning

Dentro de las inteligencias artificiales nos vamos a centrar en Machine Learning o también llamado aprendizaje automático, es una disciplina de las inteligencias artificiales que desarrolla técnicas para permitir a un programa a través de procesar datos obtener experiencia, en este caso a través de analizar un documento clasificado por categorías pueda aprender. Esto lo realiza utilizando una serie de algoritmos, clasificación, análisis y recopilación de la información, las plataformas que utilizan Machine Learning tienen muchas aplicaciones dentro de varios ámbitos como puede ser la predicción de resultados, personalización de la experiencia de interacción entre un usuario y un sistema, personalización de la publicidad enviada a un usuario, etc.

El desarrollo cloud nativo es una herramienta indispensable cuando una empresa quiere estar dirigida por datos, ya así aprovechan las ventajas de la computación en la nube. El modelo tecnológico que se va a implantar en sistemas de Machine Learning está basado en una arquitectura cloud híbrida impulsada por MLOps y Kubernetes para que así la tecnología cloud puede simplificar las implementaciones de Machine Learning. Esto implica que al facilitar las implementaciones ML también está aumentando la demanda de científicos de datos hacia áreas que necesitan sistemas de aprendizaje automático. Este avance hacia las soluciones en la nube con modelos auto ajustables de Machine Learning puede desplazar a los modelos de IA hacia la ciencia de la decisión [7].

1.5.1. Tipos de sistemas Machine Learning

Para clasificar los distintos tipos de Machine Learning podemos tipificarlos en función si necesitan o no que los modelos sean entrenados mediante supervisión humana, la habilidad de aprender o no de forma incremental a medida que en el sistema se obtenga nuevos datos o si se basa en un modelo de aprendizaje basado en instancias o modelos.

- Aprendizaje supervisado: el objetivo es crear una función que pueda predecir el valor correspondiente a los datos de entrada dados, teniendo un conjunto de datos anteriormente clasificados y etiquetados, la salida puede resultar ser un número o también una etiqueta de clase.

- Aprendizaje no supervisado: El programa recibe datos de entrenamiento sin categorizar e intenta aprender por si solo, los algoritmos utilizados hacen uso de la técnica *(clustering)*, ordenando los datos que hemos obtenido por grupos que tengan características relacionadas entre ellas. Se utilizan en entornos donde se quiera detectar anomalías utilizando

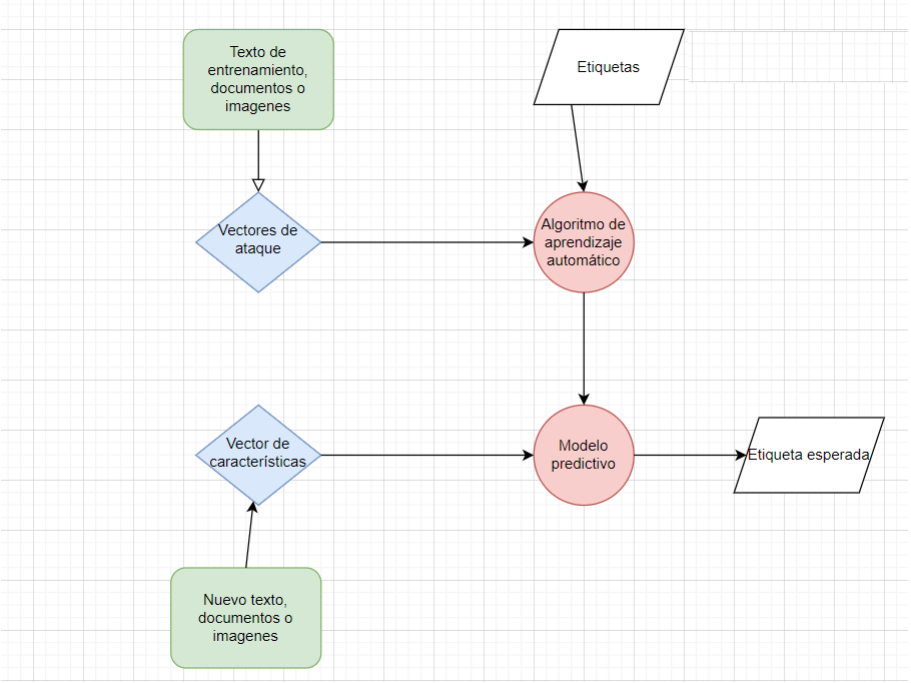

Figura 1.2: Modelo de aprendizaje supervisado

los datos para entrenar y en cuanto se presente un dato diferente o extraño se detecta la anomalía y el sistema alerta de ello.

- Aprendizaje por refuerzo: El sistema informático va a interactuar con el entorno y mediante una lista de procedimientos posibles, evaluara cuál es la mejor opción y tomara las decisiones correspondientes, el sistema aprenderá por si solo cuál es la mejor estrategia a seguir según los datos recibidos.

La inteligencia artificial es la clave para el internet de las cosas (IoT) porque gracias a las IA nuestros dispositivos pueden comunicarse entre sí para dar servicios más personalizados a los clientes, también la inteligencia artificial se beneficiara de la computación cuántica, ya que al poder obtener más velocidad de procesamiento los ordenadores pueden trabajar más rápido y realizar más operaciones en menos tiempo por lo que la probabilidad de fallo disminuirá considerablemente.

Palabras clave:

- Python
- Ciberseguridad
- Aprendizaje automático
- Machine Learning
- IA
- Dataset
- Spam
- Clasificación
- ...

