Secure Learning para detección de *Android Malware*

Jhoan Steven Delgado Villarreal

Dirigido por:

Christian Urcuqui, Msc.

Javier Díaz, Ph.D.

Andrés Navarro, Ph.D.

Facultad de Ingeniería

Programa de Ingeniería de sistemas

Departemento TICS

Universidad Icesi

Proyecto de grado

Santiago de Cali, Junio 2019

En memoria de mi querido abuelo Miguel A. Villarreal.
(1938 – 2017)
El hombre más sabio que alguna vez haya conocido...

A mi querida madre Gloria A. Villarreal, por su tiempo, tenacidad, paciencia, entrega total, y apoyo incondicional.

A mi querida abuela, Rosenda D. por sus buenas enseñanzas. Mi tío Fernando Villarreal y mi tía Fanny Villarreal por sus sabios consejos y apoyo. Mi tía Ruby y mi tía Martha por su buena disposición y apoyo. Mi padre, Benjamín Delgado por el mindset y apoyo. Mi tío Gerardo Delgado por su apoyo.

A Dios, por las maravillosas oportunidades que me ha brindado ...

Agradecimientos especiales

Era una tarde de noviembre del año 2016, cuando terminaba de sustentar el examen final del curso de electromagnetismo, estaba muy emocionado al ver finalmente la demostración de que las ondas electromagnéticas no necesitaban un medio para propagarse. Justo al momento de despedirme del profesor Hernán Triana, él se levanta de su silla y me hace una cordial invitación a unirme al semillero de investigación del grupo de informática y telecomunicaciones (i2t), la cual decidí aceptar. Hoy, 2 años y medio después, me encuentro infinitamente agradecido con el profe Triana, pues el grupo de investigación i2t es uno de los mejores eventos y recuerdos que tendré de toda mi carrera universitaria.

Conocí y pude colaborar con muchas personas increíbles entre las cuales se encuentra mi tutor, el profesor Christian Urcuqui, quién es muy apasionado por el mundo de la investigación, y además es una gran persona. El profesor Urcuqui fue mi guía a través del maravilloso mundo de la investigación, aprendí muchísimo de él. Este proyecto de grado es una continuación de un trabajo previo [1], el cual pude colaborar con él y presentarlo al congreso COLCOM 2018 en la ciudad de Medellín. Infinitas gracias a el profesor Christian Urcuqui por todo.

También quisiera agradecerle al profesor Helmer Peña en la ciudad de Palmira, quien fue mi guía en los primeros semestres de mi carrera cuando no tenía ni idea de como realizar una factorización. A mis otros dos tutores el profesor Javier Díaz quien fue mi profesor durante el curso de analítica de datos, un curso en donde pude ver las bondades de la ciencia de datos en su máxima expresión. Al profesor Andrés Navarro, por haberme dado la bienvenida en el grupo de investigación y permitirme colaborar en dos proyectos importantes.

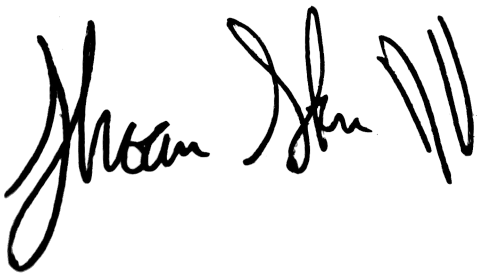
Finalmente quisiera agradecerle al grupo i2t, especialmente al profesor Domiciano Rincón, quien fue mi profesor de Apps Móviles, al profesor Marlon, por colaborarme en la resolución de algunas dudas. Al departamento de matemática y estadística, especialmente a los profesores Aníbal Sosa, Hendel Yaker, Alfonso Bustamante, y Ernesto Peláez, además de dar clases muy emocionantes, sus enseñanzas me han acompañado a lo largo del desarrollo de este proyecto.

Al departamento de TICS, especialmente al profesor Luis Eduardo Múnera, quien fue mi profesor de Inteligencia Artificial, al profesor Andrés Aristizábal quien fue mi profesor de Matemáticas discretas e Informática teórica, al profesor Juan Manuel Madrid, quien fue mi profesor de Seguridad, también sus enseñanzas me acompañaron a lo largo del desarrollo del proyecto. A la profesora Norha Villegas, y al profesor Gabriel Tamura, por enseñarme que la curiosidad intelectual es muy importante.

