Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA EN TECNOLOGÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DETECCIÓN DE MALWARE MEDIANTE APRENDIZAJE PROFUNDO

Alumna: Zufiaurre, Soto, Gloria
Director: Del Ser, Lorente, Javier

Curso: 2018-2019

Fecha: Miércoles, 10 de Julio del 2019

PÁGINA EN BLANCO

RESUMEN

RESUMEN

Las aplicaciones móviles son una fuente de vulnerabilidad para los hackers. Cada vez son más los ataques realizados a través de ellas. Por ello, es muy importante identificar qué aplicaciones son empleadas para realizar ataques. Esta identificación hace que el usuario evite la instalación de dichas aplicaciones en su dispositivo. Para poder clasificar una aplicación en malware o benignware, se crearán varios sistemas clasificadores mediante diferentes técnicas de *Machine Learning*.

Para la creación de los sistemas, se emplearán técnicas tradicionales de aprendizaje basadas en algoritmos clasificadores y *Deep Learning*. De todos los sistemas creados con las técnicas tradicionales se elegirán tres de ellos: el que tenga mayor exactitud, el que presente la precisión más elevada y, por último, aquel que más sensibilidad tenga. Finalmente, para cada una de las tres métricas, se decidirá si elegir el sistema entrenado mediante aprendizaje profundo o el entrenado con el aprendizaje tradicional seleccionado anteriormente. Así, se podrá hacer uso de tres herramientas con distintos enfoques capaces de detectar aplicaciones malignas.

Palabras clave: Machine Learning, Deep Learning, Clasificador, Aprendizaje, Entrenamiento, Exactitud, Precisión, Sensibilidad.

LABURPENA

Aplikazio mugikorrak ahultasun jatorri bat dira hackerrentzat. Haien zehar egiten diren erasoak gero eta gehiago dira. Horregatik, oso garrantzitsua da erasoak egiteko erabiltzen diren aplikazioak ezagutzea. Hori jakitearekin, erabiltzaileak aplikazio horiek bere mugikorr gailuan ez instalatzea lortzen da. Aplikazio mugikor bat malware edo benignware sailkatu ahal izateko, hiru sistema sailkatzaile sortuko dira *Machine Learning*-eko teknika ezberdinak erabiliz.

Sistemak sortzeko algoritmo sailkatzaileetan oinarritutako ohiko teknikak eta *Deep Learning* erabiliko dira. Ohiko teknikekin sortutako sistema guztietatik, hiru aukeratuko dira: doitasun garaiena duen sistema, sistema zehatzena eta sentikortasun handiena daukana. Azkenik, hiru metrika bakoitzerako, erabakiko da zein sistema aukeratu, sakon entrenamenduarekin sortutako sistema edo lehen aukeratu den ohikoz entrenatutakoa. Honela, aplikazio kaltegarriak sailkatzeko gai diren hiru tresna erabili ahal izango dira hiru ikuspegi ezberdinekin.

Hitz gakoak: Machine Learning, Deep Learning, Sailkatzailea, Ikasketa, Entrenamendua, Doitasuna, Zehaztasuna, Sentikortasuna.

ABSTRACT

Mobile applications are a source of vulnerability for hackers. Number of attacks performed through them are increasing. That is why It is really important to identify the applications which are mainly used to perform cyber-attacks. This app identification made the user avoid installing those apps on your mobile device. In order to be able to sort an app on malware or benignware, a classifier system will be set up using Machine Learning different methods.

Shallow learning Techniques based on classifier algorithms and Deep Learning methods are going to be used so as to create the systems. Three of all of the systems created using shallow learning technics will be chosen: the one which holds the highest accuracy, other which holds the best precision and finally, that which holds the highest recall. In the end, depending on each metric, it will be decided whether to choose between the system trained by Deep Learning and the previously chosen one trained by Shallow Learning. Thereby, three tools capable of detecting malicious apps with different approaches will be available to users.

Key words: Machine Learning, Deep Learning, Classifier, Learning, Training, Accuracy, Precision, Recall

SOMMARIO

Le applicazioni mobili sono una sorgente di vulnerabilità per gli hacker. Il numero di attacchi Messi in atto attraverso di esse sta crescendo. Per questo è molto importante identificare le applicazioni piú usate nei cyber-attacchi. Tale identificazione permette all'utilizzatore di non installare queste app sul suo device. Al fine di classificare un'applicazione come malware o benignware, verrano creati tre sistemi di classificazione usando diversi metodi di Machine Learning.

Verranno utilizzate le Shallow Learning tecnica basate sulla classificazione di algoritmi e Deep learning metodi. Tre, ta tutti, sistema creati usando le techniche di apprendimento di shallow verranno scelti: Quello che raggiunge la maggior accuratezza, quello che resulta più preciso e, infine, quello più sensible. Al termine, a seconda di ogni misurazione, si deciderà se scegliere il sistema messo a punto con il apprendimento profundo o quello precedentemente scelto con il método di apprendimento superficiale. In tal modo, the strumenti in grado di rilevare app dannose con differenti approcci saranno disponibili per gli utenti.

Parole chiave: Machine Learning, Deep Learning, Classificatore, Apprendimento, Allenato, Accuratezza, Precisione, Sensibilità.

TABLA DE CONTENIDO

RESUMEN **2**
1 INTRODUCCIÓN **9**
2 CONTEXTO **10**
3 OBJETIVOS **12**
3.1 OBJETIVO PRINCIPAL 12
3.2 OBJETIVOS SECUNDARIOS 12
4 BENEFICIOS **13**
4.1 TÉCNICOS 13
4.2 SOCIALES 13
4.3 ECONÓMICOS 14
5 ESTADO DEL ARTE **15**
5.1 INTELIGENCIA ARTIFICIAL 15
5.2 MACHINE LEARNING 15
5.2.1 Introducción *15*
5.2.2 Funcionamiento *16*
5.2.3 Tipos de Machine Learning *16*
5.2.4 Perspectivas y aplicaciones actuales *18*
5.3 DEEP LEARNING 18
5.3.1 Introducción *18*
5.3.2 Funcionamiento Redes Neuronales *18*
5.3.3 Proceso de aprendizaje de una red neuronal *19*
5.4 TRATAMIENTO DATOS 21
6 METODOLOGÍA **22**
6.1 OBTENCIÓN DE INFORMACIÓN 22
6.1.1 Descripción del dataset *23*
6.2 PRIMERA PARTE: SHALLOW LEARNING 25
6.2.1 Volcado de la información del dataset a Python y modificación de ella *26*
6.2.2 Entrenamiento *26*
6.2.2.1 Entrenamiento de un modelo clasificador *26*
6.2.2.2 Selección de características *26*
6.2.3 Predicción y medidas *27*
6.2.4 Evaluación de métricas combinando modelos con seleccionadores *29*

6.3 SEGUNDA PARTE: DEEP LEARNING 31
6.3.1 Volcado de la información del dataset a Python y modificación de ella *32*
6.3.2 Diseño y entrenamiento de una red neuronal *32*
6.3.3 Predicción y medidas *33*
7 ANÁLISIS DE ALTERNATIVAS 34
7.1 IDENTIFICACIÓN Y DESCRIPCIÓN DE ALTERNATIVAS 34
7.1.1 Modelos *34*
7.1.1.1 K-Nearest Neighbors Classifier *34*
7.1.1.2 DecisionTreeClassifier *36*
7.1.1.3 C-Support Vector Classifier *38*
7.1.1.4 Multi-layer Perceptron Classifier *39*
7.1.1.5 Gradient Boosting Classifier *39*
7.1.1.6 Gaussian Naive Bayes Classifier *40*
7.1.1.7 Random Forest Classifier *40*
7.1.2 Técnicas de selección de características *41*
7.1.2.1 Variance Threshold *41*
7.1.2.2 Select K Best *42*
7.1.2.3 Select Percentile *42*
7.1.2.4 Select Fpr, Fdr & Fwe *42*
7.1.2.5 Generic Univariate Select *43*
7.1.3 Red neuronal *43*
7.2 ELECCIÓN DE ALTERNATIVAS 44
7.2.1 Interpretación de los sistemas elegidos en base a las métricas *45*
8 PLANIFICACIÓN 46
8.1 CICLO DE VIDA DEL PROYECTO 46
8.2 RECURSOS NECESARIOS PARA SU DESARROLLO 47
8.3 DIAGRAMA DE GANTT 48
9 PRESUPUESTO EJECUTADO 50
9.1 HORAS INTERNAS 50
9.2 AMORTIZACIONES 50
9.3 GASTOS 51
9.4 COSTE TOTAL 51

10 ANÁLISIS DE RIESGOS 52
10.1 R1: PRESUPUESTO DEMASIADO BAJO 52
10.2 R2: ERRORES EN EL DESARROLLO SOFTWARE 53
10.3 R3: SUPERACIÓN FECHA LÍMITE 53
10.4 R4: PÉRDIDA DE DATOS O AVERÍA DE EQUIPOS 54
11 CONCLUSIONES 55
12 REFERENCIAS 56

LISTA DE ILUSTRACIONES

Ilustración 1: Número de aplicaciones Android normales y malignas añadidas por mes en Google Play Store 10
Ilustración 2: Áreas de la inteligencia artificial empleadas en este proyecto 15
Ilustración 3: Diferencia entre el clásico paradigma y el basado en machine learning 16
Ilustración 4: Tipos de algoritmos empleados para cada subcampo de Machine Learning [8] .. 16
Ilustración 5: Neurona real y su representación en una red neuronal creada con Deep Learning 19
Ilustración 6: Fases de aprendizaje de una red neuronal 20
Ilustración 7: Diagrama de estados Shallow Learning del proyecto 25
Ilustración 8: Importacia de las características 26
Ilustración 9: Matriz de confusión de un sistema del proyecto 27
Ilustración 10: Medida de la exactitud en algunos sistemas creados 29
Ilustración 11: Medida de la precisión en algunos sistemas creados 29
Ilustración 12: Diagrama de Estados Deep Learning 31
Ilustración 13: Pasos ejecución de algoritmo KNN Classifier 35
Ilustración 14: Árbol completo 37
Ilustración 15: Fragmento del árbol 37
Ilustración 16: Diagrama Gantt del proyecto 49
Ilustración 17: Matriz probabilidad-Impacto 54

LISTA DE TABLAS

Tabla 1: Principales diferencias entre aprendizaje tradicional y profundo [9] 18
Tabla 2: Actividades a realizar 48
Tabla 3: Recursos humanos 50
Tabla 4: Amortizaciones 50
Tabla 5: Gastos 51
Tabla 6: Coste Total 51
Tabla 7: Resultados de los sistemas elegidos 55

ACRÓNIMOS

- IA — Inteligencia Artificial
- MW — Malware
- BW — Benignware
- TIC — Tecnología de la Información y la Comunicación
- ADN — Ácido Desoxirribonucleico
- AMaaS — Audience Management as a Service
- BBDD — Bases de Datos
- Spyder — Scientific Python Development Environment
- App — Aplicación Informática (en el proyecto, es aplicación móvil)
- API — Application Programming Interface
- DRY — Don’t Repeat Yourself
- RAD — Rapid Application Development
- Sk-learn — Scikit-learn
- TP — True Positives
- FP — False Positives
- FN — False Negatives
- TN — True Negatives
- IFTTT — IF This Then That
- RBF — Radial Basis Function
- MLP — Multi-Layer Perceptron
- ANOVA-F — Analysis Of Variance Factor
- SVM — Support Vector Machine
- SO — Sistema Operativo

1 INTRODUCCIÓN

La finalidad de este proyecto es la realización de tres sistemas, cada uno de ellos con un enfoque diferente, capaces de clasificar una aplicación para el sistema operativo Android en software maligno o no. Se ha decidido crear para aplicaciones creadas para dispositivos Android, ya que estos representan el 80% del mercado, comparado con el sistema iOS, que representa alrededor del 15%.

A día de hoy, los dispositivos móviles, tablets… no cuentan con medidas de seguridad tan desarrolladas como los ordenadores. Los de ataques producidos en estos dispositivos suelen ser haciendo uso de aplicaciones que el usuario instala. Además, los ciber atacantes se encargan de elegir las herramientas y técnicas más óptimas para dificultar su detección, por ejemplo, empleando técnicas como vulnerabilidades de día cero. Esto puede convertir a los móviles o tablets en los mejores dispositivos espías jamás creados.

Es por ello que es conveniente la creación de técnicas de detección de malware para prevenir o disminuir lo máximo posible los ciber ataques. La mayoría de ataques realizados por medio de aplicaciones que son consideradas malware suelen presentar una serie de características que las diferencian de aquellas que no suponen ningún peligro para el usuario. Dicho esto, para poder llevar a cabo este proyecto, se empleará información proporcionada por [1] sobre ejemplos de aplicaciones, las cuales han sido etiquetadas con certeza como malware o no en base a las características que presenta cada app.

En este documento se describirán aspectos técnicos, económicos y sociales del proyecto, así como los beneficios del proyecto en cada aspecto. Sin embargo, al tratarse de un proyecto de ingeniería, el contenido técnico predomina sobre los otros dos. Por ello, antes de explicar su desarrollo técnico, se procederá a hablar sobre la situación actual de las técnicas empleadas para su desarrollo, *Machine Learning* y *Big Data*. Dichas técnicas son las encargadas de la creación del sistema clasificador basándose en el aprendizaje de una máquina mediante ejemplos.

Posteriormente, un apartado es dedicado especialmente a la metodología del desarrollo software, en este caso, se ha optado por el lenguaje de programación Python. De este modo se podrán explicar los procesos llevados a cabo durante dicho desarrollo. Como existe una única manera para alcanzar el objetivo, se evaluarán los diferentes modos de creación en base a la calidad que presente cada sistema creado.

Respecto al ámbito social y económico, se expondrá el contexto en el que está situado este proyecto y se explicará la planificación llevada a cabo en el mismo, para poder ver cómo ha sido organizado. Por otro lado, se mostrarán detalladamente los costes que ha supuesto llevar a cabo el proyecto y los riesgos que pueden conllevar la realización del mismo.

2 CONTEXTO

A día de hoy, Google Play Store es la tienda de aplicaciones más popular del mercado junto a la App Store de iOs. Tal y como expone [2] hay en tienda 2.702.938 aplicaciones Android, de estas, un 13% están repletas de malware, cuyo principal objetivo es infringir la propiedad de datos de los usuarios.

Estas aplicaciones maliciosas tienen más éxito de lo esperado, Lukas Stenfanko [3], experto en ciberseguridad, ha detectado varias aplicaciones que cargan en el teléfono móvil todo tipo de malware y adware. De esta manera, los desarrolladores de dichas apps hacen que sea posible mostrar anuncios invisibles. Esto hace que el funcionamiento del dispositivo móvil se ralentice y que los datos personales del usuario estén en peligro. Lo que es más impactante, es que son aplicaciones que registran más de 400.000 descargas, buenas valoraciones de los usuarios y se encuentran entre las más populares.