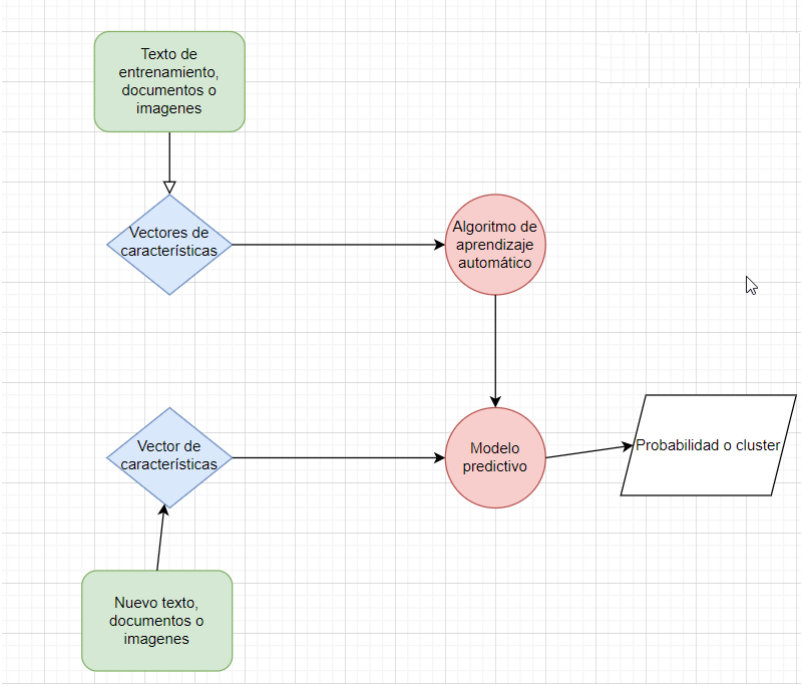

Figura 1.3: Modelo de aprendizaje no supervisado

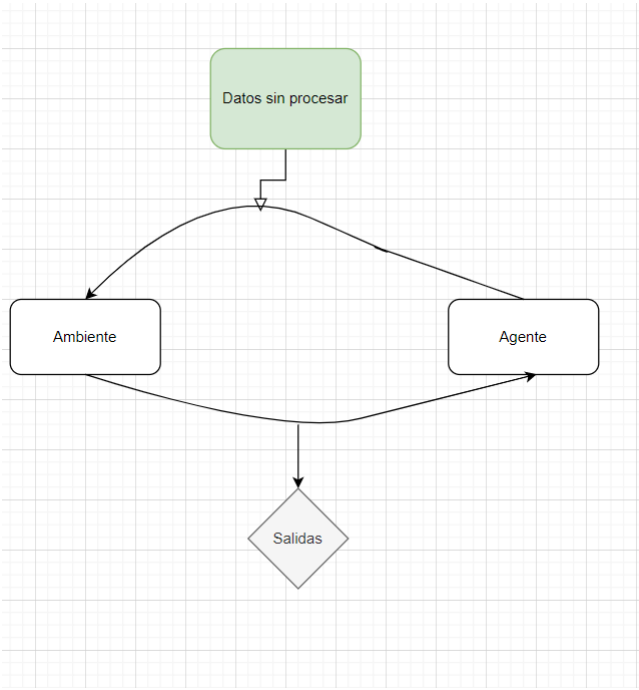

Figura 1.4: Modelo de aprendizaje por refuerzo

2

Objetivos, planificación y metodología

2.1. Objetivos

El objetivo principal es crear un clasificador de correos de spam utilizando varias técnicas de aprendizaje Machine Learning, para realizar nuestro objetivo realizaremos varias tareas (carga y análisis de datos, comparación entre diferentes alternativas existentes, comparar mejor alternativa existente con Machine Learning de redes neuronales), a continuación las detallaremos con más exactitud.

Cargar los datos desde un archivo CSV con los datos a analizar.

Pre procesar los datos de la forma más óptima para un mejor manejo de datos ordenándolo con etiquetas. Obtener información de los modelos predefinidos más usados y que mejor nos convengan para poder realizar una comparación.

Utilizar el entrenamiento del modelo en varios modelos predefinidos para comparar la exactitud y elegir el que mejor nos convenga por su exactitud para poder realizar la comparación frente a los modelos de redes neuronales.

Crear modelo predefinido con los mejores hiperparámetros para eso buscaremos los mejores posibles para el modelo.

Compilar y evaluar el modelo con los valores óptimos para su mejor funcionamiento.

Definir y compilar el modelo de la red neuronal con los valores óptimos para entrenar el modelo con los datos de entrenamiento.

Evaluar el modelo para encontrar los mejores valores.

Comparar de la exactitud de los dos modelos finalmente entrenados para comprobar la eficiencia de los modelos predefinidos contra los modelos basados en redes neuronales, se pretende comparar el rendimiento de estas dos técnicas en la resolución de este problema.

Obtener un CSV con el contenido de los mensajes catalogados como *Spam*.

2.2. Planificación

Dividiremos las opciones de salida en *spam* y *ham* (no *spam*).

Leer los datos del dataset y guardarlos en una variable llamada 'df' en la que eliminaremos los datos innecesarios y pondremos las etiquetas que necesitemos para poder categorizar los datos.

Contaremos el volumen de registros que tenemos y los compararemos con la etiqueta que nos indica si es spam o ham y mostraremos en un gráfico los datos analizados que son de cada tipo. Desde aquí empezaremos con la parte del algoritmo ya importado y para eso dividiremos los datos en matrices en dos conjuntos que serán de prueba y entrenamiento y crearemos una función que analice los datos, mostrando el resultado que nos indicara el nombre del modelo, la exactitud y el tiempo empleado en analizar los datos.

Compararemos el rendimiento de los distintos modelos y seguiremos el siguiente paso con el que mejores resultados arroje para poder compararlo con el modelo de redes neuronales.

Después de realizar la parte anterior utilizaremos el modelo predefinido que mejores resultados haya obtenido.

Buscaremos los mejores hiperparámetros para optimizar nuestro modelo y crearemos nuestro modelo con el mayor rendimiento posible.

Pasaremos los datos de entrenamiento a este modelo y mediremos la exactitud. Crearemos nuestra modelo de red neuronal con las capas que necesitemos.

Compilaremos el modelo ajustando las perdidas, las métricas y el optimizador.

Pre procesaremos los datos para este nuevo modelo de red neuronal.

Entrenamos el modelo con los datos de entrenamiento y sacamos la exactitud.

Comparamos la exactitud del modelo predefinido contra la exactitud del modelo basado en redes neuronales, para así poder demostrar en este caso que por su rendimiento y exactitud los modelos predefinidos son superiores a los modelos de redes neuronales para este tipo de experimento.

Escribimos los registros de los emails que sean Spam en un archivo CSV.

2.3. Metodología

Para el desarrollo del software hemos utilizado la metodología de programación incremental y el desarrollo en espiral.

- Incremental: en este modelo de desarrollo de software se va a desarrollar el programa con elementos del modelo en cascada agregando nuevas funcionalidades, de esta manera obtendremos rápidamente los resultados comparados con el modelo en cascada, el software se puede probar antes de finalizarlo, incluso solo una parte de programa dando bastante flexibilidad a este modelo [8].
- Desarrollo en espiral: Con este modelo de desarrollo obtenemos la combinación de los desarrollos en cascada y el desarrollo incremental, este modelo es la respuesta a las desventajas del desarrollo en cascada, el ciclo de vida se interpreta como una espiral que se repite hasta llegar a la finalización del producto, las fases que se repiten serían análisis, evaluación, desarrollo y planteamiento, las ventajas de usar este modelo serían reducción

de conflictos de software y diseño, se puede obtener *feedback*, se pueden registrar todos los cambios.

- Vamos a utilizar un modelo supervisado a la hora de aplicar nuestro logaritmo para Machine Learning, hablamos de modelos supervisados cuando trabajamos con algoritmos que aprenden con datos previamente organizados y definidos con etiquetas *(labels)*. Cuando dichos modelos tienen la suficiente cantidad de datos, pueden recibir nuevos datos sin la necesidad de clasificarlos con base en los patrones que se han venido registrando durante el entrenamiento. El algoritmo aprende a predecir unas etiquetas conocidas en función a las variables de entrada *(features)*.

  Este modelo se caracteriza por:

  1. Es necesario que una persona humana u otro programa intermedio previamente programado introduzca los datos, los clasifique y los etiquete, por lo que en esta parte no está totalmente automatizado
  2. En la salida los datos que genera el algoritmo son esperados, ya que los datos de entrada han sido etiquetados y clasificados previamente

  El algoritmo admite dos tipos de datos:

  1. Clasificación. Clasifican los objetos en diversas categorías para así predecir los resultados. En este caso nos serviría para predecir si un correo es *spam* o no.
  2. Regresión sirven para predecir un valor de tipo numérico comprendiendo las relaciones entre las variables numéricas que ha recibido, siendo útil para predecir resultados numéricos.

Este tipo de Machine Learning se puede usar para: Predicción de costes de una entidad aseguradora al dar de alta un siniestro, detección de spam, prever que sistemas operativos son más vulnerables a un tipo de malware.

3

Estado del arte

3.1. Definición phishing

Es un tipo de fraude informático mediante correo electrónico muy utilizado en los últimos tiempos tanto a grandes empresas como a pequeños usuarios en los que el medio puede ser el *spam* o correo basura, utilizando ingeniería social para así poder engañar mediante la suplantación de identidad y obtener datos sensibles como usuarios, contraseñas, cuentas bancarias entre otras, esto lo realizaría el atacante, o *pisher* que es el encargado de distribuir los correos electrónicos aprovechando los caracteres especiales que no están en nuestro alfabeto para recrear direcciones webs o de correo falsas que a primera vista lucen auténticas y que al introducir los datos en la página falsa se los devuelve al atacante.

Según indica los informes que indica **APWG**(*Anti-Phishing Working Group*) la situación sobre este primer trimestre: En el primer trimestre de 2022, **APWG** observó 1 025 968 ataques de *phishing* en total. Este fue el peor trimestre para el *phishing* que **APWG** haya observado, y la primera vez que el total trimestral superó el millón [9].

Por lo que los ataques de *phishing* cada vez son más frecuentes, aunque cada vez están más dirigidos a empresas del sector financiero.

La suplantación de la identidad de ejecutivos corporativos en las redes sociales fue un riesgo empresarial observado cada vez más [9].