¡Y por supuesto, un agradecimiento al querido lector por tomarse el tiempo de leer este trabajo!

¡Qué viva la física teórica! ¡Qué vivan las matemáticas! ¡Qué viva la ciencia de datos! ¡Qué viva la ciberseguridad! ¡Qué viva el *software*! Mejor dicho, ¡Qué viva la ciencia y la investigación!

Hecho en LaTeX



Jhoan Steven Delgado Villarreal

Abstract

Since the beginning of the year 2009 until today Android has been growing its market share almost exponentially, becoming the world's most used mobile operating system. Due to its technological impact, its open source code and the possibility of installing applications from third parties without any central control, Android has also become a Malware target.

In a previous work [1], *Machine Learning* models were trained and evaluated successfully reaching detection rates of 99 % and 81 % respectively. The following project is a continuation of this previous work. We develop a *Reinforcement Learning* strategy to exploit a *Machine Learning* defense model for classifying Android Apps, and then we improve its robustness through *Adversarial training*. We also suggest an activity concerning the security of *Machine Learning* models against adversarial attacks in the evaluation phase of the CRISP-DM methodology.

Resumen

Desde principios del año 2009 hasta hoy Android ha ido creciendo de forma casi exponencial, convirtiéndose en el sistema operativo móvil más utilizado del mundo, y debido a su impacto tecnológico, a su código abierto y a la posibilidad de instalar aplicaciones de terceros sin ningún tipo de control central, Android se ha convertido también en un objetivo de malware.

En un trabajo anterior [1], los modelos de *Machine Learning* fueron entrenados y evaluados exitosamente alcanzando tasas de detección del 99 % y 81 % respectivamente. El siguiente proyecto es una continuación de este trabajo anterior. Desarrollamos una estrategia de *Reinforcement Learning* para realizar un *exploit* a un modelo de defensa de ML para la clasificación de aplicaciones Android, y luego mejoramos su robustez a través de *Adversarial Training*. Sugerimos también una actividad relativa a la seguridad de los modelos de *Machine Learning* contra ataques adversarios en la fase de evaluación de la metodología CRISP-DM.

Índice general

Listado de acrónimos

- **ML:** *Machine Learning.*
- **AML:** *Adversarial Machine Learning.*
- **MD:** Modelo de defensa.
- **AT:** *Adversarial Training.*
- **SL:** *Secure Learning.*
- **RL:** *Reinforcement Learning.*
- **PDM:** Procesos de decisión de Markov.
- **APK:** *Android Application Package.*
- **APP:** Applicación.
- **TICS:** Tecnologías de la información y comunicaciones.
- **FP:** Falsos positivos.
- **FN:** Falsos negativos.

Glosario de términos

- ***Dataset:*** Conjunto de datos históricos relativos a un problema o acontecimiento.
- ***Over-fitting:*** Término empleado para referirse cuando un modelo de *Machine Learning* se entrena excesivamente sobre los datos y no le es posible generalizar el conocimiento.
- ***Under-fitting:*** Término empleado para referirse cuando un modelo de *Machine Learning* no se entrena lo suficiente sobre los datos, lo que implica que esté apartado de la realidad (Es decir, tenga un sesgo).
- ***Malware:*** Programa malicioso diseñado con el propósito de perjudicar a un usuario, ya sea robando información sensible o dañando el dispositivo en el cual se ejecuta.
- ***Secure Learning:*** Término empleado para referirse cuando un modelo de *Machine Learning* tiene un buen desempño (defensa) ante ataques adversarios que tienen como propósito engañar al modelo.
- ***Exploit:*** Término utilizado para referirse al aprovechamiento de una vulnerabilidad con un propósito malicioso (Es decir, un ataque) de un sistema informático.
- ***Accuracy:*** Tasa de los registros correctamente clasificados por el modelo de *Machine Learning*.
- ***White-box:*** Término utilizado para referirse al ataque de un modelo de *Machine Learning* donde se tiene el conocimiento previo de sus parámetros, su *dataset*, y su algoritmo de clasificación.
- ***Black-box:*** Término utilizado para referirse al ataque de un modelo de *Machine Learning* donde se desconoce previamente sus parámetros, su *dataset*, y su algoritmo de clasificación.