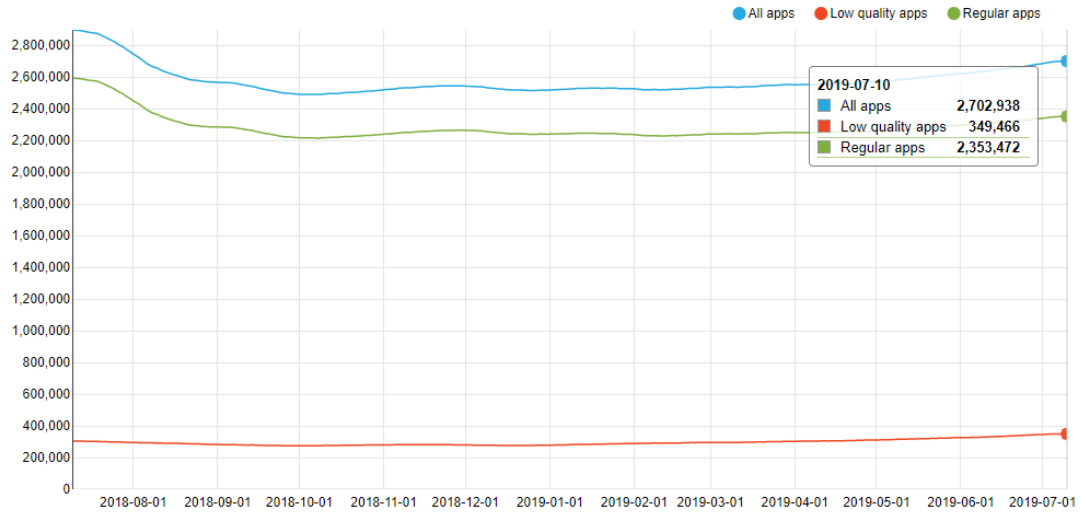

Ilustración 1: Número de aplicaciones Android normales y malignas añadidas por mes en Google Play Store

En la ilustración 1 se puede observar el número de aplicaciones malignas y benignas añadidas por mes. El decrecimiento, se debe a que Google elimina de vez en cuando aplicaciones con los consideradas de baja calidad, malware. Sin embargo, la presencia de apps malignas está siempre presente.

Por otro lado, no todas las aplicaciones Android provienen de Google Play Store, existen una gran cantidad de aplicaciones creadas por terceros, principalmente aquellas que imitan a una app original de pago haciéndola gratuita. Estas últimas son potentes portadoras de malware.

Tal y como explica Chema Alonso [4], miembro del consejo ejecutivo de telefónica, hacker y experto en seguridad, el número de nuevas aplicaciones en Android crece cada día, para un analista en seguridad, hacer frente al mundo del cibercrimen en Android es un trabajo duro, por ello, es necesario crear herramientas que ayuden a desempeñar este trabajo.

Por todo esto se deben crear herramientas que permitan clasificar una aplicación Android en maligna o benigna para evitar su instalación y/o uso por parte de los usuarios.

El propósito de este trabajo de fin de grado es la creación de herramientas/sistemas mediante inteligencia artificial. Dentro de este amplio campo, se emplearán técnicas de aprendizaje máquina tradicionales y de aprendizaje profundo. Estos aprendizajes se realizarán a partir de muestras reales.

Se trata de grandes cantidades de ejemplo, por lo que su tratamiento se hará haciendo uso de librerías capaces de trabajar con *Machine Learning* y operar con *Big Data*.

3 OBJETIVOS

3.1 OBJETIVO PRINCIPAL

El principal objetivo de este proyecto es la realización de tres sistemas óptimos, independientes entre sí, que permitan predecir si una aplicación para el sistema operativo Android se trata de malware o no. Para ello, es necesario evaluar las diferentes técnicas de Machine Learning empleadas para realizar el sistema y elegir aquella con la que se obtenga la mayor exactitud en la predicción.

3.2 OBJETIVOS SECUNDARIOS

Para una realización estructurada del proyecto, se han ido estableciendo etapas con objetivos en cada una de ellas. Cada uno de estos contribuirá al alcance del objetivo principal deseado. Para su correcto desarrollo se hará uso de un conjunto de datos reales (Dataset). Estos datos corresponden a características de aplicaciones y la clasificación de estas apps en malware o benignware.

3.2.1 Shallow learning entrenando modelos haciendo uso de todas las características

Creación de varios sistemas haciendo uso de diferentes tipos de algoritmos clasificadores. Será necesario entrenar mediante aprendizaje tradicional modelos de aprendizaje máquina con el dataset para que cada sistema sea capaz de realizar clasificaciones. Se estudiará la precisión, exactitud y sensibilidad de cada modelo entrenado.

3.2.2 Shallow learning entrenando modelos haciendo selección de características

Creación de varios sistemas haciendo uso de diferentes tipos de algoritmos clasificadores y técnicas de selección de características. Dado que no todas las características tienen igual importancia a la hora de determinar la clasificación final, es conveniente hacer una selección de algunas características para aumentar el rendimiento del sistema. Los modelos se entrenarán con el dataset eliminando de él varias características con diferentes técnicas de selección de características. Finalmente, se evaluarán las métricas de exactitud, precisión y sensibilidad de los sistemas que resultan al combinar cada técnica de selección de características con cada modelo.

3.2.3 Deep learning

Creación de un sistema mediante aprendizaje profundo. Esto implica la creación de una red neuronal y el entrenamiento de la misma empleando el dataset. Será necesario hacer uso de una clase de algoritmos ideados para este tipo de aprendizaje. Por último, al igual que en los demás objetivos, se evaluará la exactitud, precisión y sensibilidad del sistema creado.

4 BENEFICIOS

En este apartado se describen los principales beneficios técnicos, económicos y sociales del proyecto.

4.1 TÉCNICOS

- Hoy en día casi todo usuario tiene instalado en su ordenador algún detector de malware, mejor conocido como antivirus. Sin embargo, no ocurre lo mismo cuando se habla de smartphones. Al tratarse este proyecto de la creación de sistemas para proporcionar seguridad en smartphones supone una innovación en cuanto a tecnologías empleadas para la seguridad en estos dispositivos.

- A consecuencia de dicha innovación en cuanto a la seguridad, se mejorará el funcionamiento de los smartphones, ya que el usuario no tendrá instaladas aplicaciones no recomendadas. Así, el teléfono no estará cargado de adware y malware, responsables de ralentizar el dispositivo, por lo que el rendimiento de este será mayor.

- Por otra parte, puede servir como API para desarrollo de apps que dependan de otras para complementar su funcionamiento. Podrán comprobar si la aplicación de la que van a hacer uso se trata de software seguro. Por ejemplo, aplicaciones que necesitan representar mapas y quieran hacer uso de otras aplicaciones que proporcionen mapas, tales como Google Maps, podrán verificar que Google Maps es segura y puede ser usada para complementar la aplicación principal. Por el contrario, el uso de una aplicación que se complemente con otra maligna conllevaría a resultados indeseados.

4.2 SOCIALES

- Indudablemente, el mayor impacto de este proyecto es en la sociedad. La mayoría de la población cuenta con al menos un dispositivo con sistema operativo Android. Cada día son instaladas miles de aplicaciones, con lo cual, la probabilidad de instalar aplicaciones que contengan software malicioso es elevada. Sin embargo, gracias a estos sistemas se puede saber de antemano si es seguro instalar la aplicación que se desea. Esto trae un alto beneficio sobre los usuarios, ya que evitan sufrir ataques como acceso a información privada que casi todo usuario almacena en su móvil, accesos no debidos a la cámara, control remoto de su dispositivo… Al poder evitar todo esto, el usuario puede disfrutar de un servicio más seguro.

- Aparte de un buen servicio para el usuario, también aporta grandes beneficios en lo que respecta a denuncias, juicios y condenas. Cuantas más barreras se le pongan al atacante, menos posibilidades tendrá para atacar. Esto disminuirá muchas de las denuncias debidas a ataques a cualquier usuario Android. Además de reducir juicios debido a acceso o modificación de datos importantes de empresas. Todo esto hará que la ciberdelincuencia dirigida a dispositivos Android decrezca.

4.3 ECONÓMICOS

- Muchas veces, el ciberdelincuente infecta el dispositivo de manera que tiene la capacidad de bloquearlo remotamente y encriptar los archivos quitando el control de toda la información y datos almacenados al usuario. El virus lanza una ventana emergente que solicita al usuario el pago de un rescate, dicho pago se hace en monedas virtuales. Esto es lo que se conoce como *ransomware*. Si se consigue disminuir este tipo de ataques, se conseguirá un beneficio económico. De esta manera, se evitarán pagos de víctimas, los cuales dependiendo de la información bloqueada pueden dar lugar a miles de euros. Cuantas menos probabilidades tenga por ejemplo una empresa de sufrir este tipo de ataque, mejor funcionará económicamente esta, ya que el dinero que se guarda para imprevistos podrá ser empleado en otro tipo de incidentes.

- Además, esto contribuirá positivamente a las empresas que ofrezcan aplicaciones seguras, ya que, si tenían competencia con alguna app portadora de software malicioso, esta disminuirá considerablemente. Dicho esto, tendrán mayores usuarios que harán uso de su app, con lo cual contribuirá positivamente a la economía de la empresa.

- Finalmente, junto al beneficio técnico de un mejor funcionamiento de los dispositivos, se tiene un beneficio económico. Los dispositivos que han sido infectados, se ralentizan y no consiguen realizar las funciones como debían. Esto hace que el usuario no disfrute de un buen servicio y acabe decidiéndose por comprar otro. Esto supone un gasto injusto para los usuarios infectados y acaba pareciéndose a la filosofía de la obsolescencia programada, comprar, tirar, comprar. Con lo cual, si el usuario tiene un dispositivo el cual presenta un buen rendimiento, no necesitará renovarlo cada tan poco tiempo.

5 ESTADO DEL ARTE

Las 3 áreas en las que está enfocado el proyecto son Machine Learning, Deep Learning y Big Data. Las dos primeras hacen referencia a las técnicas empleadas para el aprendizaje(learning) del sistema. Por otro lado, Big Data se define como el modelo de ingesta, almacenamiento y explotación de la información, es decir, todo lo necesario para trabajar con una gran cantidad de datos.

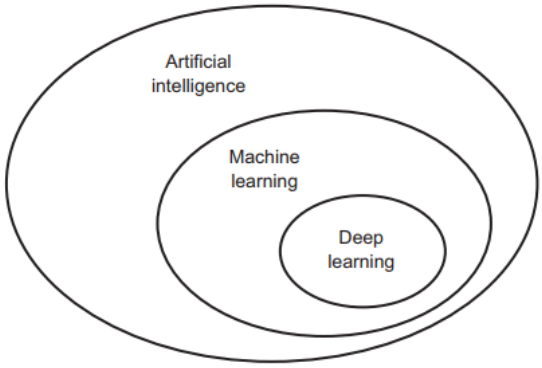

Ilustración 2: Áreas de la inteligencia artificial empleadas en este proyecto

Tal y como se puede observar en la Ilustración 2, las dos primeras áreas del proyecto mencionadas son a su vez subcampos de la Inteligencia Artificial. Por este motivo, para poder entender mejor los dos aprendizajes se comentará brevemente en qué consiste la IA.

5.1 INTELIGENCIA ARTIFICIAL

La Inteligencia Artificial es la simulación de procesos que realiza la inteligencia humana en máquinas como sistemas informáticos. Es decir, una "máquina inteligente" es aquella que imita las funciones cognitivas como percibir, razonar, aprender y resolver problemas. Por lo que esta simulación de procesos implica un previo aprendizaje, razonamiento y autocorrección. Para poder cumplir estas tres funciones, se ayudan de la combinación de algoritmos.
Actualmente se pueden encontrar muchos ejemplos de tecnología de IA como procesamiento del lenguaje natural, robótica, automatización, visión por computador y Machine Learning. Esta última es de la que se ha hecho uso en este proyecto.

5.2 MACHINE LEARNING

5.2.1 Introducción

El aprendizaje máquina, en inglés, Machine Learning nace como una disciplina de la Inteligencia Artificial. Esta disciplina es capaz de crear sistemas que pueden aprender automáticamente basándose en millones de datos, identificar patrones y tomar decisiones con mínima intervención humana. Al final, el sistema creado es un algoritmo, ya que revisa datos y es capaz

de predecir comportamientos futuros, es decir, mediante una serie de instrucciones representa la solución a un problema. El propósito del Machine Learning es que las máquinas sean capaces de aprender como un humano lo haría. Este aprendizaje máquina permite que los sistemas mejoren de manera autónoma con el tiempo.

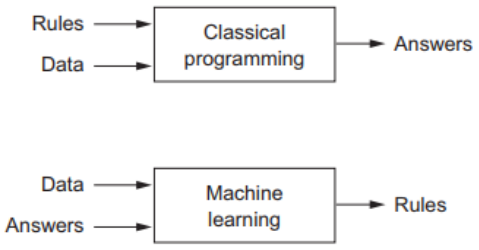

Ilustración 3: Diferencia entre el clásico paradigma y el basado en machine learning

Como se puede observar en ilustración 3, hay diferencias entre los dos paradigmas, la principal es que ahora no se programan los sistemas, sino que se entrenan. Las reglas obtenidas por Machine Learning pueden ser aplicadas a nuevos datos para producir respuestas automáticamente generadas por el sistema en base a las reglas que el sistema aprendió y no por reglas codificadas por los programadores.

5.2.2 Funcionamiento

El principal objetivo es que los sistemas desarrollen la capacidad de generar y asociar, es decir, que sean capaces de realizar sus funciones con precisión y exactitud tanto en casos familiares como en nuevos o imprevistos. Esto se consigue haciendo que formen modelos que generalicen la información que se les presenta para realizar sus predicciones. La información que se le presenta son grandes cantidades de datos, mejor conocido como big data, pero el sistema los percibe como casos, ejemplos prácticos que le ayudan a aprender y clasificar.

5.2.3 Tipos de Machine Learning

El Machine Learning abarca un campo bastante amplio que puede dividirse en tres subcampos diferenciados en base a la función de los tipos de algoritmos empleados. En la ilustración x se pueden observar algoritmos pertenecientes a cada tipo de aprendizaje.

Ilustración 4: Tipos de algoritmos empleados para cada subcampo de Machine Learning [8]

Para empezar, un subcampo es el aprendizaje supervisado o ***Supervised Learning***. Este tipo de aprendizaje depende de datos previamente etiquetados. Consiste en una técnica que sirve para deducir predicciones a partir de los datos empleados para el entrenamiento. Estos datos suelen ser pares de vectores cuyos componentes son por una parte los valores de entrada para entrenar a la máquina y por otra los valores de salida, es decir, los resultados deseados. Lo normal es que estos vectores sean colocados por las personas, ya que tiene que tratarse de datos(vectores) con total certeza.

Lo que se pretende es enseñar a una máquina que diversos casos (combinación de características) afirman o niegan un resultado final. Se trata de casos que ya se han resuelto pero que seguirán existiendo en el futuro, por ello, conviene que las máquinas aprendan de ejemplos y puedan hacer las predicciones sin que los humanos tengan que volver a introducir información.

Aplicaciones del aprendizaje supervisado son reconocimiento de voz, de escritura, faciales o detección de spam, malware.

En segundo lugar, se encuentra el aprendizaje no supervisado, en inglés, ***Unsupervised Learning***. En esta técnica, a diferencia de la anterior, los datos que se le pasan a la máquina para aprender ya no son vectores con dos componentes. Ahora, sólo hay datos de entrada, no se le pasan etiquetados a la máquina, por lo que esta no cuenta con ninguna indicación previa. Sin embargo, se le pasan una gran cantidad de características propias de un objeto final y es ella encargada de determinar la estructura de los datos, es decir, saber qué es a partir de las características recopiladas. Los algoritmos utilizados en este aprendizaje suelen resultar útiles para disminuir la dimensionalidad de los datos o para dividir un conjunto de objetos en subconjuntos de objetos similares entre sí, mejor conocido como *clustering*.