3.1.1. Tipos de ataques phishing

Los diferentes tipos de ataques basados en *phishing* que se dan mayormente en la actualidad y que están relacionados con el *phishing* por correo electrónico son los siguientes:

- *Phishing* por Correo Electrónico. Es una estafa en la que en atacante envía un correo electrónico a la víctima en la que suplanta a una empresa o servicio de confianza con la intención de obtener datos privados, que realicemos algún tipo de pago o instalar software malicioso en nuestro equipo.

- *Spear phishing.* Este tipo de estafa comienza poniendo como objetivo recopilar información como las redes sociales, ubicación o información web que se pueda recopilar de una empresa, para luego dirigirse a individuos específicos por correo electrónico con datos verídicos para que así la víctima creyendo que es algún procedimiento interno realice las acciones mencionadas en el correo.
- *Phishing* basado en *malware.* La diferencia con el *Phishing* por correo electrónico, que tienes que abrir un enlace y descargar un archivo para que se active el *malware,* con este método el propio correo electrónico en sí es el *malware.*
- *QRishing.* Este tipo de ataque se realiza a través de un QR modificado con un *malware* que se instala en el teléfono cuando es leído.

3.1.2. Detección temprana y contra medidas

Existen medidas preventivas para evitar este fraude

- Evitar abrir correos de contactos que no son confiables, que nos pidan algún tipo de pago o pidiendo información personal.
- Leer los correos con atención y estar atento a caracteres extraños tanto dentro del cuerpo del mensaje como en el remitente del mensaje.
- Comprobar si existen errores ortográficos en el cuerpo del mensaje, también comprobar si parece una traducción o si existe algún error en el formato.
- No acceder a ningún enlace ni descargar nada.
- Comprobar que la dirección de la página web no contenga ningún carácter extraño y comprobar los indicadores de seguridad.
- Marcar como *spam* todos los correos que no nos parezcan de confianza.
- Evitar conectarse a redes no fiables y si no es posible no utilizar aplicaciones de banca electrónica ni similares.

3.2. Otros trabajos existentes

En esta sección se mostrarán trabajos similares de otros autores y como lo han abordado.

Otra forma de abordar este experimento sería con el clasificador *Gaussian Naive Bayes (GaussianNB)* que usa el algoritmo *Gaussian Naive Bayes* que supone que la probabilidad de sus características es *gaussiana* que sería el siguiente:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{3.1}$$

Dando una exactitud de 95.8118556701031 Una vez entrenado el modelo se ha realizado otra prueba con el clasificador *RandomForestClassifier* estimador que ajusta una serie de clasificadores de árboles de decisión o *decision tree* en varias sub partes del conjunto de datos y utiliza el promedio para mejorar la exactitud y limitar el sobre ajuste. En este caso con este otro método se ha conseguido una exactitud de 97.6159793814433, ambos partes están dentro del mismo trabajo, se puede consultar el código completo de este autor a continuación [10].

En el siguiente caso se ha abordado el problema con el clasificador *MultinomialNB* que está basado en algoritmo *Naïve Bayes* para datos con distribución multinomial. El resultado de este experimento devuelve una exactitud de 98.65470852017937, bastante superior a otros clasificadores, en este mismo experimento también se ha abordado *RandomForestClassifier* con una exactitud de 97.847533632287, en este experimento se ha dado una exactitud mayor porque se ha aplicado la matriz de confusión, se puede consultar el código completo de este autor a continuación [11].

En el último caso, se ha abordado el mismo problema con 6 modelos distintos, los cuales son, Regresión logística, SVC, KNN, clasificador MLP, *RandomForestClassifier* y por redes neuronales.

Regresión logística utiliza un algoritmo que clasifica según las clases o categorías, se clasifica de forma binaria siendo los valores 1 o 0, este modelo tiene una exactitud de 95.61.

SVC o clasificación de vectores de soporte, este modelo escala con el número de muestras, aumentando el tiempo de ejecución con un gran número de datos, la exactitud que ha obtenido este modelo ha sido 98.03.

KNN(*KNeighborsClassifier*) clasificador basado en el aprendizaje basado en los k vecinos más cercanos, estando k relacionada con los datos de entrada. Este modelo muestra una exactitud de 94.35.

Clasificador MLP(*MLP Classifier*) o Clasificador Perceptron multicapa. Consta de varias capas que están conectadas entre sí, de manera que siempre están conectadas con la siguiente. Este modelo muestra una exactitud de 98.39.

RandomForestClassifier. Este modelo muestra una exactitud de 97.13

Redes neuronales(*Neural Networks*) modelo de red neuronal con 5 capas y optimizado con Adan, en este caso la exactitud ha sido de 98.74 siendo la más alta de este trabajo. se puede consultar el código completo de este autor a continuación [12].

4

Desarrollo

En este capítulo vamos a profundizar tanto en como hemos creado la aplicación, con qué tecnologías y la arquitectura que hemos seguido.

4.1. Desarrollo

En esta sección profundizaremos sobre el desarrollo de la aplicación llevada a cabo, tanto en el lenguaje utilizado, el tipo de objeto que vamos a crear, librerías utilizadas, programas utilizados para la implementación y arquitectura.

4.1.1. Lenguaje utilizado

Vamos a desarrollarlo con Python por su facilidad para la creación de Machine Learning, el lenguaje se hace hincapié en la transparencia y la comprensibilidad de su código.

Es un lenguaje de programación que usa varios modelos, ya que en parte soporta la orientación a objetos, también incorpora programación imperativa y en menor parte programación funcional. Es un lenguaje multiplataforma porque puede desarrollarse en distintos entornos [13].

Es un lenguaje dinámico porque las operaciones que se realizan en tiempo de compilación también se pueden realizar en tiempo de ejecución.

Es un lenguaje de programación interpretado, ya que el código se ejecuta línea a línea, instrucción por instrucción.

Este lenguaje de programación tiene una gran variedad de usos y aplicaciones, en nuestro caso la usaremos por la aplicación de algoritmos y librerías compatibles para Machine Learning.

4.1.2. Tipo de programas que vamos a desarrollar

Utilizaremos el clasificador *(Complement Naïve Bayes)* ya que nos sirve para corregir las suposiciones severas realizadas por el clasificador estándar *Multinomial Naïve Bayes*. Es especialmente adecuado para conjuntos de datos desequilibrados. cuando tenemos muchos más datos de una clase que de otra, podemos entrenar con un número similar, aunque luego la evaluación no esté balanceada.

Hemos escogido modelos de Liblinear porque es un clasificador lineal que admite clasificación y regresión, tiene disponible la opción multinucleo por lo que aumenta bastante la velocidad del entrenamiento y selecciona automáticamente los parámetros para optimizar el resultado. También hemos escogido los algoritmos basados en SGD o descenso de gradiente estocástico por su popularidad, es un algoritmo iterativo, ya que puede empezar desde un punto totalmente aleatorio de una función y viaja por la pendiente de esta función hasta llegar al punto más bajo y se puede usar en una gran variedad de situaciones distinta. Thresholding o clasificación de umbral es un clasificador que utiliza la regresión logística que nos devuelve una probabilidad que se transforma en un valor binario, con la cual se calcula el umbral de clasificación o también llamado umbral de decisión, en nuestro caso un valor por encima significaría que el correo es *spam* y un valor por debajo que no lo es.

Con estos clasificadores entrenaremos una pequeña cantidad de datos del conjunto y obtendremos el valor de precisión que ha obtenido cada modelo para así seleccionar el que consideremos más óptimo para analizar el resto de datos.

Una vez seleccionado el modelo, también utilizaremos otro modelo basado en una red neuronal con un algoritmo de regresión logística en contraparte para comparar los resultados obtenidos de los dos modelos finales y así analizar cuál de los dos sería más óptima para este tipo de situaciones.