Índice de figuras

Índice de tablas

Motivación y antecedentes

1.1. *Contexto*

Hoy en día, Android es el sistema operativo móvil más usado del mundo, además es de código abierto (Open Source), cuenta con una cuota de mercado para teléfonos inteligentes aproximadamente del 75-80 % . Android, permite instalar aplicaciones APK (Android Application Package), tanto de la tienda oficial (Play Store) como de otras fuentes de terceros (no oficiales). Debido a que Android tiene una cuota de mercado tan alta, el sistema operativo es muy apetecido por los Cibercriminales, puesto que puede llegar a la mayor cantidad de las personas que usan un terminal móvil [1].

Según [10] los Cibercriminales crean Malware, en su mayoría con el fin de obtener ganancias de una forma ilegal o en algunos pocos casos para realizar activismo político, como por ejemplo el grupo *Anonymous*.

1.2. *Antecedentes del problema*

[1] Discute los distintos métodos para analizar el Malware en Android. Enfocándose en *Machine Learning*, obtiene algunas características claves, tanto en la capa de red, como en la capa de aplicación que son fundamentales en determinar si cierta APK es Malware o no, luego entrena seis modelos de *Machine Learning* tienendo en cuenta estas características, y concluye que al evaluarlas sí es posible hacer la detección de Android Malware utilizando técnicas de *Machine Learning*. Como trabajo a futuro [1] argumenta que a pesar de que los clasificadores de ML son muy buenos en hacer la clasificación, existe el riesgo de que estos puedan ser fácilmente engañados, haciendo pensar que una APK *Malware* es *Goodware*. Este suceso pertenece a un campo de investigación llamado *Adversarial Machine Learning*. [11] Discute que ML es una herramienta muy poderosa que ha sido utilizada en muchas aplicaciones,

incluyendo servicios web, agentes de servicio online, monitoreo de *Cluster*, y entre otras aplicaciones donde se evidencian patrones dinámicos de datos cambiantes. Posteriormente, otro resultado de estos investigadores[12] proponen un marco de trabajo para la seguridad del aprendizaje en ML, 8 tipos de ataques y mecanismos de defensa. Sin embargo [13] discute que se ha hecho un gran descubrimiento que varios modelos de ML incluyendo las redes neuronales del estado del arte son vulnerables a muestras adversarias, es decir, que los modelos de ML realizan mal la clasificación de las muestras obtenidas de la distribución. Incluso, en la mayoría de los casos, diferentes modelos entrenados con subconjuntos del dataset, clasifican mal la misma muestra adversaria. Esto nos lleva a que las muestras adversarias exponen puntos ciegos fundamentales de los algoritmos de entrenamiento (ML).

1.3. *Justificación*

Es importante solucionar este problema, ya que así la tasa de error de clasificación de los algoritmos de ML se reduciría bastante, además que la clasificación errónea según lo plantea [12], hace que los usuarios desconfíen del modelo o aplicación y dejen de usarlo (este sería el mejor caso), o en el peor de los casos, que no se den cuenta y sigan usando el modelo, que traería catástrofes, debido a que se estaría clasificando algo maligno como si fuese benigno.

Descripción del problema

2.1. *Formulación del problema*

En la actualidad existen vulnerabilidades en los algoritmos de ML para la clasificación de Android Malware [14], esto se debe a que algunos modelos de ML son muy lineales, (es decir, no muy flexibles), también es debido a que se tiene por premisa que el *dataset* de entrenamiento y el *dataset* de evaluación pertenecen a la misma distribución estadística y los atacantes (adversarios, en este caso Apps malignas modificadas) pueden engañarlos, y parecer benignos [15]. Otra razón por las que estas vulnerabilidades ocurren es cuando los modelos de ML se vuelven muy complejos (overfitting). Esto ha causado no solo que se clasifiquen mal las Apps, sino también que se pierda la confianza de los usuarios con respecto al modelo de ML, perjudicando sus datos y sus terminales móviles.