Entre otras aplicaciones del aprendizaje no supervisado se pueden encontrar detectar morfología en oraciones y clasificar información.

Finalmente, el último subcampo del *Machine Learning* es el aprendizaje por refuerzo o ***Reinforcement Learning***. Los algoritmos pertenecientes a este subcampo se basan en aprender en base a pruebas y errores. La máquina conoce de antemano los resultados, sin embargo, no sabe cuáles son las mejores decisiones para llegar a ellos.

El proceso para comprobar si se ha tomado la decisión correcta se puede hacer mediante la retroalimentación con el entorno o recibiendo respuestas del sistema. Esta última manera de comprobación se basa en hacerle saber a la máquina cuando toma una decisión es la correcta o no. Se puede decir que a la máquina se le premia para que aumente la probabilidad de que ocurra una decisión correcta mientras disminuye la probabilidad de que ocurra la incorrecta. De esta manera, el sistema asociará los patrones de éxito y los irá repitiendo hasta perfeccionarlos. Este entrenamiento permitirá al sistema tomar buenas decisiones y asemejarse cada vez más al aprendizaje humano, ya que irá mejorando continuamente el aprendizaje y su efectividad de predicción.

Ejemplos en los que se emplea la técnica de aprendizaje por refuerzo pueden son navegación de un vehículo en automático, toma de decisiones…

5.2.4 Perspectivas y aplicaciones actuales

A día de hoy, Machine Learning facilita muchas tareas cotidianas y puede ser aplicable a cualquier campo siempre y cuando existan suficientes datos y el problema planteado en ese campo sea solucionable. Algunos ejemplos conocidos son buscadores en Internet, clasificación de secuencias de ADN, diagnósticos médicos u optimización e implementación de campañas digitales publicitarias.

Respecto a la última aplicación mencionada, se puede hablar de Adext, el primer y único AMaaS que aplica Inteligencia Artificial y Machine Learning a la publicidad digital. Tiene como objetivo encontrar la audiencia perfecta para cualquier anuncio en plataformas como Google AdWords, Facebook e Instagram [5].

5.3 DEEP LEARNING

5.3.1 Introducción

El aprendizaje profundo, mejor conocido como *Deep Learning* es un área del Machine Learning que puede ser tanto de tipo Supervised Learning como Unsupervised Learing. Este tipo de aprendizaje emplea algoritmos basados en estructuras lógicas similares a la organización del sistema nervioso de los seres humanos. De esta manera, se forman capas de unidades de proceso (neuronas artificiales) cuya principal función es detectar determinadas características que poseen los objetos percibidos. Estas estructuras lógicas son conocidas como redes neuronales.

SHALLOW LEARNING	DEEP LEARNING
Buenos resultados con pequeños datasets	Necesidad de grandes datasets
Rapidez en el entrenamiento de modelos	Computacionalmente intensivo
Necesidad de probar diferentes características y clasificadores para lograr mejores resultados	Aprende las características y los clasificadores automáticamente
La precisión está limitada	Precisión ilimitada

Tabla 1: Principales diferencias entre aprendizaje tradicional y profundo [9]

5.3.2 Funcionamiento Redes Neuronales

Las redes neuronales tienen como objetivo imitar el funcionamiento de las redes neuronales de un ser vivo. Estas redes están formadas por un conjunto de neuronas conectadas entre sí y que trabajan en conjunto, ninguna de ellas tiene una tarea determinada que realizar. A medida que pasa el tiempo, las neuronas van mejorando conexiones y lo aprendido se queda fijo en el tejido. Se basan en recibir parámetros, combinarlos de una forma correcta y predecir un resultado. La cuestión es que no se sabe cómo combinarlos (entrenar a la red) y aplicarlos (predecir) al mismo tiempo.

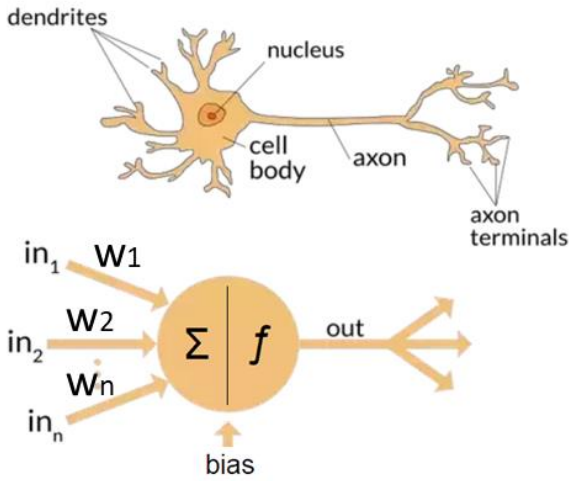

Ilustración 5: Neurona real y su representación en una red neuronal creada con Deep Learning

En la ilustración 5 se puede observar una neurona del cerebro humano y su representación en la red neuronal empleada en Deep Learning. Tiene n entradas in_1, in_2, …, in_n y cada una de ellas su respectivo peso w_1, w_2, …, w_n. Si la suma de las entradas multiplicada por sus pesos es mayor que un número, la salida de la neurona será 1, por el contrario, será 0. Estos pesos son lo que desconoce la red y lo que hay que averiguar mediante el entrenamiento de la red.

En lo que respecta a las capas de la red, estas tienen una gran importancia. Estas capas son necesarias ya que las salidas de una neurona pueden depender de las salidas en las anteriores. Por ejemplo, si se quiere saber el porcentaje de un examen teórico en la nota final de un alumno, pero hay una condición que dice que si se suspende la parte práctica la nota teórica no cuenta. Para esto son necesarias las capas, para tener una neurona intermedia que diga si la práctica está aprobada o no y contar eso en la neurona de salida. Con las capas se consigue añadir información que antes no existía ya que cada una aprende a encontrar las características que más contribuyen en la clasificación de los datos.

Traductores inteligentes, reconocimiento de voz, visión computacional, detección de virus y robótica son algunas de las aplicaciones más usadas del Deep Learning.

5.3.3 Proceso de aprendizaje de una red neuronal

El proceso de aprendizaje de una red neuronal se basa en aprender los valores de pesos w_i de cada característica de entrada cada neurona y el valor de salida de esta. El entrenamiento se empieza con dos pesos aleatorios y se ve qué resultado da la red. Ese resultado se comprueba con el real y si falla, se van ajustando los pesos poco a poco.

Esto se consigue siguiendo un proceso iterativo de ida, estimación y vuelta" por las capas neuronales. La fase de ida del proceso es llamada *forward propagation*, la de estimación, *loos* y la de vuelta *back propagation*.

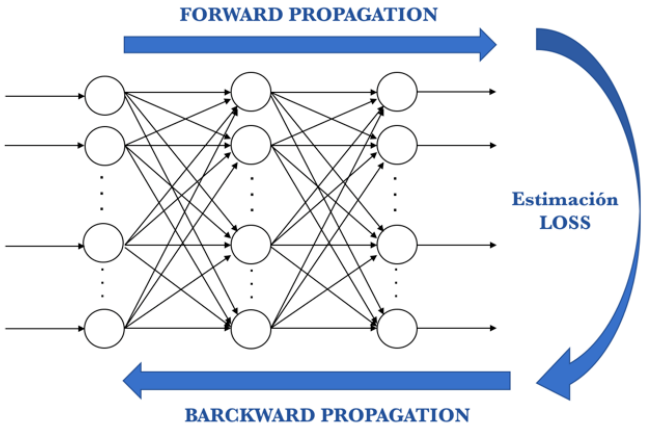

Ilustración 6: Fases de aprendizaje de una red neuronal

- **Forward Propagation:** En esta fase se le pasan datos de entrenamiento a la red y estos atraviesan la red neuronal para calcular sus predicciones. Todas las neuronas aplican la su transformación a la información que reciben de las neuronas procedentes de la capa anterior y se la envían a las de la siguiente capa. Una vez que los datos hayan atravesado todas las capas se alcanzará la capa final con un resultado de predicción.

- **Función Loss:** Una vez acabada la fase de forward propagation, se usa una función para estimar los errores y comparar los resultados predichos con los reales. Lo que se pretende intentar hacer un sistema lo más óptimo posible, es decir, que la variación entre las predicciones y los datos reales se aproxime a 0.

- **Back Propagation:** Cuando se ha calculado la calidad de la predicción se propaga hacia atrás esta información. Es decir, se propaga por todas las neuronas desde la capa de salida a la de entrada. Cada neurona va procesando la cantidad que le llega de la información de *loss* información y ajusta los costes en base a la información recibida. Para ello, se ayuda de diferentes técnicas de optimización (*gradient descent, Adam optimizar…)*. Lo que hace es ir cambiando poco a poco los pesos en base a los resultados obtenidos por *loss*. Esto ayuda a ver en qué dirección se acerca a conseguir el mínimo lo

Hay que ir realizando estos procesos mediante varias iteraciones para alcanzar un buen sistema de predicción. Esto se va realizando en porciones de datos (batches) en las continuas iteraciones (epochs) de todos los datos que se le pasan a la red neuronal en cada iteración.

5.4 TRATAMIENTO DATOS

Las escalas del dataset empleadas para este proyecto resultan pequeñas y pueden ser tratadas con recursos modestos. Sin embargo, una implementación real, requeriría emplear infraestructura sumamente paralela en cuanto a dato, lo que derivaría en el uso de herramientas Big Data.

El término Big Data engloba las infraestructuras, tecnologías y servicios que han sido creados para gestionar analizar y procesar grandes volúmenes de datos que no pueden ser tratados de manera tradicional. Esto se debe a que las herramientas de software normalmente utilizadas, para capturar, gestionar y procesar datos no son capaces de tratar con estas cantidades, un ejemplo de este tipo de herramienta serían las BBDD.

Una herramienta de Big Data de posible uso sería el sistema Hadoop. Se trata de un sistema de código abierto que se utiliza para almacenar, procesar y analizar grandes volúmenes de datos.

6 METODOLOGÍA

La parte técnica del proyecto consiste en la creación de tres sistemas óptimos con diferentes enfoques que permitan detectar si una aplicación para el sistema operativo Android es considerada malware o no. Todos ellos serán capaces de realizar dicha detección. Sin embargo, cada uno va a satisfacer diferentes intereses del usuario. De manera que uno será el sistema que presente más exactitud, otro, el de mayor precisión y, por último, otro con mayor sensibilidad. Posteriormente, en el apartado 6.2.3 se explicarán más detalladamente estos conceptos y su impacto en el usuario.

Se trata de un proyecto software, por lo tanto, gran parte del proyecto se basa en la programación. Esta se va a desarrollar haciendo uso del lenguaje de programación Python y Spyder como entorno de desarrollo. Se ha elegido Python porque es uno de los lenguajes de programación más utilizados para IA. [10] lo considera ideal por la simplicidad y sus lógicas de DRY y RAD. En ese caso su uso va a ser para generar algoritmos de IA, ya que cuenta con abundantes librerías para ello.

En este apartado se expone la metodología seguida para la creación de dichos sistemas siguiendo el orden cronológico en el que se ha ido desarrollando. Para una correcta organización, el proyecto está dividido en dos partes principales, independientes entre sí. Ambas partes tienen la misma meta, que es la creación de los sistemas que se ha mencionado anteriormente y analizar la precisión de este. La diferencia entre ellas es el aprendizaje empleado. En la primera parte el aprendizaje se hace mediante el entrenamiento de modelos clasificadores empleando aprendizaje tradicional, conocido como Shallow Learning y la segunda es el entrenamiento de una red neuronal empleando Deep Learning. Primero se explica de donde se ha obtenido la información empleada en los dos aprendizajes y en que consiste esta.

6.1 OBTENCIÓN DE INFORMACIÓN

Para entrenar tanto a los modelos como a la red neuronal es necesario contar información sobre el tema de interés. En este proyecto la información ha sido obtenida de [1]. AI+DA proporciona el dataset OmniDroid, empleado para todos los entrenamientos de los diferentes sistemas. Se trata de un conjunto de datos de referencia de características dinámicas, estáticas y pre-estáticas de aplicaciones Android y ha sido construido mediante AndroPyTool. Esta es una herramienta que extrae automáticamente características estáticas y dinámicas de un conjunto de aplicaciones Android.

6.1.1 Descripción del dataset

El dataset empleado es un fichero csv estructurado de la siguiente manera. Las filas interpretan las diferentes apps de las cuales se han obtenido las características con AndroPyTool, es decir, ejemplos reales. Por otro lado, las columnas representan diversas características de aplicaciones, salvo una columna que es la responsable de indicar si cada aplicación(fila) con sus características es MW o BW. Estos datos son hechos reales al igual que su clasificación en malware o benignware. A continuación, se describen los grupos principales que engloban a las características que más contribuyen a la clasificación de aplicaciones y Android como pueden contribuir a que una aplicación sea maligna. No se han mencionado todos los grupos para no resultar pesada la lectura de la memoria, se debe contemplar que existen más grupos en el dataset a parte de los siguientes a pesar de no ser explicados.

- **Permisos de aplicación:** Son permisos que solicitan las apps para acceder a contenido sensible del usuario (contactos, cámara). Puede haber dos tipos de permisos, los normales, los que el sistema concede automáticamente y los peligrosos, los cuales se le solicitan al usuario aprobarlos. Dependiendo de la versión de Android, los permisos se notifican a la hora de ejecutarlos o a la hora de instalarlos. La razón por la que es importante proporcionar información sobre los permisos de las aplicaciones es porque aplicaciones maliciosas abusan de permisos. El objetivo de ellas es acceder a información del usuario o controlar remotamente los dispositivos de manera que lancen aplicaciones o sitios web de phishing.

  Por ejemplo, [11] explica como troyanizar un dispositivo Android mediante Inyección manual. Consiste en inyectar payload malicioso en el AndroidManifest.xml. Este es el archivo en el que albergan los permisos normales.

- **Opcodes:** Son los códigos de operación, es decir la parte de una instrucción de lenguaje máquina que indica la operación que debe ser realizada. Sin embargo, existen opcodes ilegales en aplicaciones Android. Una de las funciones de estos opcodes ilegales es el cracking, que en el caso de apps móviles consiste en la desprotección de ellas (normalmente de pago) para que puedan ser usadas sin límite.

  Tal y como se explica en [12], la mayoría de dispositivos Android infectados son consecuencia de la instalación de aplicaciones crackeadas proporcionadas por terceros, no Google Play. Es por ello, que los opcodes se consideran una característica importante a tener en cuenta para saber si una aplicación contiene software que infecte a los dispositivos.

- **Llamadas a APIs**: Son las llamadas a APIs que realizan las aplicaciones para evitar tener que programar funciones ya hechas, las cuales nos pueden proporcionar las APIs. Mirando las llamadas a APIs que hacen que las apps realicen ciertas operaciones se puede detectar malware. Para ello, se examinan los ficheros en los que viene escrito el diseño de una app para que realice sus funciones, aquí se incluyen las llamadas a APIs.

  Hay muchos ejemplos de cómo realizar llamadas a APIs no debidas, [13] explica un ejemplo. Además, cuenta que son muy típicas la API de descarga URLDownloadToFile, la cual descarga

archivos no debidos. Por otra parte, la API GetWindowDC es característica de grabadores de pantalla típicamente vistos en spyware y keyloggers. Spyware es un programa espía que recopila información de un dispositivo y la transmite a una entidad externa sin que el usuario se dé cuenta. Por otro lado, los keyloggers se basan en realizar un segumiento y registrar cada tecla que se pulsa en un dispositivo sin el permiso ni conocimiento del usuario.