4.1.3. Librerías más utilizadas

- Pandas: esta librería nos permitirá leer y escribir ficheros en varios formatos como pueden ser CSV, Excel y bases de datos SQL, nos permitirá ordenar por índices y columnas la información extraídas del archivo [14].
- NumPy: acrónimo de *Numerical Python* es una librería que se especializa en el análisis de datos de gran volumen porque los archivos que se suelen recibir son de gran tamaño, permite crear vectores y matrices multidimensionales de gran tamaño, cuenta con una serie de funciones matemáticas para operar con ellas [15].
- Seaborn: librería que nos permite crear fácilmente sofisticados gráficos. *Seaborn* se basa en *Matplotlib* y proporciona una interfaz de alto nivel, en Anaconda esta librería está instalada por defecto.
- Matplotlib: herramienta que nos permite generar gráficos contenidos en listas de dos dimensiones a partir de los datos obtenidos, nos permite crear y personalizar los tipos de gráficos y su extensión *NumPy*
- TensorFlow: es la librería por excelencia para la generación de aprendizaje profundo, es de código abierto y fue creado por Google y publicado bajo Apache 2.0, la API principalmente es usada en Python, esta librería fue diseñada para investigación y desarrollo. Tiene gran versatilidad porque se puede usar en CPU, GPU, Dispositivos móviles y en cientos de máquinas [16].
- Scikit-learn: librería de código abierto que proporciona una gran variedad de algoritmos de aprendizaje supervisado y también no supervisados, se programa en Python y como

está basado en *NumPy*, *SciPy* y *Matplotlib* se puede aprovechar el código que use estas librerías [17].

- Nltk: contiene un conjunto de librerías y programas en *Python* que nos serán útiles para el procesamiento de lenguajes estadísticos [18].

4.1.4. Algoritmos usados en los modelos

En nuestro proyecto hemos realizado dos máquinas con dos algoritmos distintos, en este caso hemos utilizado algoritmos de redes neuronales y algoritmos bayesianos.

1. Prueba de rendimiento de varios modelos distintos para optar por una prueba final usando un modelo predefinido en contra de un modelo basado en red neuronal.

2. Los algoritmos bayesianos son algoritmos que utilizan el Teorema de Bayes de probabilidades para problemas de Clasificación y Regresión [19]. Usaremos *ComplementNB* que asume que la probabilidad previa de características es distribución polinomial

3. Los algoritmos de redes neuronales están basados en algoritmos y estructuras que intentan imitar las funciones biológicas de las redes neuronales, principalmente se pueden utilizar para problemas de Clasificación y Regresión, pero tienen una gran capacidad para poder resolver una gran cantidad de problemáticas. Suelen dar muy buenos resultados para detectar patrones [19].

4. Las Redes Neuronales Artificiales usan una gran capacidad de procesamiento y memoria y con la capacidad de la tecnología del pasado estuvieron muy limitadas hasta estos últimos años en los que han estado resurgieron con bastante hincapié, dando como resultado el Aprendizaje Profundo, lo utilizaremos en nuestro modelo porque el algoritmo de regresión logística que es un tipo de análisis estadístico que trata de modelar la predicción de una variable cualitativa binaria (dos posibles valores) en función de las variables que pueden ser independientes o predictoras. La regresión logística se usa principalmente para la creación de modelos de clasificación binaria [19].

Para la compilación de este modelo se ha utilizado:

1. Se ha utilizado un optimizador que implementa el algoritmo *Adamax*. Es una variante del algoritmo Adán basada en la norma del infinito [20]. Los parámetros predeterminados siguen los proporcionados en el documento. *Adamax* es a veces superior a Adam, especialmente en modelos con incrustaciones.

2. Para las perdidas hemos utilizado *'binary crossentropy'* que calcula la métrica de entropía cruzada entre las etiquetas y las predichas.

En nuestro caso hemos optado por una neurona de entrada, dos capas ocultas, la primera de 16 neuronas y la segunda de 8 por último la capa de salida tendrá 1 neurona.

4.2. Pre procesado

Cuando se entrenan modelos basados en redes neuronales, va a realizar al menos dos tipos de transformaciones de los datos que vamos a recibir en nuestro conjunto de datos o " dataset".

1. Binarización u *One hot encoding* de las variables de entrada que recibimos, que son denominadas variables categóricas, consisten en crear por cada valor diferente una etiqueta que exista en la peculiariedad que codificamos, es decir, podemos crear etiquetas para dividir nuestras variables de la forma que nos sea más lógica.

2. Estandarización - Transformamos los valores de manera que la media de los valores sea 0 y la desviación estándar sea 1. Para realizarlo se calcula la media y la desviación estándar de los valores que hemos recibido en cada uno de los puntos de datos y después restando la media y dividirlo por la desviación estándar, normalmente este tipo de operaciones las realizaremos a través de funciones ya creadas

3. Escala de funciones- El escalado de características sería la parte final del preprocesamiento de datos para Machine Learning. Es un método para estandarizar las variables independientes de un conjunto de datos dentro del mismo rango característico, permitiendo limitar el rango de variables para que se pueda comparar en términos comunes.

4.2.1. Entrenamiento y predicción

1. Primero se realizará el entrenamiento, en esta parte el programa va a aprender tendencias, comportamientos o patrones que se ajusten a los datos que hemos divido para el conjunto de prueba [21]. Para así optimizar los parámetros de una función para que la salida que nos dé sea lo más cercano a al resultado real que queremos predecir.

2. La etapa de predicción viene después que se ha completado las iteraciones del entrenamiento con el conjunto de entrenamiento y este proceso va a evaluar datos completamente nuevos que no se han tenido en cuenta para el entrenamiento utilizando la función que se ha optimizado para obtener un resultado lo más ajustado posible al real.

3. Dependiendo del resultado podemos dividirlo en varios tipos de problemas, al obtener una categoría sería un problema de clasificación, si el resultado es un número es un problema de regresión. Es bastante interesante poder distinguir los puntos a los que nos vamos a enfrentar para escoger la mejor estrategia posible

4.3. Implementación

4.3.1. Con qué aplicaciones se ha implementado

Hemos ejecutado el programa con *Notebook JupyterLab* que se ejecuta a través del programa Anaconda. Hemos escogido este programa por las siguientes razones:

1. Anaconda: Administra los paquetes de las diferentes versiones de *Python*, con una distribución de código abierto, esto nos ayudará a preparar el entorno para desarrollar nuestra ML.

2. Jupyter Notebook: Es un cuaderno interactivo en el que puedes realizar cualquier actividad. En este cuaderno, puede escribir código y ejecutarlo, escribir documentos, tener diagramas, esquemas, etc. Con otros programas, escribimos código en el editor y luego escribimos documentos en Word para presentar el proyecto. Con Jupyter Notebook, podemos implementar la programación y la escritura, y desarrollar programas de acuerdo con nuestra propia lógica.

La ejecución ha sido en local para así reducir el tiempo de ejecución del programa.

4.3.2. Implementación del proyecto en ciberseguridad

Una vez que tengamos asegurados los datos, en este caso se trata de un dataset descargado de Kaggle, lo siguiente que tenemos que replantearnos es qué conocimientos o aplicaciones prácticas queremos conseguir con la implementación del Machine Learning. Este aspecto es el más importante, pues dependiendo de la necesidad específica de cada negocio se optará por alguna de las diferentes soluciones que existen dentro de los ML.

En este caso hemos optado por implementar dos soluciones:

1. Solución personalizada. Esta opción suele ser la más precisa, pero requiere una gran cantidad de datos y definir los algoritmos de nuestro modelo de Machine Learning, en este caso lo hemos creado con nuestra red neuronal.

2. Modelos predefinidos. son un modo de implementar Machine Learning de una forma rápida y bastante sencilla dentro de ML, pero puede entrañar cierta dificultad a aquellas empresas que no tienen bastante experiencia en el uso de estas herramientas o no cuentan con empleados cualificados para operar dichas herramientas.