Objetivos

3.1. *Objetivo General*

Implementar *Secure Learning* a un modelo clasificatorio para detección de *Android Malware*

3.2. *Objetivos Específicos*

- Entrenar y evaluar seis modelos de ML clasificatorios para la detección de *Android Malware* a partir del tráfico de red.
- Implementar un método de programación para realizar un *exploit* al mejor modelo de ML previamente entrenado.
- Proponer una conjunto de recomendaciones de *Secure Learning* para el mejor modelo de ML previamente entrenado.

Marco teórico

El marco teórico del proyecto se dividirá en tres categorías. La primera, redes y comunicaciones, en donde se realizará una brevemente el concepto de redes, el modelo OSI, y los diferentes protocolos utilizados a lo largo del proyecto. La segunda, *Machine Learning* (ML), en donde se abordan los conceptos de Aprendizaje supervisado, Aprendizaje no supervisado y *Reinforcement Learning* (se hace énfasis en aprendizaje supervisado y *Reinforcement Learning* debido a los objetivos del proyecto). La tercera, Ciberseguridad, en donde se abordan los conceptos de *Malware*, los distintos tipos de análisis para detectarlos, el concepto de *exploit*, *Secure Learning* y finalmente el concepto de *Adversarial Machine Learning*.

4.1. *Redes y comunicaciones*

Una red es un "Sistema de comunicación que permite el intercambio de información entre un conjunto de dispositivos autónomos geográficamente dispersos, sobre una base de tiempos total o parcial". [16]. Las redes surgen debido a esta necesidad de compartir datos en cierto momento incluso cuando los dispositivos estén geográficamente dispersos.

Para realizar dicho proceso, se debe tener un modelo referencia, que permite "definir el alcance de la arquitectura de un sistema" [16]. Es decir, un modelo conceptual que ayuda a entender la estructura de un sistema que cumple un fin, en este caso el de transferir los datos.

Para el caso de las redes y comunicaciones, se tiene como modelo de referencia el modelo OSI (*Open Systems Interconnection model*), y consta de las siguientes siete capas, que se ilustran a continuación:

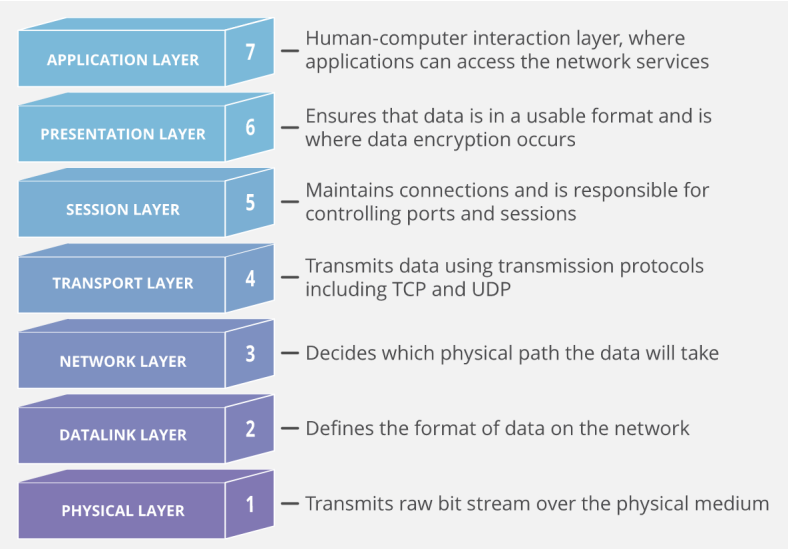


Figura 4.1: El modelo OSI [2].

Cuando uno o varios equipos se conectan a una red, las reglas y procedimientos técnicos que gobiernan su interacción y comunicación, se llaman protocolos. [16]. Existen variedades de protocolos en la actualidad, y cada uno tiene sus ventajas y desventajas. Entre los protocolos más usados y que conciernen el presente proyecto son:

- **Protocolo TCP:** Es un protocolo orientado a la conexión, es decir, crea una conexión virtual entre dos procesos para enviar datos, además es muy confiable ya que se asegura que los paquetes lleguen de forma correcta a su receptor, pues cuenta con un control de flujo y error a nivel de transporte [17]. Un ejemplo de este puede ser la descarga de un archivo desde el navegador.