- **Comandos del sistema**: Estos comandos ofrecen una descripción de las acciones que ocurren a nivel bajo, por lo que pueden revelar interesantes acciones desarrolladas por el sistema. Es importante contemplarlo como característica importante de apps porque se dan muchos ataques de inyección de comandos malignos. Lo que hace el atacante mediante estos comandos es extender la funcionalidad por defecto de la aplicación, la cual ejecuta comandos de sistema sin necesidad de inyectar código.

  Estos ataques suelen ocurrir debido a la insuficiencia verificación de entrada y cuando una aplicación pasa de manera insegura datos suministrados por el usuario, como cookies, formularios… a un intérprete de comandos. La ejecución de estos comandos suele ser ejecutados con privilegios de la aplicación vulnerable.

- **Actividades:** Una actividad (activity) es un componente de la aplicación que presenta una pantalla a los usuarios para que interactuando con ella realicen una acción, es decir, el interfaz usuario.

  Según cuenta [14] la mayoría de apps maliciosas no incluyen ninguna actividad, no tienen interfaz gráfica.

- **Servicios:** Son elementos de las aplicaciones que son capaces de realizar operación de larga ejecución en segundo plano y que, a diferencia de las actividades, no proveen una interfaz de usuario. Los servicios pueden operar con transacciones de red, reproducir música, realizar I/O de archivos o interactuar con un proveedor de contenido, todo en segundo plano.

  Al hacerlo en segundo plano, el usuario no se da cuenta y muchas aplicaciones cuentan con espía de servicios para captar información que se intercambia en estos servicios.

- **Receivers:** Son los componentes que se dedican a recibir y responder a los eventos generados por el sistema, como avisos de baja batería, un mensaje recibido o enviado.

  Estos elementos suelen tener aplicados permisos de seguridad, sin embargo, pueden ser modificados en archivo de manifestó de la aplicación y ocasionar fallos como que no reaccionen ante eventos que deberían ir seguidos de una notificación.

- **Paquetes API**: Los paquetes API son mecanismos de agrupación y organización de APIs. En este caso, los paquetes se usan para agrupar las anteriormente mencionadas llamadas a APIs. Así en estos paquetes están definidas estas llamadas para describir patrones de características generales. Los más habituales son Android.app y java.lang.

- **FlowDroid:** Es una herramienta de código abierto para el análisis estático de aplicaciones Android. Es capaz de detectar flujos de datos intra-procesados. En el dataset lo que se muestra son resultados obtenidos tras el análisis con FlowDroid de cada app.

6.2 PRIMERA PARTE: SHALLOW LEARNING

Para realizar un sistema/máquina que aprenda y posteriormente sea capaz de dar respuesta a los problemas planteados se han de tener en cuenta varios procesos a la hora de programar. A continuación, se muestran los pasos que se han de seguir en un diagrama de estado.

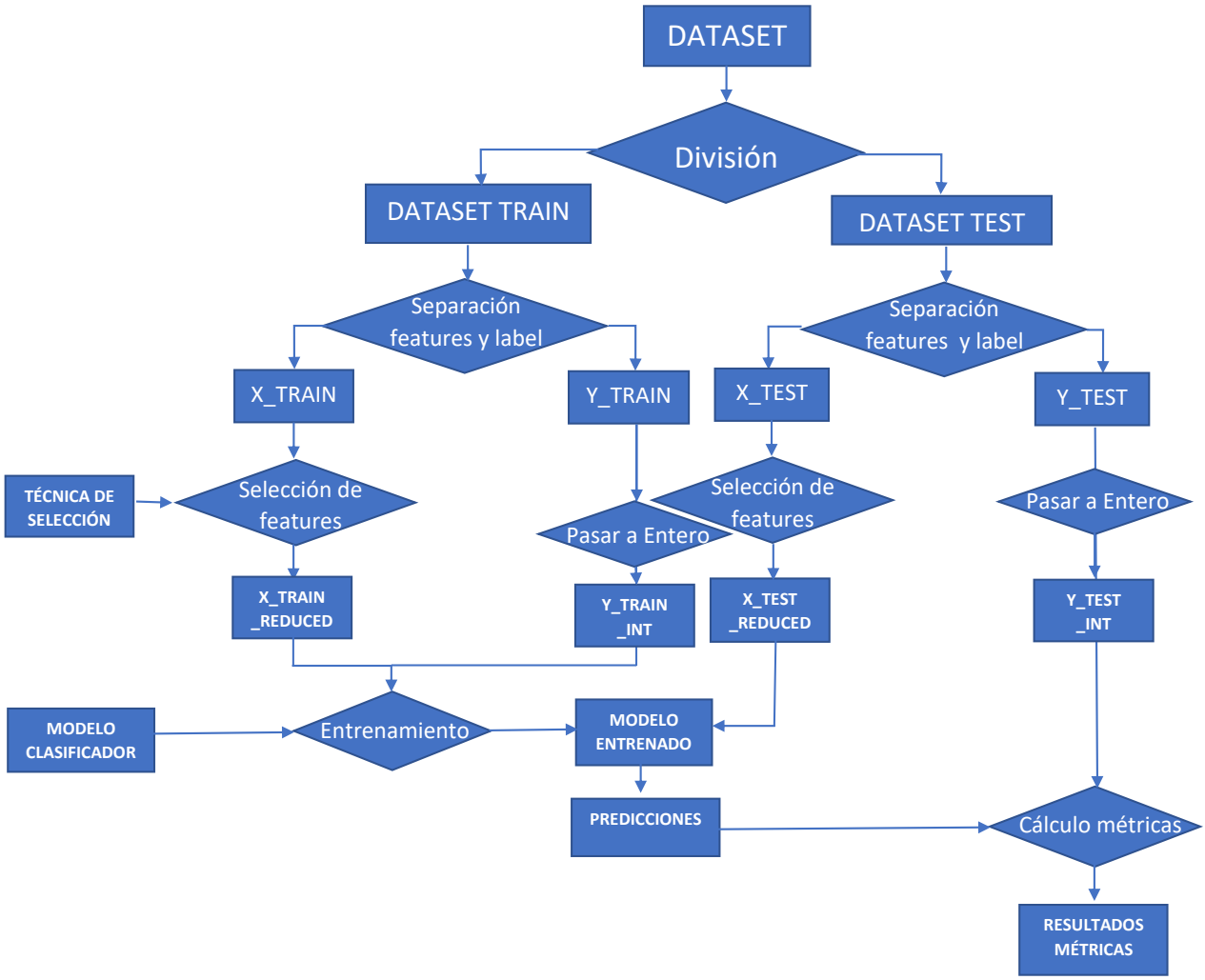

Ilustración 7: Diagrama de estados Shallow Learning del proyecto

6.2.1 Volcado de la información del dataset a Python y modificación de ella

Para poder trabajar con la información en Python lo primero que se debe hacer es leer el fichero csv que alberga los ejemplos para el entrenamiento. De esta manera se tendrá en el programa un dato de tipo dataframe con todos los datos que había en el fichero, distribuido en las mismas filas y columnas. Un dataframe consiste un tipo de dato con estructura bidimensional, de tamaño variable, con datos heterogéneos distribuidos en una tabla con filas y columnas etiquetadas. Se debe mencionar que este proceso tarda un tiempo, ya que el fichero tiene un peso de 1 GigaByte.

Una vez creado este dato en el programa, se divide en otros dos dataframes con distintos números de filas. Esto se hace para tener un conjunto de datos para entrenar al sistema (el que contiene más filas) y otro para testeo, es decir, para tener casos que se le pasen al sistema y que este realice su función de predicción. Esta división la hace una función de la librería Sk-Learn.

Como se va a realizar un entrenamiento de Machine Learning supervisado, para entrenar al sistema es necesario pasarle las características de las aplicaciones por un lado y por otro la etiqueta, es decir, si es MW o BW. Por lo que se extrae de cada uno de los dos dataframes la columna referida a dicha etiqueta y se obtienen así dos arrays adicionales con la información de esa columna. Uno de ellos para el entrenamiento y otro para que cuando el sistema realice predicciones se puedan comparar los datos predichos con los reales. Así se podrán comprobar métricas como efectividad y sensibilidad del sistema.

6.2.2 Entrenamiento

6.2.2.1 Entrenamiento de un modelo clasificador

Para poder realizar predicciones, primero hay que entrenar al sistema, para ello se ha elegido primero la creación de un modelo clasificador para poder entender su comportamiento. Este entrenamiento se hace con el dataframe sin la columna de etiquetas y el array con los datos de la columna de etiquetas.

6.2.2.2 Selección de características

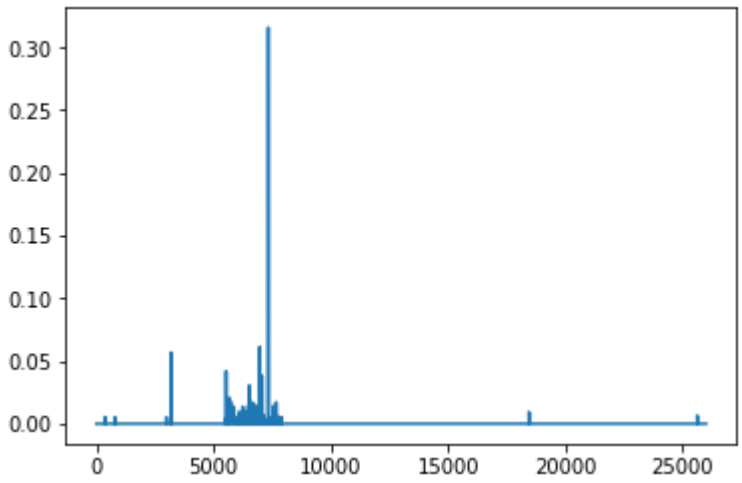

Ilustración 8: Importacia de las características

En la ilustración 8 se puede observar la importancia de cada una de las 25998 características. Para ello, se ha hecho uso de una función que proporciona la librería Sk-Learn. Dicha función proporciona un resultado que indica el valor o la utilidad de cada característica a la hora de construir un árbol. Cuantas más veces sea empleada una característica para tomar decisiones clave, mayor importancia tendrá. Como unas presentan mayor peso a la hora de determinar la predicción final se decide realizar una selección de características y entrenar al modelo con menos características, lo que hará que los sistemas sigan siendo efectivos tardando menos tiempo en crearse.

La función empleada en la ilustración 8, permite saber cuáles son las características más importantes, con lo cual, se pueden apartar aquellas características menos importantes y quedarse sólo con las de mayor importancia.

Aparte del método comentado en el párrafo anterior, existen distintos seleccionadores de características, los cuales seleccionan características con una función de sk-learn. En el proyecto, se ha optado hacer uso de seleccionadores de características distintos al método comentado de calcular la importancia y posteriormente quedarse con las más importantes. Estos realizan el proceso de selección en una sola función, por lo que se entrena al modelo desde un principio con la información reducida. Al principio sólo se hará uso de uno para entender su funcionamiento. Esta fase de selección de características se realiza sobre el dataframe elegido para el entrenamiento. Una vez habido reducido las columnas de este dataframe, se entrena el modelo tal y como se ha explicado en el apartado anterior.

Una vez efectuados estos dos apartados, ya estaría el sistema entrenado.

6.2.3 Predicción y medidas

Este apartado consiste en darle uso al sistema creado. Por lo que se le pasan al sistema los datos del dataframe de testeo para que realice las predicciones. Después se procede a la comprobación de dicha predicción. Para ello, se hace uso de varias métricas.

Matriz de confusión: es una matriz nxn en la que las filas se nombran según las clases reales y las columnas, según las clases previstas por el modelo. Sirve para mostrar de manera clara cuándo una clase es confundida con otra.

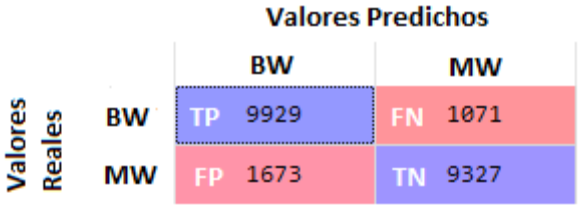

Ilustración 9: Matriz de confusión de un sistema del proyecto

La Ilustración 9 muestra una matriz de confusión hallada tras haber creado un modelo con todos los ejemplos, 22.000. Esta matriz tiene muchas interpretaciones. Una de ellas sería la siguiente: De 22.000 aplicaciones analizadas, el sistema ha clasificado 9.929+1.673=11.602 como benignas y 1.071+9.327=10.398 malignas. De las 22.000 analizadas, en 9.929+9.327=19.256 aplicaciones, el valor predicho y el real coincidían. De las 11.602 que se ha dicho que son benignas, 9.929 lo eran de verdad (TP), mientras que las otras 1.673 restantes no (FP). Por otra parte, de las 9.929 + 1.071=11.000 aplicaciones que realmente son benignas, el sistema ha clasificado sólo 9.929 de ellas en benignas, es decir, ha acertado en 9.929 de ellas.

Cada una de estas tres interpretaciones de la matriz de confusión tiene su propia métrica, son las siguientes:

- **Exactitud:** En inglés, accuracy score. Mide la exactitud, el porcentaje de acierto de los valores que ha predicho el sistema con los reales. Matemáticamente se halla así:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}}$$

- **Precisión:** Es la medida que indica la probabilidad de que, dada una predicción positiva, la realidad sea positiva también. Es decir, cuando predice algo positivo (BW) con cuanta frecuencia acierta.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Sensibilidad**: Indica la probabilidad de que, dado un valor real positivo, el sistema lo clasifique en positivo también. Es decir, la tasa de verdaderos positivos.

$$\text{Sensibilidad} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Las últimas tres métricas se aplican a cada sistema creado. Ya que en base a dichas métricas se hará la elección final de tres sistemas entre todos los creados. La librería *sklearn.metrics* cuenta con funciones que realizan estas métricas. A simple vista se podría decir que con calcular la exactitud valdría, sin embargo, en esta temática de seguridad no es suficiente sólo con ello. Esto es debido a que un sistema puede que sea bastante exacto, sin embargo, la poca exactitud que le falte para ser ideal sea fruto de una baja precisión. En este proyecto, esto se traduce a que lo que falle en el sistema para no ser cien por cien exacto sea predecir que una aplicación sea benigna cuando en realidad es maligna. Este hecho pondría en peligro la seguridad del usuario, por lo que la medida de la precisión cobra gran importancia en este proyecto.

6.2.4 Evaluación de métricas combinando modelos con seleccionadores

Tras ser capaces de deshacerse de características haciendo uso de un seleccionador de ellas, se puede hacer lo mismo con muchos de los seleccionadores disponibles y así, comprobar cuál es el mejor en cuanto a las métricas. Del mismo modo, se pueden crear más sistemas mediante la creación de diferentes modelos clasificadores, de manera que se pueda comprobar cuál de todos da mejor resultados en las métricas.

Para realizar este proceso, se decide realizar diferentes sistemas combinando de seleccionadores y modelos clasificadores. Posteriormente, se deben obtener resultados sobre las métricas de cada combinación.