Sin duda con estas dos opciones se trata de una buena solución para introducir a un negocio en la Inteligencia Artificial rápidamente, pero para poder obtener buenos resultados y hacerlo pronto necesitamos datos de prueba para poder entrena y afinar los logaritmos. Dentro de la ciberseguridad y nuestro proyecto se ha implementado el Machine Learning para recibir un archivo CSV, el siguiente paso es identificar el tipo de data que será útil para solucionar el problema propuesto, etiquetar los datos y analizarlos para poder realizar las predicciones.

4.4. Arquitectura

4.4.1. Arquitectura de modelos basados en redes neuronales

Dentro de las redes neuronales nos referimos a arquitectura al conjunto de:

- Número de capas neuronales
- Número de neuronas en cada capa
- Conexión entre neuronas
- Conexión entre capas
- Tipos de neuronas
- Forma de entrenar red neuronal

El tipo que utilizaremos será *Feed fordward networks*, este tipo de red es una red pre alimentada, este tipo de red fue una de las primeras formas de red neuronal, a su vez es una extensión de perceptrón que nos permite realizar cálculos para detectar patrones [22].

Se compone de varias capas neuronales interconectadas de manera *"fully connected"* (Una neurona está conectada con todas las neuronas de la siguiente capa sucesivamente hasta la salida, existe por lo menos una capa oculta, el flujo de datos fluye de izquierda a derecha *("feed forward")* y la red se entrena mediante retropropagación *(back-propagación)*.

Nuestra arquitectura constará de una capa de entrada con una neurona y dos capas ocultas, la primera con un total de 16 neuronas y la segunda con un total de 8 neuronas y dispondremos de una capa de salida con 1 neurona, por lo que tendría un total de 4 capas, en resumen tendremos una arquitectura unidireccional multicapa.

Retropropagación *(Backpropagation)* Este algoritmo se basa en que el error de las capas anteriores está directamente relacionado con el error de las capas posteriores, por ello gracias al análisis podemos optimizar la propagación del error hacia atrás en la red, con estos datos podemos asumir también que si tenemos el error controlado para N neurona de la m-ésima capa, entonces la configuración de parámetros para cada una de las neuronas de las m-1 capas anteriores que tienen influencia en los datos de entrada de nuestra neurona N, es relativamente poco influyente para nuestro error final [23].

Aquí tenemos un ejemplo de los tipos de arquitectura:

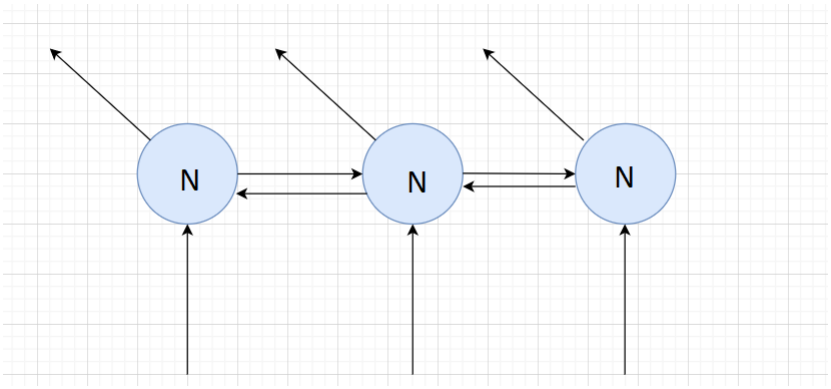

Figura 4.1: Monocapa retroalimentada

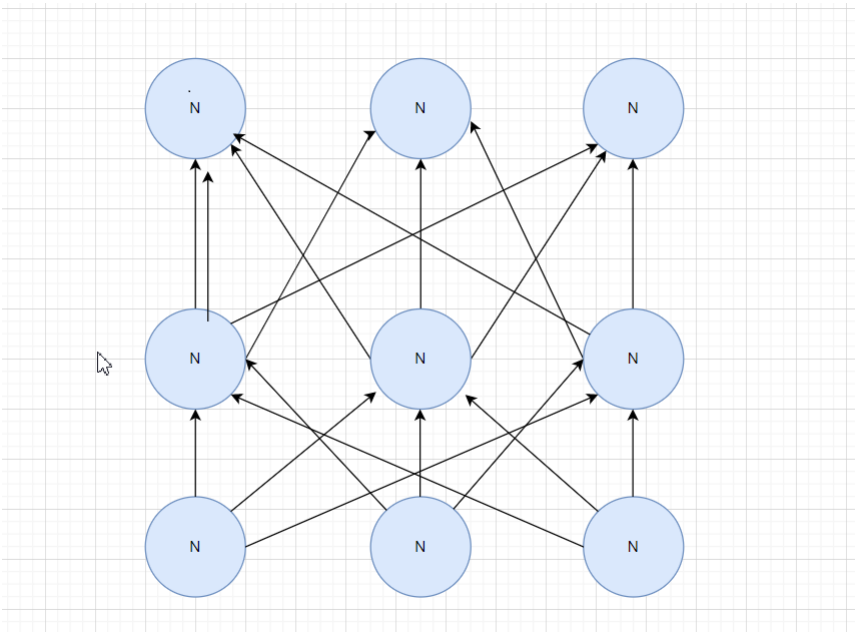

Figura 4.2: Multicapa unidireccional

4.4.2. Arquitectura de modelos árbol de decisiones

Estos modelos son una opción bastante completa y alternativa al modelo basado en neuronas, están basados en la toma de decisiones y es una de las técnicas de predicción más sencillas, pero también de la más eficaz, ya que es utilizada en todos los sectores.

Forma una estructura en forma de árbol con nodos y ramas que se extienden, dándole esa característica forma de árbol.

Parte de la idea de que existe un problema o pregunta a resolver y existen diferentes opciones para abordar este problema, a su vez al tomar una de las decisiones que compondrían las ramas de este árbol existen más decisiones que se ramifican a partir de una decision tomada hasta la salida que representarían las hojas, para llegar a este punto es necesario definir las decisiones que se pueden llevar a cabo, precisar las variables que van a intervenir en las decisiones y determinar las reglas de decisión con las que conseguiremos las decisiones óptimas.

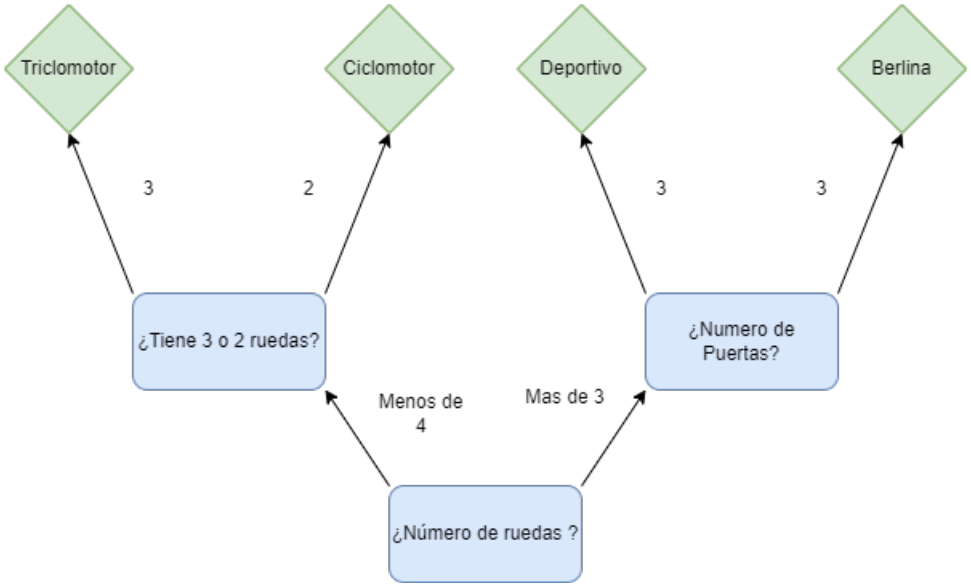

Figura 4.3: Ejemplo árbol de decisiones para la clasificación de autos

4.4.3. Arquitectura clasificadores Naïves

Este tipo de clasificadores usan el algoritmo *Naïve Bayes* o también llamado Bayesiano ingenuo, es decir, dentro del algoritmo se aplica el teorema de Bayes, que destaca por su rendimiento respecto a la velocidad para clasificar, además de su facilidad para implementarlos, dentro de este modelo las variables predictoras no están relacionadas entre sí.