	0	1	2	3	4	5
0	0.796748	0.765766	0.646707	0.801653	0.8	0.745902
1	0.885965	0.855769	0.646707	0.868421	0.863636	0.87156
2	0.853211	0.818182	0.646707	0.853211	0.858491	0.891089
3	0.838095	0.722222	0.650602	0.831683	0.831683	0.841584
4	0.863014	0.729323	0.650602	0.857143	0.867647	0.863636
5	0.830189	0.846154	0.650602	0.810345	0.836957	0.493506
6	0.65625	0.711268	0.650602	0.70068	0.696552	0.702128

Ilustración 10: Medida de la exactitud en algunos sistemas creados

	0	1	2	3	4	5
0	0.796748	0.765766	0.646707	0.801653	0.8	0.745902
1	0.885965	0.855769	0.646707	0.868421	0.863636	0.87156
2	0.853211	0.818182	0.646707	0.853211	0.858491	0.891089
3	0.838095	0.722222	0.650602	0.831683	0.831683	0.841584
4	0.863014	0.729323	0.650602	0.857143	0.867647	0.863636
5	0.830189	0.846154	0.650602	0.810345	0.836957	0.493506
6	0.65625	0.711268	0.650602	0.70068	0.696552	0.702128

Ilustración 11: Medida de la precisión en algunos sistemas creados

La Ilustración 10 muestra la medida de la exactitud de cada modelo(fila) combinado con algunos seleccionadores de características(columna). Del mismo modo, la Ilustración 11, muestra la medida de la precisión de cada modelo(fila) combinado con seleccionadores de características(columnas). En las dos imágenes los modelos son los siguientes:

0.Decision Tree Classifier
1.Gradient Boosting Classifier
2.Random Forest Classifier
3.K-Nearest Neighbors Classifier
4.C-Support Vector Classifier
5.Multi-layer Perceptron Classifier
6.Gaussian Naive Bayes Classifier

En cuanto a seleccionadores de características, los que aparecen en las Ilustraciones 10 y 11 son:

0.Variance Threshold
1.Select K Best
2.Select Percentile
3.Select Fpr
4.Select Fdr
5.Select Fwe

En este caso, estos seleccionadores han trabajado computando la función ANOVA-F. Se ha de tener en cuenta que, para la realización del proyecto, han trabajado tanto con dicha función como con chi cuadrado.

6.3 SEGUNDA PARTE: DEEP LEARNING

Para esta segunda parte, al igual que en el Shallow Learing, se quiere realizar un sistema/máquina que aprenda y posteriormente sea capaz de dar respuesta a los problemas planteados. Para ello, se han de tener en cuenta varios procesos a la hora de programar. Seguido, se muestran los pasos que se han de seguir en un diagrama de estado.

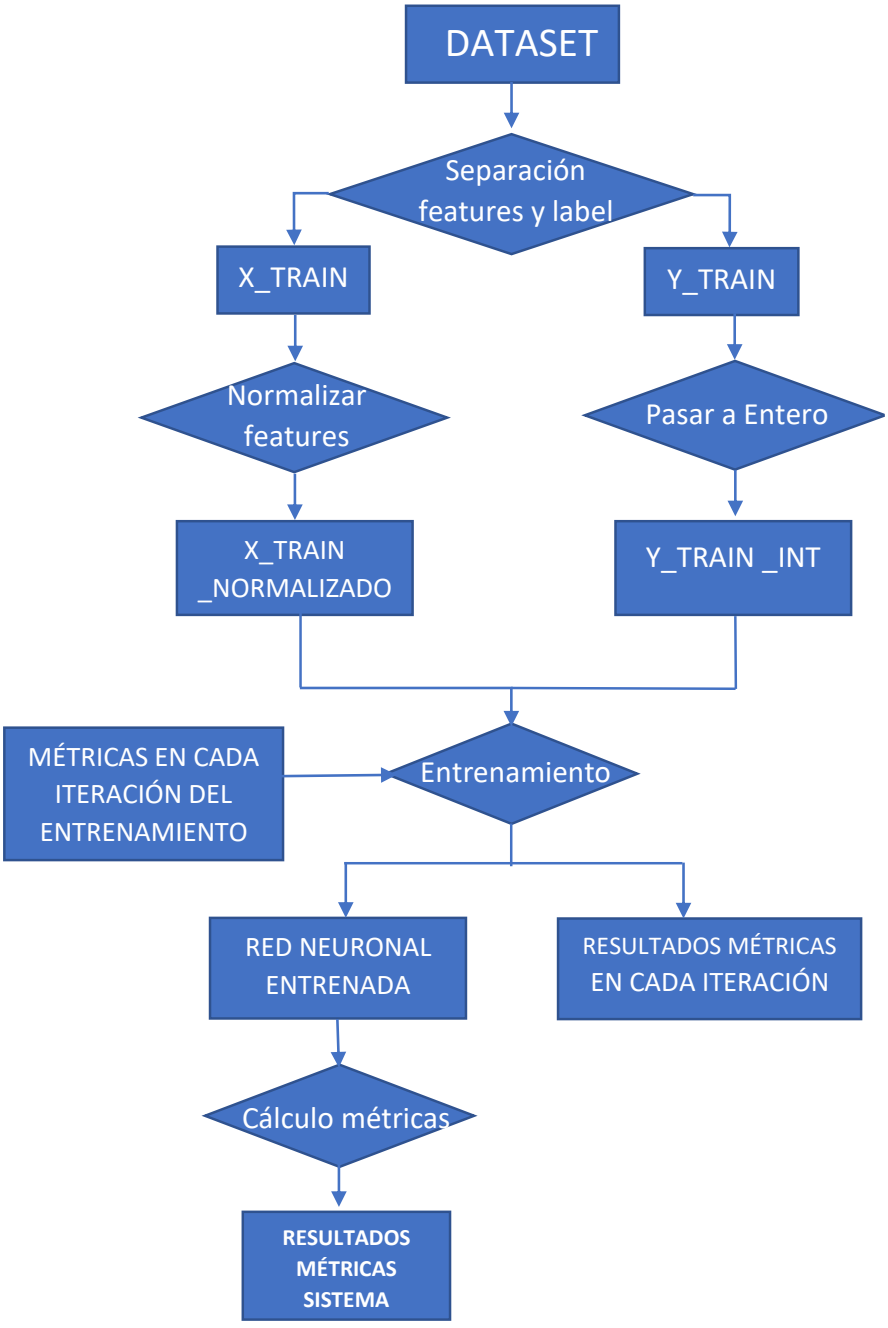

Ilustración 12: Diagrama de Estados Deep Learning

6.3.1 Volcado de la información del dataset a Python y modificación de ella

Al igual que con la primera parte, para trabajar con la información en Python se lee el fichero csv que contiene los ejemplos para el entrenamiento. Del mismo modo, se volcará en un dato de tipo dataframe.

En este caso no se divide el dataframe en parte de entrenamiento y testeo. Esto se debe a que se va a entrenar una red neuronal, y se va a comprobar su éxito con los valores que han sido empleados para su entrenamiento. Como también se trata de un entrenamiento supervisado, para entrenar a la red neuronal, hay que pasarle las características de las aplicaciones separadas de la etiqueta de cada aplicación. Este proceso es igual que en la parte de aprendizaje tradicional pero sólo se realiza en el único dataframe que se tiene.

Para entrenar a la red neuronal es necesario hacer otra modificación. Las funciones empleadas para su creación no soportan datos de tipo String en las etiquetas. Dicho esto, se deben pasar las etiquetas de BW y MW a 0 y 1 respectivamente.

La mayoría de los modelos de sk-learn esperan como parámetro de entrada una matriz con las dimensiones y las dimensiones [número de muestras, número de características]. El dataframe que se posee contiene valores que son diferentes en la escala, por lo que hay que estandarizarlos para que sigan una distribución normal. De esta manera, cada característica(columna) va a tener como media 0 y como desviación estándar 1. Para ello, se deben tipificar las columnas, esto se hace así:

Partiendo de lo siguiente:

La X es la variable aleatoria, las características de cada app.

Media: $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$ **Desviación estándar:** $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)}$

La **estandarización** se realizaría de la siguiente manera: $Z = \frac{x-\mu}{\sigma}$

La librería sk-Learn cuenta con una función que se encarga de esto. Por lo que permite estandarizar cada columna del dataframe que contiene las características.

6.3.2 Diseño y entrenamiento de una red neuronal

Después de tener ya toda la información modificada, se pasará a la creación de un sistema capaz de predecir. Un sistema creado mediante aprendizaje profundo se basa en la creación y entrenamiento de una red neuronal. Para ello, se hace uso de la biblioteca Keras de código abierto, la cual se ejecutará sobre TensorFlow.

6.3.2.1 Diseño

Primero se diseña la red neuronal. Para ello, lo primero que se hace es definir el modelo haciendo uso de una función encargada de ello, que posteriormente será la misma usada para el entrenamiento de la misma. La definición del modelo se basa en elegir con cuantas capas se quiere que cuente la red neuronal y el número de nodos de cada una.

Tras haberlo definido, se procede a definir cómo se va a compilar. En la compilación se elige la función de *loss* que se desea calcular entre los resultados predichos y los reales. En nuestro caso se ha elegido *binary cross entropy* porque tenemos sólo dos etiquetas de clasificación 0 y 1 (BW y MW). Por otra parte, *Adam* como función de optimización. Finalmente, se quiere que se obtenga la exactitud de cada iteración en red neuronal, por lo que se selecciona *accuracy* como medida.

6.3.2.2 Entrenamiento

El entrenamiento de la red neuronal se realiza mediante funciones de la librería Keras pasándole a esta el dataframe de las características y el correspondiente a las etiquetas. Este entrenamiento viene determinado por el programador. Es él quien elige las iteraciones (epochs) que se han de realizar para que la red neuronal mejore su aprendizaje, cuantas más iteraciones, mejor estará entrenada la red. Además, también se indica el tamaño de las porciones de datos sobre las que se va realizando el proceso de entrenamiento (batches). Este proceso es bastante laborioso y puede tardar en realizarse.

6.3.3 Predicción y medidas

Durante el entrenamiento, se le ha indicado a la red que proporcione la medida de exactitud que presenta la misma en cada iteración.

Una vez la red ha sido entrenada con el número de iteraciones elegidas (150) se procede a evaluar la calidad de la red creada. Para ello se ha optado por la evaluación de cuatro métricas. Por una parte, la exactitud de la red, por otra, la precisión, seguido, la sensibilidad y, por último, la matriz de confusión. Esta última se ha seleccionado simplemente para tener una visión general de las métricas. Finalmente, los resultados se almacenan en un archivo.py para cargarlos cuando se desee.

7 ANÁLISIS DE ALTERNATIVAS

El objetivo final del proyecto se puede alcanzar de bastantes maneras. Las principales formas de alcance son la creación de los sistemas mediante aprendizaje máquina o mediante aprendizaje profundo. Tal y como se ha comentado en el apartado anterior, en el caso de Machine Learning, se han creado diferentes sistemas surgidos de la combinación de modelos clasificadores y técnicas de selección de características. En lo que respecta a Deep learning sólo se ha creado un sistema con redes neuronales.

7.1 IDENTIFICACIÓN Y DESCRIPCIÓN DE ALTERNATIVAS

A continuación, en lo referente al aprendizaje máquina/tradicional, se explica por separado cada modelo y cada técnica de selección empleada. Se ha de tener en cuenta que la creación de un sistema mediante este aprendizaje implica la combinación de un modelo y una técnica de selección. Por otra parte, también hay que considerar que todos los modelos se han combinado con todos los seleccionadores de características, es decir, hay tantos sistemas creados como combinaciones.

7.1.1 Modelos

Se han elegido siete modelos clasificadores proporcionados por la librería Sk-learn.

7.1.1.1 K-Nearest Neighbors Classifier

Es un tipo de algoritmo supervisado basado en instancia. Esto último quiere decir que no aprende expresamente de un modelo, si no que memoriza las instancias de entrenamiento para posteriormente predecir. Cuando se le pasa un caso para clasificar, el método que sigue es el siguiente:

Primero, calcula la distancia entre el caso que tiene que clasificar y los demás ejemplos del dataset de entrenamiento. Después, elige los k ejemplos más cercanos y en base a la etiqueta predominante en los casos de alrededor decide la clasificación final. El número k es un dato que elige el programador, por defecto es 5. Conviene que sea un número impar para evitar que empate la predominancia y que no sea muy bajo. Sin embargo, elegir un número mayor no implica que sea mejor la precisión, pero sí que tardará más en procesar. Por lo que, para encontrar el mejor valor de k, se puede ir probando con más de un valor y ver con cual se obtienen mejores resultados.

Es posible que el peso de los k vecinos no sea igual a la hora de realizar la clasificación. La función que proporciona Sk-learn para el entrenamiento de este modelo permite elegir si todos los pesos son iguales, si disminuyen con la distancia u otra distribución definida por el programador. En la Ilustración 13 se puede observar detalladamente los fases de este algoritmo.

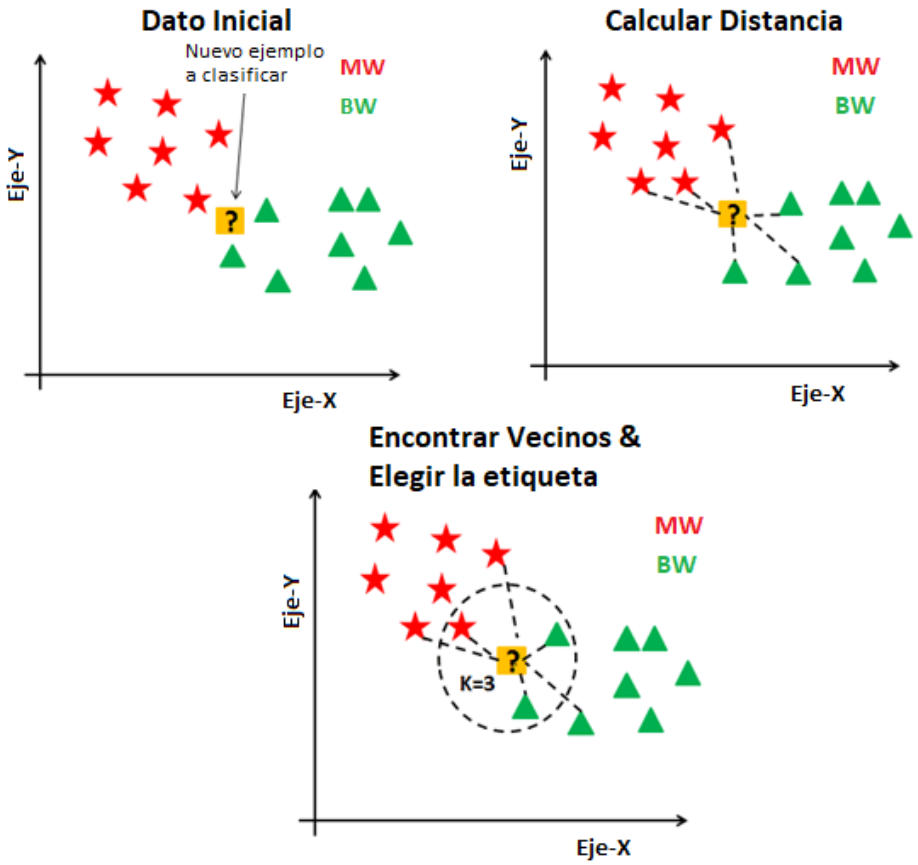

Ilustración 13: Pasos ejecución de algoritmo KNN Classifier

Se pueden apreciar ciertas **ventajas** a la hora de aplicar el algoritmo:

- Se trata de un algoritmo simple y fácil de comprender e interpretar.
- No es paramétrico, dicho de otro modo, las suposiciones que hace, no las realiza fundamentándose en la forma funcional de los datos.
- Es robusto en cuanto a valores desconocidos.

En cuanto a las **desventajas**, se encuentran las siguientes:

- Al estar basado en instancia, memoriza las instancias de entrenamiento para posteriormente predecir. Esto se traduce a que sólo hará uso de las instancias de entrenamiento cuando se le pase un caso que se quiera que clasifique.
- El tiempo de computación es bastante elevado porque va almacenando todos o casi todos los datos de entrenamiento.
- Se necesita tener bastante capacidad de memoria, ya que almacena casi todos los datos de entrenamiento.
- No funciona muy bien con grandes cantidades de datos.

7.1.1.2 DecisionTreeClassifier

Es un modelo de clasificación no paramétrico, al igual que KNN. Se trata del modelo supervisado más utilizado en machine learning y está basado en árboles de decisión. Estos son representaciones gráficas de posibles soluciones planteadas a un caso basándose en las características que este posee. Es decir, se va clasificando poco a poco en función de las características que posee y finalmente se etiqueta/clasifica. Esta clasificación se basa en decisiones de tipo IFTTT.

La estructura de los árboles de decisión es la siguiente. Cuentan con un primer nodo, el nodo raíz (*root*) y después se plantea una condición que puede ser verdadera o falsa. Dependiendo de esta respuesta, se tomará un camino u otro, los cuales han sido creados mediante la ramificación de dicho nodo en dos ramas que conducen a dos nodos distintos. Posteriormente, estos dos nodos, realizan el mismo proceso y se vuelven a bifurcar. Esto se repite sucesivamente en cada nodo hasta llegar a los nodos finales, los cuales clasifican el caso. En el caso de interés de nuestro proyecto, el árbol se irá creando en base a las características (features) de cada aplicación y su etiqueta de MW o BW. De este modo, cuando se le pase un caso nuevo que clasificar, irá pasando el caso por los sucesivos nodos en base a las características de este y finalmente decidirá si es una aplicación benigna o maligna. Cuanto más profundo sea el árbol, más compleja será la decisión y más ajustado será el modelo.

Creación óptima del árbol

Se puede llegar a cuestionar si el árbol creado al principio es realmente el mejor ya que existen miles de combinaciones posibles. En cambio, tal y como explica [15], el algoritmo de árboles de decisión devuelve el árbol óptimo para la mejor decisión fundamentándose en la probabilidad. Para ello el algoritmo va midiendo las predicciones conseguidas y las valora para comparar entre todas las combinaciones posibles y elegir la mejor. Para ello, se ayuda de las siguientes funciones.

Por una parte, el índice de Gini, que indica el grado en que los nodos están mezclados una vez divididos. Interesa que sea mínimo. Por otra, la ganancia de información, mediante la cual se estima la información que aporta cada característica. A continuación, se muestra el árbol de decisión creado en nuestro proyecto entrenado con 1000 aplicaciones en lugar de 22000:

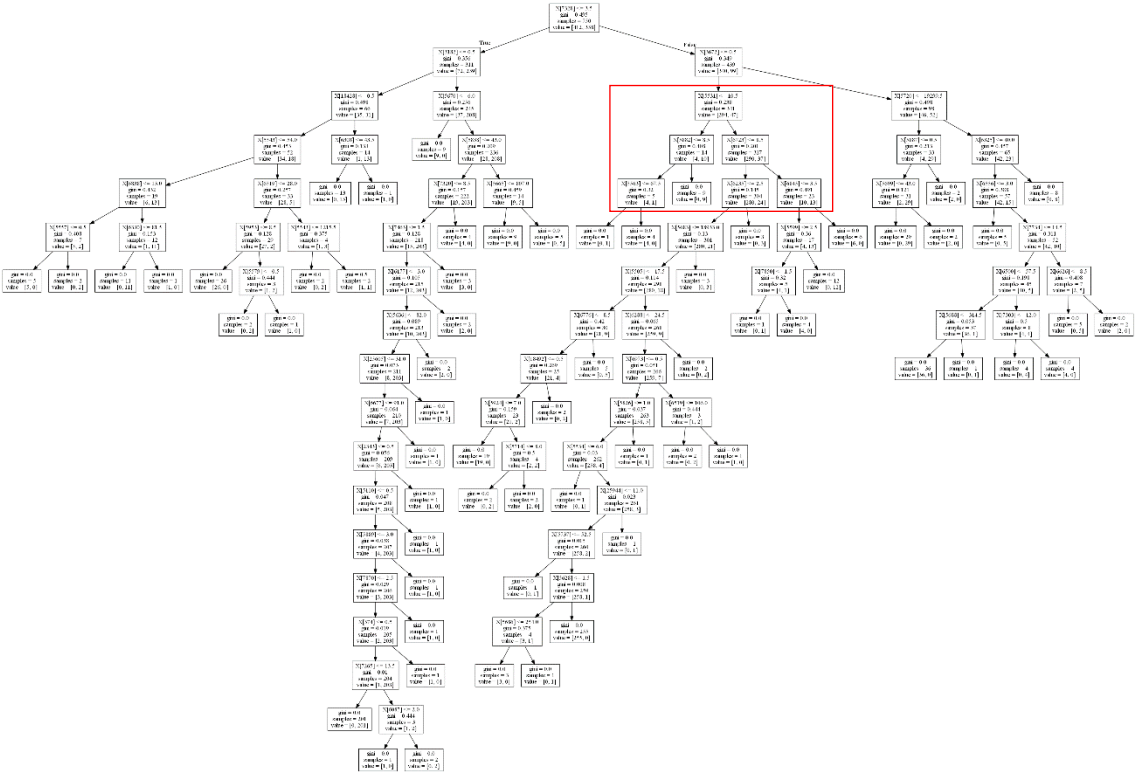

Ilustración 14: Árbol completo

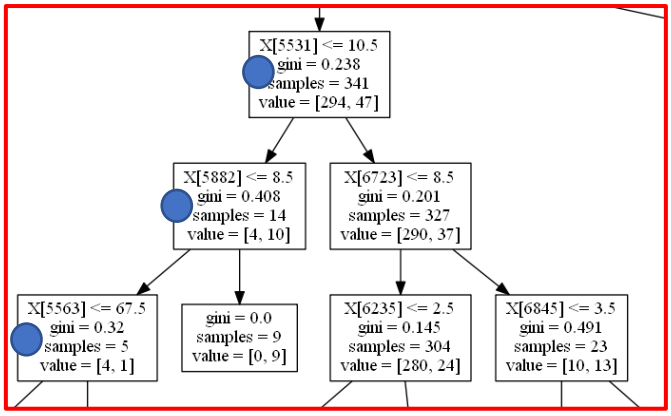

Ilustración 15: Fragmento del árbol

En la ilustración 13 se puede observar una bifurcación de un nodo, la cual se ramifica en otras. X[n] representa una característica de la aplicación. Para explicarlo mejor, se va a suponer que el primer nodo de la imagen es el *root*. Siguiendo el camino indicado por los puntos azules, una posible representación sería la siguiente: La primera caja dice que Tenemos 341 ejemplos. De esos, 294 son BW, 47 MW. La segunda caja dice que de los de los 341 ejemplos, 14 llaman a la API C, viene del true de arriba X[5531]<=10.5. Para finalizar, la última caja indica que de los 341 ejemplos, 5 tienen permiso de acceder a la cámara X[5882]<=8.5. De este modo se pueden ir haciendo suposiciones, 9 aplicaciones son BW, llaman a la API C y no acceden a la cámara. % son BW, llaman a la API C y acceden a la cámara. Y así se irían suponiendo con todos los demás caminos.

Las **ventajas** que presenta este modelo son bastantes:

- Es capaz de seleccionar hasta las características más excluidas. Analiza todas.
- Analiza todas las posibles consecuencias que implica tomar una decisión
- Clasifica la información sin necesidad de muchos cálculos
- Tolera la información incompleta o ruidosa
- Ayuda a realizar las mejores decisiones sobre la gran cantidad de datos disponible.

El principal **inconveniente** que presenta es que

- La metodología sólo asegura que cada subdivisión es óptima pero no que el árbol lo sea.
- Las interacciones de orden menor no preceden a las de mayor
- Cálculos exponenciales hacen que el problema aumente

7.1.1.3 C-Support Vector Classifier

La clasificación vectorial de soporte hace uso de máquinas de vector de soporte, las cuales son conjuntos de algoritmos de aprendizaje supervisado. Su funcionamiento consiste en relacionar datos en grandes espacios de características. Entre dichas categorías se encuentra un separador de manera que se puede ajustar la distribución de los datos para que el separador quede como un hiperplano. Este ajuste se hace con la función kernel, que puede ser lineal, polinómico, de RBF, sigmoide, precalculado u otro definido por el usuario. La primera se emplea cuando la separación lineal de los datos es sencilla, si no, se emplearían las otras. Lo mejor es experimentar para ver cual se ajusta mejor a cada problema planteado.

La librería Sk-learn proporciona una función para la creación y el entrenamiento de SVC. Por defecto emplea el kernel RBF, pero se le puede indicar el que se desee. Además, permite definir el parámetro C. Este parámetro es el encargado de indicarle al SVM el grado en que se quiere evitar clasificar erróneamente. Cuanto mayor sea, escogerá menores separadores, por lo que son de esperar mejores resultados, por defecto es uno.

Al igual que todos los modelos, presenta sus pros y sus contras, como **ventajas** tenemos las siguientes:

- Funciona bastante bien en espacios de muchas dimensiones
- Presentan una buena clasificación con nuevos datos cuando el modelo está bien parametrizado.
- El proceso de entrenamiento no depende del número de características
- El modelo final es sencillo, una combinación de vectores de soporte.

La principal **desventaja** del SVC es:

- No siempre predicen mejor que otras simples, por lo que el mayor coste de implementación y cálculo resulta un inconveniente.
- No operan bien con información incompleta o ruidosa.
- Puede resultar complicado de implementar para usuarios no especializados.

7.1.1.4 Multi-layer Perceptron Classifier

Este algoritmo clasificador hace uso del algoritmo supervisado *multi-layer perceptrón*, MLP, que entrena empleando la retro propagación. El algoritmo MLP consiste en un tipo de red neuronal artificial retroalimentada con al menos tres capas, la de entrada, la oculta y la de salida. El proceso que sigue este algoritmo es aprender una función de tipo: f (·): $R^m \rightarrow R^o$ mediante el entrenamiento en un conjunto de datos. La letra *m* representa la dimensión de entrada y *o* la de salida. Dado un conjunto de características X = x_1, x_2, …, x_m y una etiqueta clasificadora Y, es capaz de aprender una aproximación de una función no lineal para cada clasificación.

Las **ventajas** del clasificador MLP son las siguientes:

- Es capaz de aprender modelos no lineales
- Es capaz de aprender modelos a tiempo real empleando el parámetro adecuado para ello.

Entre las **desventajas** de este clasificador se encuentran:

- Al tener capas ocultas, presenta una función de pérdida no convexa en la cual existe más de un mínimo local. Esto implica que al inicializar la red con pesos aleatorios la precisión de validación del sistema creado sea diferente.
- Hay que ajustar varios parámetros antes de ponerlo en marcha, como el número de neuronas, capas e iteraciones ocultas.
- Finalmente, es sensible al escalado de características.

7.1.1.5 Gradient Boosting Classifier

La clasificación mediante la técnica de potenciación de gradiente se basa en un modelo construido de forma escalonada seguido de una generalización de *boosting.* Gracias a esto, se consigue lograr la optimización arbitraria de una función de pérdida diferenciable. Se trata de un preciso y efectivo proceso de caja, es decir, no desarrollativo y comercial. Este modelo es caracterizado por tres elementos:

El primero, una función de pérdida diferenciable a optimizar. Para nuestro caso, la entropía cruzada o la pérdida logarítmica resultan las más útiles. En segundo lugar, un algoritmo de aprendizaje débil para hacer las predicciones, el árbol de decisión. Finalmente, un modelo aditivo para añadir los algoritmos de aprendizaje débiles que minimicen la función de pérdida

Este modelo es empleado en una gran variedad de campos como la posición que ocupa una página web cuando el usuario teclea una frase en Google o en áreas como ecología. Soporta tanto clasificación binaria como multiclase, de todos modos, en este proyecto, sólo interesa la binaria, ya que la clasificación es o MW o BW, dos etiquetas.

A continuación, se pueden observar algunas **ventajas** que presenta este modelo:

- Sencillo manejo con conjuntos de datos de características heterogéneas.
- Presenta gran poder predictivo
- Es robusto en cuanto a valores desconocidos.

Sus principales **desventajas** son:

- Tiene poca escalabilidad debido a la naturaleza secuencial de *boosting*, apenas puede ser paralelizado.
- Para grandes datasets, según [16], este modelo no opera bien y recomiendan usar otro como alternativa.
- Requiere un ajuste cuidadoso de los parámetros.

7.1.1.6 Gaussian Naive Bayes Classifier

El modelo GNB es una extensión de los *Naive Bayes* para casos reales, es decir, que el etiquetado es real, usualmente asumiendo que las características de entrenamiento siguen una distribución gausiana. Un *Naive Bayes* es un clasificador probabilístico basado en el teorema de Bayes y otras suposiciones simplificadoras. Da por hecho que la presencia o ausencia de cierta característica no está relacionada con la presencia o ausencia de otra característica. Dicho de otro modo, cada característica contribuye de manera independiente en la clasificación final.

Ventajas de GNB:

- Es un modelo fácil de implementar y de poca complejidad.
- Puede ser empleado para grandes y pequeños datasets.

Inconvenientes de GNB:

- El hecho de tratarse de información real justifica en gran parte que no presenta dependencia entre variables.
- Presenta el problema de probabilidad condicionada cuando la probabilidad de una de las variables es 0.

7.1.1.7 Random Forest Classifier

Este modelo está basado en árboles de decisión aleatorios, es perturbador y combina técnicas B1998 diseñadas especialmente para árboles. Esto es, se crean diversos clasificadores introduciendo aleatoriedad en la función de construir un clasificador. La predicción del conjunto final es la media de predicción de los valores obtenidos por cada clasificador o en el caso de variables categóricas, una mayoría de los votantes. En l apartado de 8.1.1.1.2, ya se ha explicado cómo funcionan los árboles de decisión, por lo que, conociendo su funcionamiento, se puede entender el de los bosques, conjunto de árboles.

Para empezar, el modelo Random Forest empieza con la técnica de aprendizaje de los árboles de decisiones. Una vez que todos los árboles hayan desempeñado la fase de entrenamiento, son capaces de predecir. Cuando se introduce un nuevo caso a este modelo, poco a poco se va dirigiendo hacia el final de todos los árboles, si se observa la Ilustración 13, corresponde a los nodos inferiores. Como en este proyecto la etiqueta final que se aplica a cada aplicación es una variable categórica, el resultado de clasificación obtenido por este modelo será la etiqueta obtenida por la mayoría de los árboles que forman el bosque.

Hay que tener en cuenta que es conveniente que los árboles del bosque estén lo menos correlacionados posible para disminuir la tasa de error.

Este modelo presenta muchas **ventajas:**

- Es uno de los algoritmos de machine learning con mayor certeza para grandes cantidades de datos.
- Es capaz de correr eficientemente en grandes bases de datos.
- Puede manejar muchas variables sin la exclusión de ninguna.
- Es robusto en cuanto a datasets incompletos, puede predecir los datos que faltan.
- Cuenta con un método para detectar las interacciones de las variables.
- Evalúa los prototipos que informan sobre la relación entre las características y las etiquetas.