Los puntos fuertes de usar este tipo de clasificadores, aparte de los mencionados anteriormente, son que aunque tenga menos datos de entrenamiento, su comportamiento es superior al de otros modelos de clasificación, por otro lado, son conocidos por ser pobres estimadores, por lo cual las probabilidades que obtiene no son del todo verídicas, otro de sus puntos débiles es que cuando una característica no ha sido observada en el momento de entrenar el conjunto tendrá un valor de 0 por lo que no podrá predecir esa característica. El Teorema de Bayes expresa que P (B—A) = (P(B) x P (A—B)) / P(A)

Para entender este algoritmo se usan redes bayesianas, son modelos gráficos probabilísticos,

a continuación se mostrará un gráfico para entender como funciona la toma de decisiones, según la probabilidad que tenga de suceder.

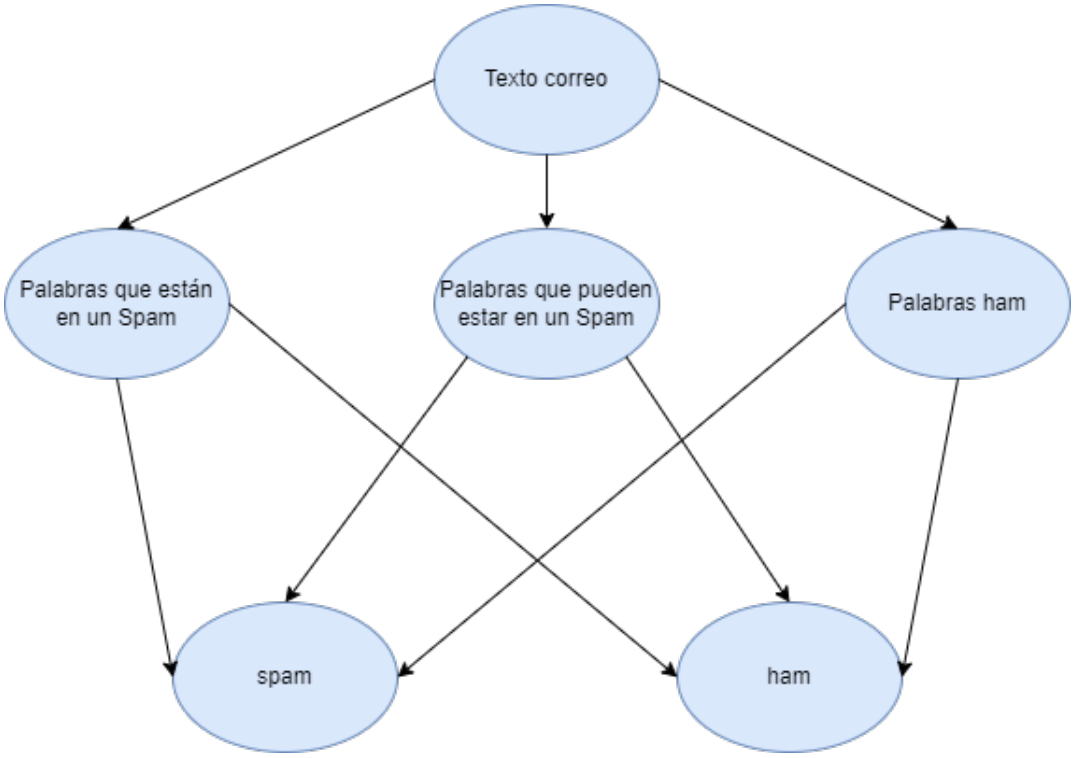

Figura 4.4: red bayesiana

5

Validación y evaluación

5.1. Experimento 1. Pruebas pre procesado de texto

Al recibir los datos que hemos leído del conjunto de datos es necesario pre procesarlos para poder organizarlos, para llevar a cabo y comprobar que datos nos interesan y ver el estado de esos datos, se puede utilizar para limpiar un texto, crear una matriz o convertir esos datos en datos que puedan ser procesados por el Machine Learning.

Primero validaremos los datos que hemos recibido y hemos reorganizado cambiando el nombre de las etiquetas por unas que nos resulte más practicas y los mostramos por pantalla para ver como queda el formato de ellos y poder visualizar el número de registros recibimos.

Out[3]:

	etiqueta	texto	clase
0	ham	Subject: enron methanol ; meter # : 988291\r\n...	0
1	ham	Subject: hpl nom for january 9 , 2001\r\n(see...	0
2	ham	Subject: neon retreat\r\nho ho ho , we ' re ar...	0
3	spam	Subject: photoshop , windows , office . cheap ...	1
4	ham	Subject: re : indian springs\r\nthis deal is t...	0
5	ham	Subject: ehronline web address change\r\nthis ...	0
6	ham	Subject: spring savings certificate - take 30 ...	0
7	spam	Subject: looking for medication ? we ` re the ...	1
8	ham	Subject: noms / actual flow for 2 / 26\r\nwe a...	0
9	ham	Subject: nominations for oct . 21 - 23 , 2000\...	0

Figura 5.1: Contenido conjunto de datos

Después de esta prueba utilizaremos la variable que nos muestra el resultado real y la mostraremos en forma de gráfico para poder tener una mejor idea de los resultados que deberíamos dar en el futuro.

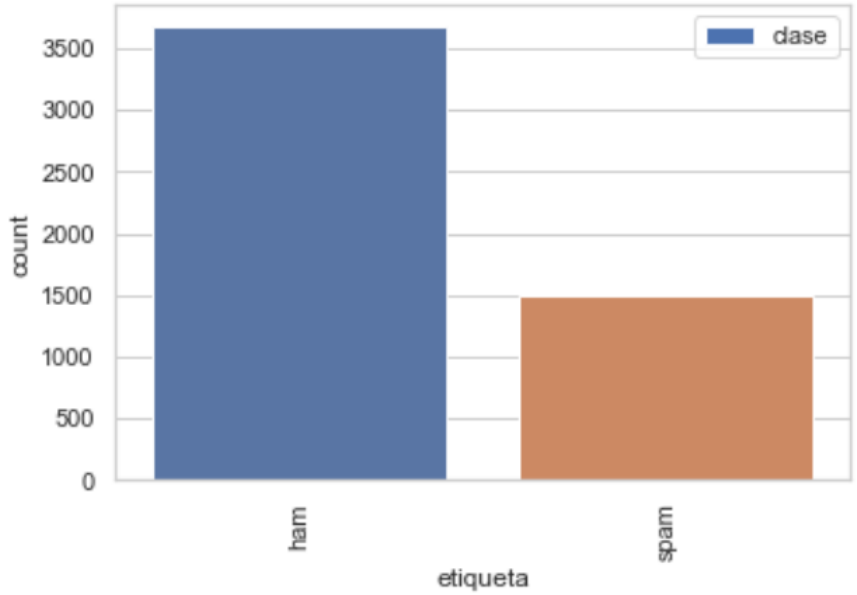

Figura 5.2: Diagrama ham/spam

Ahora mostraremos la cabecera de nuestro texto después de limpiarlo, quitándole los caracteres extraños y tokenizando el texto

	etiqueta	texto	clase
0	ham	subject enron methanol meter follow note gave ...	0
1	ham	subject hpl nom january see attached file hpln...	0
2	ham	subject neon retreat ho ho ho around wonderful...	0
3	spam	subject photoshop windows office cheap main tr...	1
4	ham	subject indian springs deal book teco pvr reve...	0

Figura 5.3: Texto después de formateo

5.2. Experimento 2. Pruebas selección modelo

Para seleccionar el modelo previamente vamos a convertir el documento de texto en un vector en función del recuento de cada palabra que aparece en todo el texto, usando la función *CountVectorizer*, después vamos a dividir los arreglos o matrices en subconjuntos de prueba y entrenamiento, podemos ver el número de observaciones y el número de tokens que consta. Ejemplo. En un texto que contiene dos registros, la matriz generada seria ['Esto es spam', 'Esto no es spam'], cada palabra adquiriría un vector único.