Al igual que todos los modelos, también cuenta con algunas **desventajas:**

- Suele sobreajustar los datos cuando tiene que realizar tareas con datasets ruidosos.
- La clasificación es de difícil interpretación para el hombre.

7.1.2 Técnicas de selección de características

Dado que no todas las características tienen igual importancia a la hora de la clasificación, se puede observar en la Ilustración 10, se han empleados los siguientes seleccionadores de características.

7.1.2.1 Variance Threshold

Este selector elimina las características con poca varianza, es decir, aquellas que no alcanzan cierto umbral. Por defecto, elimina todas aquellas cuya varianza sea cero, en otros términos, elimina las que tengan el mismo valor en todos los ejemplos. Este selector sólo se fija en las características, no en las etiquetas.

La librería sk-learn proporciona una función para ello. El umbral debe ser indicado por el programador, por defecto es 0.0, para que elimine todas las que tengan varianza cero. En nuestro caso, se ha dejado la de por defecto. En caso de querer cambiar este valor, se debería indicar como umbral la varianza máxima permitible.

7.1.2.2 Select K Best

Esta técnica de selección es para características univariadas, dicho de otro modo, trata cada característica por separado. Trabaja seleccionando las mejores características basadas en una función (test estadístico univariado). El valor K indica que se va a quedar con las k características cuyos resultados sean los más altos en la función, se eliminarán todas las demás.

En la función proporcionada por sk-learn, se debe introducir dicho valor K, que por defecto es 10, en nuestro caso se han seleccionado 30, 10 era demasiado poco. Además, hay que indicar la función que se desea aplicar a cada característica, por defecto es *ANOVA-F (f_classif)*, que se encarga de calcular el Análisis de la varianza con un factor. Esta se ha usado en el código de este proyecto.

7.1.2.3 Select Percentile

Este seleccionador funciona igual que el *Select K Best*, explicado en el apartado anterior. La única diferencia es que en lugar de indicarle el número de características con las que se va a quedar acorde a los valores más altos de la función aplicada, se le indica el porcentaje de las características con las que se va a quedar. Al igual que en el anterior, también hay que indicarle la función a desarrollar.

En nuestro caso, se ha seleccionado el porcentaje equivalente para que se quede con las 50 mejores, simplemente para cambiar con respecto al anterior modelo y poder comparar. Poner el porcentaje equivalente a 30 características, no mostraría ningún cambio. Respecto al test a aplicar, se ha seleccionado el mismo, *f_classif*.

7.1.2.4 Select Fpr, Fdr & Fwe

Esta técnica de selección es empleada para características univariadas. Dependiendo de si se trata de Fpr, Fdr y Fwe aplica diferentes testes.

Fpr: Aplica el test de tasa de falsos positivos, el cual controla la cantidad total de falsos positivos.

Fdr: Este realiza una estimación de la tasa de falsos descubrimientos.

Fwe: Calcula la probabilidad de realizar uno o más descubrimientos falsos o errores de tipo I cuando de desarrollan pruebas de hipótesis.

Una vez habido aplicado estas pruebas, se selecciona aquellas características cuyo resultado en ellas ha sido menor que cierto valor indicado.

Antes de aplicar este test, al igual que en las otras dos técnicas, hay que aplicar una función a las características, se ha elegido *f_classif* también. Tras los resultados obtenidos por dicha función es cuando se aplica el test. Se debe indicar un valor máximo del resultado obtenido por el test para que se quede con las características cuyo valor es menor.

7.1.2.5 Generic Univariate Select

La librería sk-learn proporciona una función general para la selección de características univariadas, en la cual se puede indicar el selector: *k_best, percentile, fpr, fdr o fwe* y la función a aplicar.

Dado que se han usado en todas las clasificaciones la ANOVA-F, se ha probado a emplear los mismos selectores, pero con la función de distribución *chi2* en lugar de *f_classif*. La distribución chi cuadrada o de Pearson es una distribución de probabilidad continua con un parámetro k que representa los grados de libertad de una variable aleatoria. Estos grados indican la cantidad de información proporcionada por algunos datos para estimar aquellos desconocidos.

7.1.3 Red neuronal

En este caso, el aprendizaje es profundo y se basa en la creación de una red neuronal. Se trata de una alternativa de costoso cómputo, ya que se entrena con todos los datos, no se hace selección de características, se emplean todas. Su funcionamiento se ha explicado en el apartado 7.3. Como todos los modelos empleados para machine learning, presenta sus pros y sus contras. A continuación, se explican los principales.

Ventajas de las redes neuronales:

- Crea su propia representación de la información, el usuario no tiene que hacer nada más que pasarle la información.
- Es tolerante a fallo, como almacena la información de manera redundante, esta puede seguir respondiendo, aunque esté afectada en algunas partes.
- Puede operar con señales con ruido u otros cambios en la entrada.
- Presenta gran facilidad de inserción en la tecnología existente.
- Implementada en buenas computadoras es capaz de proporcionar respuestas en tiempo real.

A su vez, presentan las siguientes **desventajas:**

- Presenta cierta complejidad de aprendizaje para grandes tareas.
- El tiempo de aprendizaje puede resultar elevado dependiendo de dos factores. Primero, si se incrementa la cantidad de patrones a clasificar y segundo si se necesita mayor capacidad de adaptación de la red para distinguir entre patrones parecidos.
- No permite interpretar lo que se ha aprendido, ella por si misma proporciona una salida, la cual debe ser interpretada por el programador y la aplicación para encontrarle sentido.

7.2 ELECCIÓN DE ALTERNATIVAS

Tras haber evaluado la calidad de cada sistema creado en base a las tres métricas, se han elegido los siguientes tres sistemas:

- **SISTEMA 1:**

  Creado empleando Shallow Learning.
  Modelo: Random Forest Classifier
  Seleccionador de características: SelectFwe con la función A*NOVA-F*.

  Este modelo suele presentar buenos resultados, ya que emplea la filosofía de árboles de decisión, pero en forma de bosque, es decir, con más de un árbol. Combinado con el seleccionador mencionado se obtiene una elevada exactitud.

- **SISTEMA 2:**

  Creado empleando Shallow Learning.
  Modelo: Multi-layer Perceptron Classifier
  Seleccionador de características: SelectFdr con la función *ANOVA-F*.

  La combinación de este modelo con dicho seleccionador de características presenta los mejores resultados en cuanto a la precisión.

- **SISTEMA 3:**

  Creado empleando Shallow Learning.
  Modelo: Multi-layer Perceptron Classifier
  Seleccionador de características: SelectFpr con la función *ANOVA-F*.

  Combinando el mismo modelo elegido para el sistema 2 y el seleccionador de características SelectFpr se obtiene el resultado más alto en cuanto a la sensibilidad.

7.2.1 Interpretación de los sistemas elegidos en base a las métricas

La razón de la creación de tres sistemas en este proyecto es el interés del usuario a la hora de saber si una aplicación es malware o no. De este modo, el usuario podrá elegir entre los tres sistemas, el que mejor se adapte a sus intereses.

- **Sistema 1: Mayor exactitud:** El sistema elegido como aquel que presenta la mayor exactitud es el que tiene un enfoque más general en la clasificación. Es decir, estaría indicado para usuarios que quieren disfrutar de una herramienta con la mayor tasa de aciertos en la predicción. Aquellos que quieren detectar malware, pero al mismo tiempo no desean que el sistema les haga pensar que una aplicación es maligna cuando en realidad no lo es. Buscan el equilibrio. No se hace distinción en que tenga más error al clasificar una app realmente benigna en una maligna o viceversa. Lo característico de este sistema es que tiene una gran tasa de acierto en clasificar una aplicación en malware o no.

- **Sistema 2: Mayor precisión:** El enfoque de este sistema es más específico. Es aquel que cuando predice que una aplicación es benigna, tiene la mayor tasa de acierto. Este sistema es el más seguro para el usuario ya que predecir que una aplicación es benigna cuando en realidad no lo es, pondría en peligro al dispositivo. En este sistema, este último caso tendría un porcentaje muy bajo de ocurrencia. Los usuarios que desean hacer uso de este sistema son aquellos a los que no les preocupa que una aplicación que sea benigna sea erróneamente clasificada. Esto último no pondría en peligro al dispositivo, simplemente podría causar en el usuario la desinstalación de aplicaciones que realmente no son malware porque se le ha hecho pensar al usuario que lo son.

- **Sistema 3: Mayor sensibilidad:** El fundamento de este último sistema se basa en lo siguiente. Dada una aplicación benigna, la clasificará correctamente con una alta tasa de acierto. Este sistema está dirigido para aquellos que quieren estar seguros de que cuando una aplicación es benigna realmente, el sistema acierte en el mayor de los casos.

8 PLANIFICACIÓN

La planificación es uno de los pasos previos más importantes en la realización de un proyecto, sin ella, el desarrollo de este causaría muchos problemas y sería guiado sin alcance, sin saber cuándo parar. En este apartado se va a explicar la programación y la estimación del orden de las actividades imprescindibles para alcanzar los objetivos del proyecto.

8.1 CICLO DE VIDA DEL PROYECTO

Para una correcta estructuración del proyecto, se ha decidido dividirlo en cuatro etapas. Cada una de ellas tiene unos objetivos. Estos deben ser claramente y correctamente alcanzados. A continuación, se van a explicar estas fases:

1. **Planteamiento inicial del proyecto**

   Esta primera fase consiste en la elección del objetivo final del proyecto con el director. A su vez, esto conlleva la identificación de los beneficios que va a tener el proyecto y de un modo su viabilidad, es decir, estudiar el éxito o fracaso de este. Por último, se acordarán los recursos, tecnologías y herramientas para su desarrollo.

2. **Desarrollo proyecto software**

   Esta segunda etapa abarca lo que es el desarrollo del código. Haciendo uso de los recursos, herramientas y tecnologías planteadas en la anterior fase, se procederá a la creación de las diferentes alternativas para el desarrollo de un sistema que alcance el objetivo principal. La mayoría de tiempo empleado para todo el proyecto se centra en esta fase, ya que es el core del mismo.

3. **Obtención de resultados para el Análisis de alternativas**

   La existencia de esta viene dada por la gran cantidad de datos tratados y los entrenamientos empleados. Hasta ahora, todo el proyecto se ha desarrollado con menos ejemplos de los existentes en el dataset. Esto es debido a que el procesamiento de los datos resultaba demasiado lento y la tarea de programar se complicaba y se ralentizaba. Por eso, en esta fase lo que se hará es correr los códigos desarrollados haciendo uso de todo el conjunto de datos en otro dispositivo capaz de ello.

   Una vez que se obtengan todos los resultados, se evaluarán y posteriormente se seleccionará la/s alternativa más óptima/s de acuerdo a dichos resultados.

4. **Documentación**
   La documentación, más que una fase, es una tarea complementaria a la realización del proyecto, es por eso que se ha decidido separar de las tres anteriores. En esta tarea se va a documentar todo lo realizado durante este. La manera en la que se ha realizado, la razón e importancia del mismo… en resumen aspectos tanto técnicos como económicos y sociales del proyecto.

8.2 RECURSOS NECESARIOS PARA SU DESARROLLO

Tal y como se ha explicado en el apartado anterior para el desarrollo del proyecto, es necesario planificar la elección de recursos, tecnología y herramientas, ya que, sin ellas, no podría llevarse a cabo el desarrollo de el mismo. Posteriormente, se explicarán los principales recursos empleados en las fases:

1. En esta primera fase, no es requerido ningún tipo de herramienta ni tecnología especial. Es importante enterarse sobre la realización de proyectos del área elegida. Para ello, con poder tener un dispositivo con conexión a Internet es suficiente para investigar un poco sobre el tema. Aparte, no está de más evaluar los problemas que pueden surgir a lo largo del proyecto, para enterarse de posibles fuentes, aparte del tutor, a las que recurrir en caso de problema.

2. Para el desarrollo software se pueden diferenciar dos partes. La primera abarca los requerimientos Hardware. Podrá emplearse cualquier ordenador que soporte y ejecute bien el programa Spyder, no importa el sistema operativo del mismo. Sería recomendable tener mínimo 8GB de RAM y procesador con 8 CPUs 1.80GHz. En cuanto a recursos software se necesitarán:

   - El entorno de desarrollo Spyder, gratuitamente descargable.
   - Librerías Sk-learn, Pandas, NumPy, MatplotLib y Keras

3. Tal y como se ha explicado en el apartado anterior, será necesario un clúster de servidores. Es decir, un conjunto de ordenadores que se unen mediante una red de gran velocidad de manera que el conjunto se ve como un único ordenador que resulta más potente que los ordenadores comunes.

4. Por último, se necesitará un ordenador que disponga de un programa informático orientado al procesamiento de textos. Pueden emplearse Microsoft Word, LibreOffice, Google Docs, AbiWord o muchos otros. Por otra parte, para la realización de el diagrama que se mostrará posteriormente, se puede hacer uso de programas informáticos como Microsoft Project o si no, existen herramientas para crear diagramas online como SmartSheet.

8.3 DIAGRAMA DE GANTT

Para planificar y programas las tareas a lo largo de la duración del proyecto, una herramienta muy útil es el diagrama de GANTT. Además, permite una fácil y cómoda visualización de las tareas, su duración y secuencia.

EDT	NOMBRE DE LA TAREA	DURACIÓN	COMIENZO	FIN
PT-1	**Planteamiento Inicial**	**24 días**	**Lun 26/11/18**	**Mie 13/02/19**
T-1.1	Elección del Tema	2 días	Lun 26/11/18	Mar 27/11/18
T-1.2	Lectura de información existente acerca del tema	11 días	Jue 24/01/19	Jue 07/02/19
T-1.3	Definición del alcance del proyecto	5 días	Jue 07/02/19	Mié 13/02/19
H-1	Inicio del proyecto	0 días	Mié 13/02/19	Mié 13/02/19
PT-2	**Desarrollo proyecto software**	**105 días**	**Jue 14/02/19**	**Mar 11/06/19**
T-2.1	Familiarización con el lenguaje de programación y entorno de desarrollo	14 días	Lun 18/02/19	Jue 07/03/19
T-2.2	Creación sistemas con aprendizaje tradicional	38 días	Vie 08/03/19	Mar 30/04/19
T-2.3	Creación sistema con aprendizaje profundo	26 días	Mar 07/05/19	Mar 11/06/19
PT-3	**Obtención de resultados para el Análisis de alternativas**	**8 días**	**Jue 27/06/19**	**Lun 08/07/19**
	Visualización de los resultados	6 días	Jue 27/06/19	Jue 04/07/19
	Elección sistema entrenado con aprendizaje tradicional	1 día	Lun 08/07/19	Lun 08/07/19
	Elección entre sistema entrenado con aprendizaje tradicional o profundo	1 día	Mar 09/07/19	Mar 09/07/19
PT-4	**Documentación**	**31 días**	**Mié 19/05/19**	**Mié 10/07/19**
	Decidir estructura memoria	3 días	Mié 29/05/19	Vie 31/05/19
	Redacción de la memoria		Lun 03/06/19	Mié 10/07/19
H-2	Fin del proyecto	0 días	Mié 10/07/19	Mié 10/07/19