Esto	no	es	spam
1	0	1	0
0	1	1	1
0	0	0	0

Tabla 5.1: matriz vectores

```
NÚMERO DE OBSERVACIONES: 5171
TOKENS: 45595
```

Figura 5.4: Número de observaciones y tokens

Después entrenaremos el texto con los modelos que hemos seleccionado para poder obtener el tiempo que ha tardado en entrenar con los datos facilitados y la exactitud que ha logrado cada modelo en una gráfica.

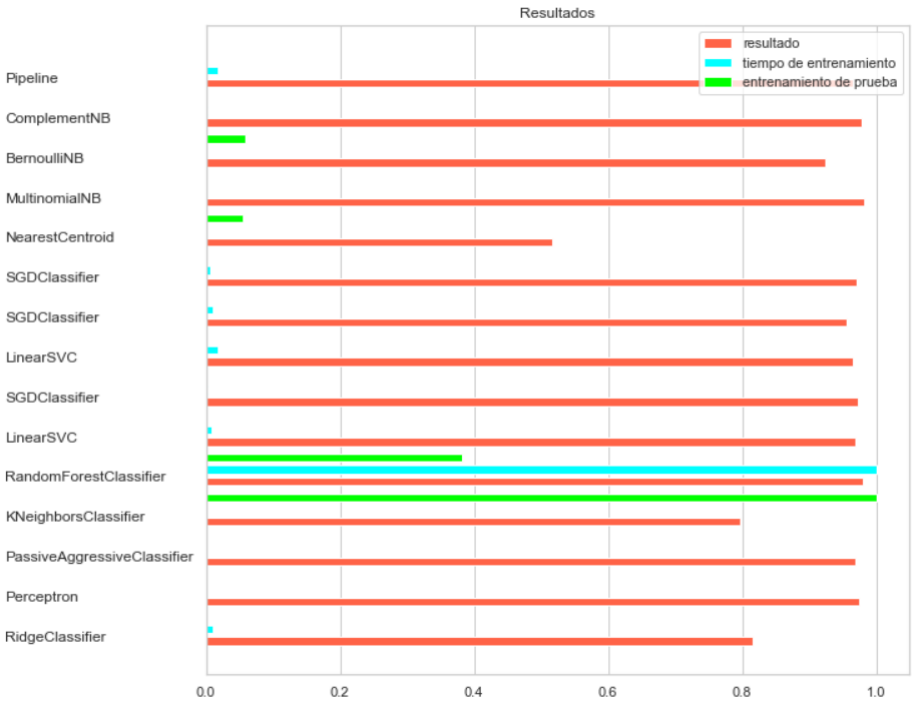

Figura 5.5: Rendimiento modelos

KNeighborsClassifier es el que mejor tiempo de entrenamiento demuestra con una estimación

de 0.003 y una exactitud del 0.79, por lo que quedaría descartado por la exactitud.

En este caso el que mejor rendimiento nos muestran es *ComplementNB* por lo que será el seleccionado para los siguientes experimentos. Necesitaremos obtener los mejores hiperparámetros para el modelo seleccionado, para eso utilizaremos *RandomizedSearchCV* para poder buscar los mejores parámetros y mostrarlos en un formato de lista indicando los parámetros y la exactitud que hemos conseguido.

Out[14]:

	params	mean_test_score
1	{'alpha': 0.2, 'fit_prior': False}	0.979207
3	{'alpha': 1, 'fit_prior': False}	0.977999
5	{'alpha': 2, 'fit_prior': False}	0.977997
2	{'alpha': 1, 'fit_prior': True}	0.977273
0	{'alpha': 0.2, 'fit_prior': True}	0.976789
4	{'alpha': 2, 'fit_prior': True}	0.973887
7	{'alpha': 5, 'fit_prior': False}	0.964458
6	{'alpha': 5, 'fit_prior': True}	0.956963
9	{'alpha': 10, 'fit_prior': False}	0.945599
8	{'alpha': 10, 'fit_prior': True}	0.937619

Figura 5.6: Exactitud hiperparámetros

Una vez seleccionado el modelo predefinido es hora de entrenarlo, una vez entrenado hemos conseguido una exactitud del 97,89 que es bastante buena. Mostraremos una matriz de confusión para poder ver de manera visual el desempeño que realiza el algoritmo.

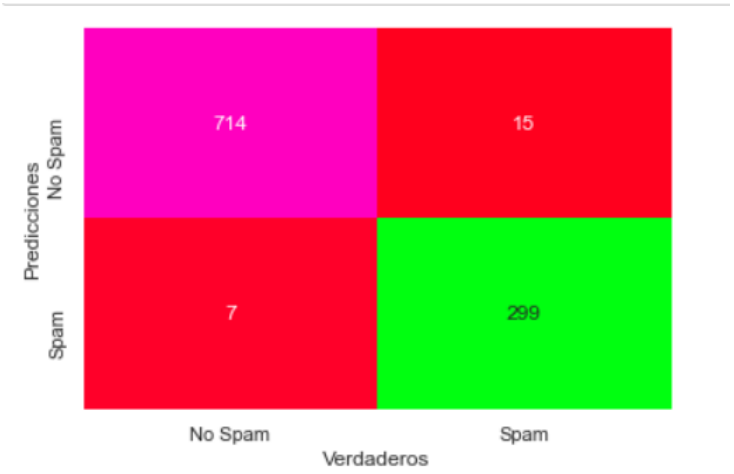

Figura 5.7: Matriz de confusión

5.3. Experimento 3. Pruebas modelo red neuronal

Para poder seleccionar las capas y las neuronas en nuestra red neuronal hemos entrenado la red con distintos parámetros a base de prueba y error para conseguir los parámetros óptimos, cabe mencionar que a mayor número de neuronas se puede llegar a mejorar la exactitud, pero también aumentan los tiempos que tarda en entrenar la red, a continuación vamos a mostrar el sumario de nuestra red

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                32
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 177
Trainable params: 177
Non-trainable params: 0
_________________________________________________________________
```

Figura 5.8: Sumaria red neuronal

Una vez probado el modelo es hora de validarlo para poder obtener la exactitud y las perdidas al evaluar los datos, obtenemos una exactitud de 71,35. Una vez obtenidos los datos vamos a visualizarlo en un gráfico que nos mostrara la precisión en el entrenamiento y validación y la perdida en entrenamiento y validación

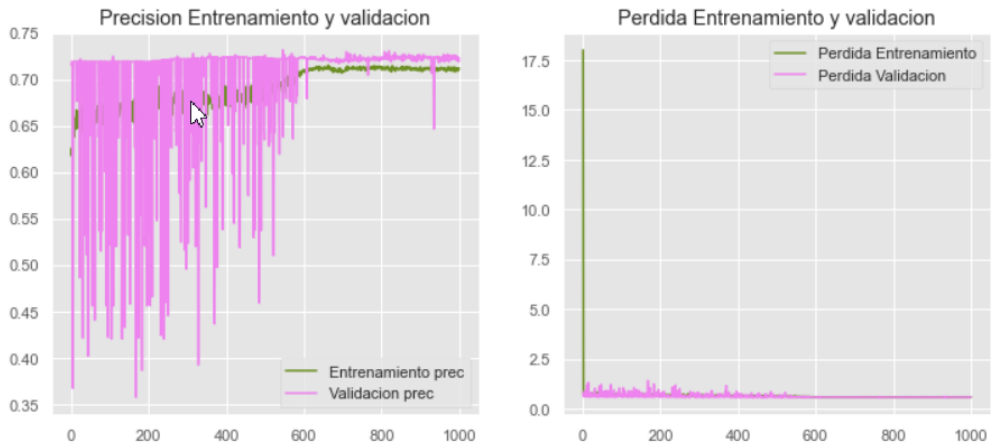

Figura 5.9: Precisión y perdida entrenamiento y valicacion

5.3.1. Experimento 4. Prueba final resultado comparativa

En este párrafo se mostrarán los resultados de la comparativa de los dos modelos usados, en estos resultados se tienen en cuenta la exactitud obtenida por varios modelos usando un mismo

set de datos.