Tabla 2: Actividades a realizar

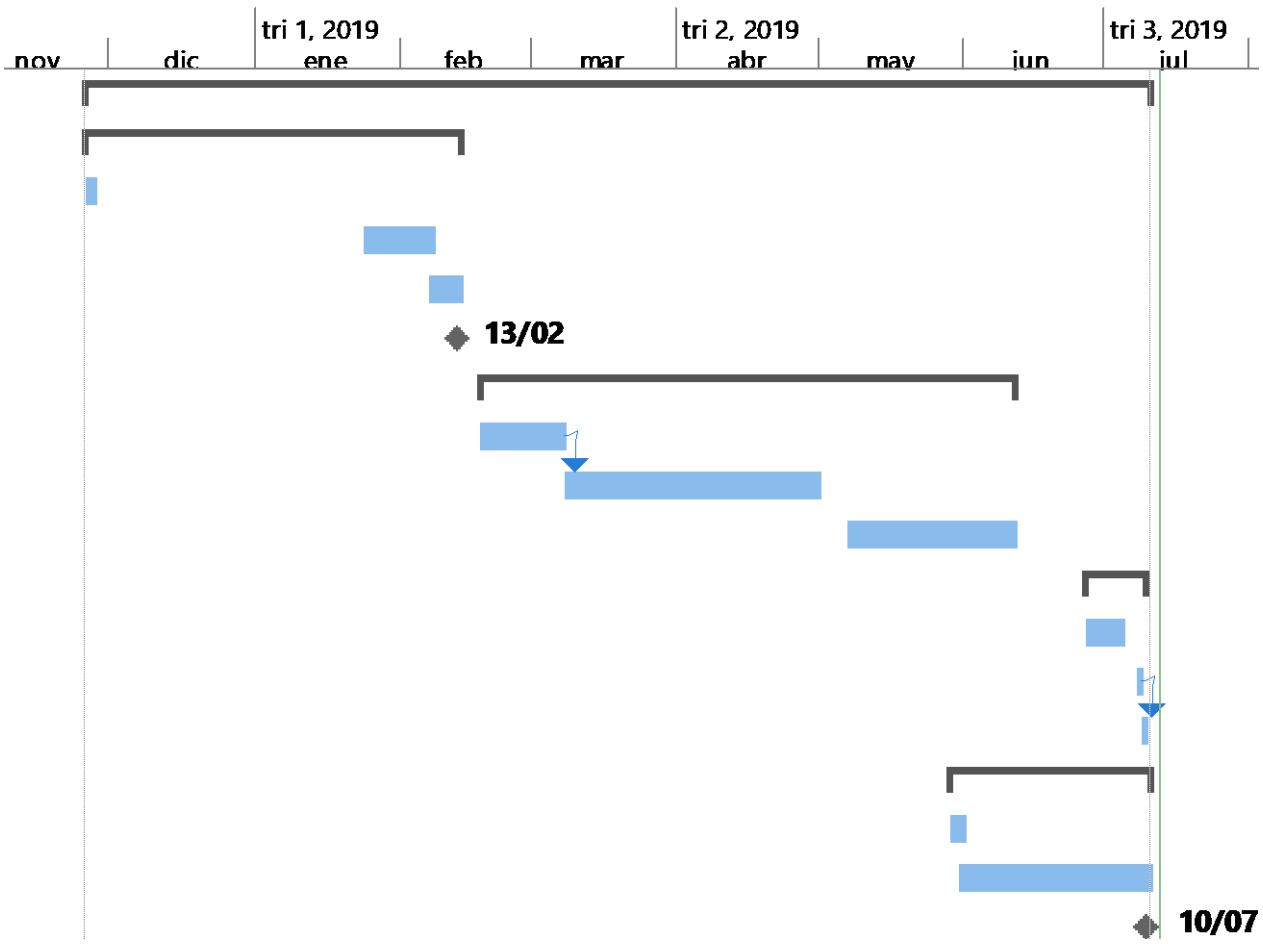

Ilustración 16: Diagrama Gantt del proyecto

9 PRESUPUESTO EJECUTADO

Este apartado engloba un resumen económico del proyecto. Se explicará el coste que ha implicado el desarrollo del proyecto. Como coste, según [17] se entiende lo siguiente: El pago que un servicio o producto requiere, puede ser tanto en efectivo o abonado en el futuro. Estos sólo afectan a los egresos y a los pagos. A pesar de referirse al presente, estos pueden subir o bajar en un futuro.

9.1 HORAS INTERNAS

En cuanto a personal necesario, para la realización de este proyecto ha sido necesario la colaboración de un ingeniero junior y un ingeniero senior(director).

Puesto	Número de horas (h)	Coste (€/h)	Coste total (€)
Ingeniero Junior	300	50	30.000
Director	50	60	3.000
SUBTOTAL			33.000

Tabla 3: Recursos humanos

9.2 AMORTIZACIONES

Respecto a los recursos empleados para el proyecto (explicados en la sección 8.2), hay algunos que son de uso libre, sin embargo, se ha tenido que pagar por otros de ellos.

Recurso	Coste Inicial (€)	Vida útil (h)	Tasa horaria (€/h)	Uso(h)	Coste (€)
Licencia Microsoft Office	80	1.500	0,067	105	5,6
Ordenador Portátil HP	730	9.800	0,075	280	20,85
Clúster de Servidores	3.000	43.970	0,068	168	11,46
SUBTOTAL					37,9

Tabla 4: Amortizaciones

9.3 GASTOS

En lo referente a todo el material empleado para el proyecto se presenta el siguiente resumen:

Puesto	Coste(€)
Libros	40
Material de oficina	20
Seguro Ordenador Portátil	60
SUBTOTAL	120

Tabla 5: Gastos

9.4 COSTE TOTAL

Concepto	Coste (€)
Horas Internas	33.000
Amortizaciones	37,9
Gastos	120
SUBTOTAL	33.157,19

Tabla 6: Coste Total

10 ANÁLISIS DE RIESGOS

Todo proyecto está expuesto a una serie de riesgos, los cuales deben ser analizados para poder anticipar soluciones. A continuación, se estudiarán las causas de posibles amenazas y eventos no deseados y daños y las consecuencias que estos pueden producir.

Para empezar, es necesario identificar los riesgos con más impacto en el proyecto. Se han podido identificar cuatro principales riesgos.

Tras su identificación se procederá a evaluar la probabilidad de ocurrencia y el impacto de cada riesgo para posteriormente representarlos en una tabla, de manera que esta información sea ilustrada de manera gráfica. Para su calificación se seguirán los siguientes parámetros:

1 Bajo
2 Medio-bajo
3 Medio-alto
4 Alto

Por último, se explicarán los planes de contingencia de cada riesgo. El objetivo de estos planes es disminuir la probabilidad de ocurrencia de los riesgos o en caso de que estos se den, establecer un plan para llevarlos a cabo.

10.1 R1: PRESUPUESTO DEMASIADO BAJO

Un error muy común es la realización de un presupuesto demasiado bajo. Este error se va manifestando a lo largo del proyecto. Poco a poco se va notando que los costes que ha tenido el proyecto hasta cierto momento han cubierto ya el presupuesto y todavía no se ha acabado el proyecto.

Probabilidad de ocurrencia: 1

Impacto: 4

Plan de Contingencia: Una fácil solución es siempre estimar de más a la hora de plantear el presupuesto. Sin embargo, suele resultar difícil establecer un presupuesto, por lo que la mejor opción sería solucionarlo aumentando el dinero dedicado a los imprevistos.

10.2 R2: ERRORES EN EL DESARROLLO SOFTWARE

La creación errónea de sistemas clasificadores provocaría serios problemas. El tener un sistema que no clasifique bien, afecta a más de una fase del proyecto. A parte de obtener resultados erróneos e incoherentes, también tiene impacto sobre las medidas de métricas del mismo sistema. Esto último a su vez, conlleva a la incorrecta elección del mejor sistema clasificador, ya que se estarían empleando falsos resultados y evidencias para dicha elección.

Probabilidad de ocurrencia: 4

Impacto: 2

Plan de Contingencia: Una posible solución sería intentar realizar las labores de programación cuanto antes y dejando un amplio margen antes de la fecha límite del proyecto. De este modo, se dispondría de suficiente tiempo para corregir los errores. Sin embargo, la mejor opción sería la realización de pruebas unitarias por fases en el proyecto. Así, sería más fácil identificar el error.

10.3 R3: SUPERACIÓN FECHA LÍMITE

Cuando se decide llevar a cabo un proyecto, es muy importante tener en cuenta el factor tiempo. Si acercándose la fecha límite todavía queda bastante del proyecto por hacer, se pondría en riesgo la planificación y la organización. Esto conlleva a intentar realizar todo lo más rápido posible y que no salga como se desee o que se alargue y haya que acordar nuevas condiciones con el cliente.

Probabilidad de Ocurrencia: 3

Impacto: 1

Plan de Contingencia: La mejor solución sería establecer siempre de más el tiempo previsto y proponer una fecha de fin de las tareas antes de que acabe el proyecto. De este modo, siempre quedarían unos días de margen, los cuales en caso de tener todo acabado se podrían emplear para posibles mejoras.

10.4 R4: PÉRDIDA DE DATOS O AVERÍA DE EQUIPOS

La pérdida de los códigos creados implicaría tener que empezar el proyecto de cero. Esto probablemente conllevaría al fracaso del mismo por la falta de tiempo.

Probabilidad de ocurrencia: 2

Impacto: 3

Plan de Contingencia: Para evitar este riesgo, se deben realizar copias de seguridad de toda la información del proyecto, así como guardarlo en diferentes dispositivos y en la nube.

Impacto \ Probabilidad	Bajo	Medio – Bajo	Medio – Alto	Alto
Bajo			R3	
Medio – Bajo				R2
Medio – Alto		R4		
Alto	R1			

Ilustración 17: Matriz probabilidad-Impacto

11 CONCLUSIONES

Tras haber creado diversos sistemas clasificadores y realizar un estudio sobre ellos en base a la calidad de los mismos, se ha podido concluir lo siguiente.

En primer lugar, tras empezar a entrenar los modelos con todas las características disponibles, se ha podido observar que no todas las características tienen la misma importancia para el entrenamiento del clasificador. Es por ello, que el uso de un seleccionador de características ha resultado muy beneficioso y útil en la creación de los sistemas. Evitando entrenar a dichos sistemas con características no importantes ha permitido un entrenamiento y una predicción más rápida de los mismos, obteniendo mejores resultados en cuanto a las métricas deseadas.

En segundo lugar, dado que el Deep Learning simula un cerebro humano mediante las redes neuronales, se esperaba que los resultados fueran mejores que con Shallow Learning. Sin embargo, con la información, los modelos y seleccionadores de características empleados en el aprendizaje no profundo, ha resultado mejor el uso de este tipo de aprendizaje en el proyecto en cuanto a las tres métricas principales.

Por otra parte, la calidad de los sistemas creados ha resultado ser bastante elevada, lo cual implica grandes beneficios para los usuarios, ya que estos estarán dispuestos a usar sistemas como los creados en este proyecto para aumentar su seguridad en sus dispositivos Android. Del mismo modo que se ha creado para este último SO, se podría haber creado para aplicaciones soportadas en otros SO como IOS, Windows Phone, BlackBerry 6…

Desde un enfoque más general, en este TFG se ha hecho uso de subcampos de la IA, Big Data y un lenguaje de programación que presentan una gran tendencia en el mundo del software. Por ello, los conocimientos aprendidos durante su desarrollo han sido realmente útiles, muy apreciables y valorables en el ámbito software. Asimismo, las áreas a las que se pueden aplicar dichas tecnologías son muy diversas y contribuyen positivamente al avance tecnológico, permitiendo así, a los usuarios disfrutar cada vez de mejores servicios.

Por último, para que quede constancia de los resultados de todos los sistemas creados, estos han sido almacenados en ficheros. De todos los sistemas, únicamente se han elegido tres de ellos. A continuación, se muestra el porcentaje de acierto de cada sistema, cada uno caracterizado por ser el mejor en cada una de las tres métricas.

Modelo Clasificador	**Seleccionador de características**	**Función empleada en el seleccionador**	**Métrica**	**Resultado métrica**
Random Forest Classifier	Select Fwe	ANOVA-F	Exactitud	89.1273 %
MLP Classifier	Select Fdr	ANOVA-F	Precisión	93.943 %
MLP Classifier	Select Fpr	Chi cuadrado	Sensibilidad	97.974 %

Tabla 7: Resultados de los sistemas elegidos

12 REFERENCIAS

[1] AI +DA, *OmniDroid: A comprehensive benchmark dataset of dynamic and static features from Android applications*, Diciembre 2018.
https://aida.ii.uam.es/datasets/

[2] AppBrain, *Number of Android apps on Google Play*, Julio 2019
https://www.appbrain.com/stats/number-of-android-apps

[3] David Justo, *Si tienes una de estas populares apps en tu teléfono móvil, bórrala: contienen virus*, Mayo 2018.
https://cadenaser.com/ser/2018/05/11/ciencia/1526016366_924982.html

[4] Chema Alonso, *¿Cuántas apps nuevas hay en Google Play en un mes? ¿Cuántas desaparecen diariamente?*, Marzo 2015.
http://www.elladodelmal.com/2015/03/cuantas-apps-nuevas-hay-en-google-play.html

[5] Adext, *Guía explicación cómo funciona el AMaaS de Adext*, Febrero 2019
https://cdn2.hubspot.net/hubfs/2839604/Adext%20Agency%20Partner/Guia-Explicacio%CC%81n-Como-funciona-el-AMaaS-de-Adext.pdf

[6] Insituto de Ingeniería del conocimiento, Universidad Autonoma de Madrid, *Las 7 Vs del Big data: Caracterísitcas más importantes,* 2018.
http://www.iic.uam.es/innovacion/big-data-caracteristicas-mas-importantes-7-v/

[7] Ángel M. Rayo, *Tipos de datos en Big Data: clasificación por categoría y por origen*, Mayo 2016.
https://www.bit.es/knowledge-center/tipos-de-datos-en-big-data/

[8] Raona Enginyers S.L., *Machine learning: Tipos de Machine Learning*, 2017.
https://www.raona.com/machine-learning-tipos-machine-learning/

[9] Jochem Grietens, *What's the difference between Machine Learning & Deep (Machine) Learning?*, Abril 2018.
https://verhaert.com/difference-machine-learning-deep-learning/

[10] Digital Tech Institute, *Los 7 mejores lenguajes de programación para IA*, Abril 2018.
https://www.digitaltechinstitute.com/8-mejores-lenguajes-de-programacion-para-ia/

[11] The Security Sentinel, *Troyanizando Dispositivos Android – Inyección manual*, Octubre 2017.
https://thesecuritysentinel.es/troyanizando-dispositivos-android-inyeccion-manual/

[12] Chris Hoffman, *Cracked Android Apps and Games: Read This Before Downloading*, Julio 2013.
https://www.makeuseof.com/tag/cracked-android-apps-and-games-read-this-before-downloading/

[13] Malwarebytes LABS, *Analyzing malware by API calls*, Octubre 2017.
https://blog.malwarebytes.com/threat-analysis/2017/10/analyzing-malware-by-api-calls/

[14] Asier Martínez Retenaga, *Android Malware Situation*, Febrero 2015.
https://www.incibe-cert.es/sites/default/files/contenidos/estudios/doc/android_malware_situation.pdf

[15] Juan Ignacio Bagnato, *Árbol de Decisión en Python: Clasificación y predicción*, Abril 2018.
http://www.aprendemachinelearning.com/arbol-de-decision-en-python-clasificacion-y-prediccion/

[16] Scikit-Learn, *Ensemble methods*, 2007-2019
https://scikit-learn.org/stable/modules/ensemble.html#classification

[17] wvggfinanzas, *Diferencias Costos y presupuestos*
https://sites.google.com/site/wvggfinanzas/home/diferencias-costos-y-presupuestos