```
# Exactitud ComplementNB: 97.89999999999999%

# Exactitud red neuronal: ('accuracy', 71.47002220153809)%
```

Figura 5.10: Resultado final:Exactitud modelo ComplementNB y red neuronal

En este caso el modelo que mejor resultado nos ha dado para el documento ha sido *ComplementNB* dando como resultado una exactitud de 97.89, muy superior a los modelos basados en redes neuronales que nos ha arrojado una exactitud de 71.47, después de esta prueba se genera un archivo indicando cuantos de los que han sido analizados por *ComplementNB* han coincidido con el fichero y cuáles no para futuros análisis de datos.

6

Conclusiones y trabajo futuro

En este capítulo se exponen las conclusiones y hallazgos extraídas del estudio llevado a cabo y se proponen líneas de trabajo que se podrían realizar en un futuro como fruto de este proyecto.

Las inteligencias artificiales se están desarrollando enormemente en todos los ámbitos, ya que debido a su funcionalidad tiene una infinidad de usos. Las Machine Learning son el futuro tanto en ciberseguridad como en todas sus aplicaciones, porque la capacidad de automatizar procesos y que puedan aprender mientras los procesan aumenta la efectividad y mejora el rendimiento. Podríamos usarlas tanto para medir la predicción de que una máquina sea infectada por un malware, pudiendo proteger esa máquina mejor, en el ámbito de seguridad de la empresa se podrían evaluar las contraseñas que se usan en su empresa.

Durante este Trabajo Fin de Máster se ha estudiado el rendimiento y la eficacia de diferentes algoritmos y modelos para la clasificación, correos spam o no spam.

Se ha probado el rendimiento de los modelos de Machine Learning, tales como *Perceptron*, *PassiveAggressiveClassifier*, *KNeighborsClassifier*, *RandomForestClassifier*, *LinearSVC*, *SGDClassifier*, *NearestCentroid*, *BernoulliNB*, *MultinomialNB* y *ComplementNB*. Se han construido los siguientes modelos de Machine Learning y se ha comparado su eficacia, *ComplementNB* y basado en redes neuronales. En relación con el análisis de los resultados de los datos obtenidos en los diferentes experimentos, el modelo *MultinomialNB* con los mejores hiperparametros escogidos por *RandomizedSearchCV* un rendimiento y eficacia bastante competitivo, de este modo los otros algoritmos también han resultado acertado para la clasificación de correo *spam* desde otros puntos de vista.

Con respecto a futuros trabajos sobre este proyecto, se plantea crear un plug-in para el navegador que funcione solo sobre nuestra bandeja de correo electrónico que sea capaz de leer el contenido de un fichero y generar un archivo en el mismo formato que acepta el proyecto y a su vez generar un fichero con los datos del correo *spam* para así poder marcarlos manualmente como *spam* en caso de no estarlos, a su vez también se propone que este mismo plug-in pudiese leer el fichero generado y poder marcar los correos automáticamente, a su vez se podría utilizar el fichero de salida con los datos de los correos spam con el cual se podría generar un diccionario con las palabras que contiene el texto de los correos.

Recomendable para futuros trabajos sobre este proyecto visitar https://www.kaggle.com/

Para finalizar, se confirma que se han cumplido todos los objetivos que se han propuesto en este Trabajo de Fin de Máster, en la sección Objetivos.

Bibliografía

[1] "Aplicaciones del machine learning y la ia en ciberseguridad - hacknoid," https://www.hacknoid.com/hacknoid/aplicaciones-del-machine-learning-y-la-ia-en-ciberseguridad/.

[2] "Machine learning aplicado en ciberseguridad. todo lo que necesitas saber," https://www.esedsl.com/blog/machine-learning-aplicado-en-ciberseguridad.

[3] "Redes neurales - curso- fundamentos - luisamayateacher," https://sites.google.com/site/luisamayateacher/redes-neurales---curso.

[4] "Redes neuronales," https://inteligencia-u1.blogspot.com/2020/03/redes-neuronales.html.

[5] "Redes neuronales con python," https://www.cienciadedatos.net/documentos/py35-redes-neuronales-python.html#.

[6] "Redes neuronales con python," https://www.cienciadedatos.net/documentos/py35-redes-neuronales-python.html.

[7] "Las tecnologías en la nube y la inteligencia artificial - academia inteligencia artificial," https://academiainteligenciaartificial.com/hello-world/.

[8] "Metodologías de desarrollo software — blog becas santander," https://www.becas-santander.com/es/blog/metodologias-desarrollo-software.html.

[9] "Apwg — informes de tendencias de actividad de phishing," https://apwg.org/trendsreports/.

[10] "spam_ham_conjunto de datos — kaggle," https://www.kaggle.com/code/ayhampar/spam-ham-dataset.

[11] "spam-detection-98.65 % — kaggle," https://www.kaggle.com/code/chirag01/spam-detection-98-65.

[12] "Email spam detection classification-6models 98.65 % — kaggle," https://www.kaggle.com/code/hosamwajeeh/email-spam-detection-classification-6models-98-65.

[13] "Python - wikipedia, la enciclopedia libre," https://es.wikipedia.org/wiki/Python.

[14] "La librería pandas — aprende con alf," https://aprendeconalf.es/docencia/python/manual/pandas/.

[15] "Numpy - wikipedia, la enciclopedia libre," https://es.wikipedia.org/wiki/NumPy.

[16] "Introducción a tensorflow -cursos de programación de 0 a experto © garantizados," https://unipython.com/introduccion-a-tensorflow/.

[17] "Introducción a la librería scikit-learn de python -aprende ia," https://aprendeia.com/libreria-scikit-learn-de-python/.

[18] "Nltk - wikipedia, la enciclopedia libre," https://es.wikipedia.org/wiki/NLTK.

[19] "Principales algoritmos utilizados — aprende machine learning," https://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/.

[20] "Tensorflow - tf.keras.optimizers.adamax - optimizador que implementa el algoritmo adamax. hereda de: Optimizer compat alia - español," https://runebook.dev/es/docs/tensorflow/keras/optimizers/adamax.

[21] "Qué es machine learning? — blog xeridia sobre inteligencia artificial," https://www.xeridia.com/blog/que-es-machine-learning.

[22] "Arquitectura de redes neuronales — interactive chaos," https://interactivechaos.com/es/manual/tutorial-de-machine-learning/arquitectura-de-redes-neuronales.

[23] "Redes neuronales y deep learning backpropagation," https://www.futurespace.es/redes-neuronales-y-deep-learning-brackpropagation/.

Apéndices

Prerrequisitos Para ejecutar en local descargar Anaconda en https://www.anaconda.com/products/individual

Dentro de la aplicación Anaconda utilizar jupyter notebook Se puede ejecutar con la instrucción jupyter lab

Librerias necesarias para ejecutarlo Instalar Keras y tensorflow, desde anaconda prompt ejecutar la instrucción conda install -c conda-forge tensorflow.

Instalar Keras instrucción pip install keras.

Instalar pandas instrucción pip install pandas.

Instalar numpy instrucción pip install numpy.

Instalar nltk instrucción pip install nltk.

Instalar matplotlib instrucción pip install matplotlib.

Instalar seaborn instrucción pip install seaborn.

Actualizar librerias scikit-learn conda update scikit-learn.

Al ejecutar el programa por primera vez ejecutar linea nltk.download('punkt') nltk.download('stopwords')