- **Protocolo UDP:** Es un protocolo que no es orientado a la conexión, es decir, a diferencia de TCP no se asegura de que el receptor reciba de forma los paquetes, sino que simplemente los envía "a la dreiva", pues no existe un control flujo y error a nivel de transporte, lo que implica que no es fiable. Un punto positivo de este protocolo, es que es mucho más rápido que el protocolo TCP [17]. Un ejemplo de este puede ser la radio en internet que se transfiere vía *UDP streaming*.

También es importante conocer un poco el sistema jerárquico DNS (*Domain Name System*), que fue inventado en el año 1984, y permite a los usuarios de red utilizar nombres jerárquicos sencillos para comunicarse con otros equipos, en lugar de

memorizar sus direcciones IP [16]. El DNS puede traducir un nombre a una dirección IP, y una dirección IP a un nombre.

4.2. *Machine Learning*

Actualmente, *Machine Learning* ha logrado conquistar a la industria, pues juega un papel muy importante en la esencia de los productos tecnológicos que conocemos hoy en día, como por ejemplo: El sistema de recomendación de Netflix, o el sistema de reconocimiento de voz de los *smartphones*. *Machine Learning* es una ciencia (también un arte), la cual se encarga de programar los computadores de cierta forma que estos puedan aprender directamente de un conjunto de datos. En el año 1959, Arthur Samuel, fue quien acuñó el término *Machine Learning* definiéndolo como: "El campo de estudio que le brinda a los computadores la habilidad de aprender sin necesidad de estar explícitamente programados". Sin embargo, en el año 1997, Tom Mitchell, da una definición más formal de lo que es *Machine Learning*: "Se dice que un programa de computador aprende de una experiencia E con respecto a alguna tarea T y alguna medida de rendimiento P, si su rendimiento en T, medido por P, mejora con la experiencia E" [3].

El corazón de esta ciencia, según las definiciones anteriores, es aprender de la experiencia (Muy similar a la forma en que aprenden los seres humanos). Esta experiencia, está expresada en forma de datos históricos almacenados en un computador, para luego extraer de ellos patrones que permitan generalizar el conocimiento (ver Figura 4.2). Dentro del mundo de reconocimiento de patrones, hay dos grandes tipos: El aprendizaje supervisado o predictivo, y el no supervisado[18]. Además, existe una rama dentro de *Machine Learning* llamada *Reinforcement Learning* que tiene una diferencia muy significativa con respecto a el aprendizaje supervisado y no supervisado.

4.2.1. *Aprendizaje supervisado*

El aprendizaje supervisado parte de un conjunto de objetos (datos históricos) con cardinalidad N, que están descritos por un vector D-dimensional $\vec{x} = \{\alpha_1, \alpha_2, \ldots, \alpha_D\}$ de características (variables predictivas) y una variable de salida y que representa un elemento del conjunto de clases C (que puede ser finito), donde sus elementos son la solución esperada de un problema específico. Este conjunto C puede tener valores

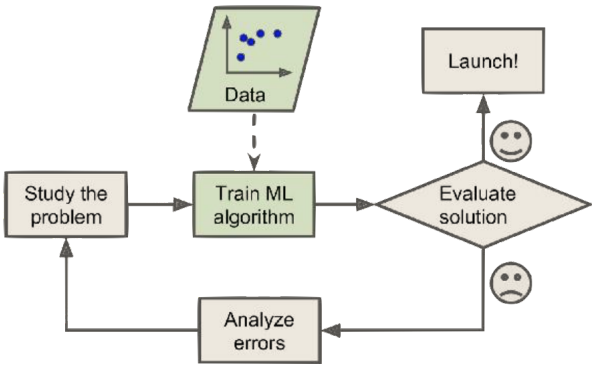


Figura 4.2: El proceso de *Machine Learning* [3]

que son categóricos o numéricos (números reales). A el conjunto de objetos (datos históricos) descritos por $\vec{x}$ junto con la variable y se expresan matemáticamente como:

$$\mathcal{D} = \{ (\vec{x_1}, y_1), (\vec{x_2}, y_2), \ldots, (\vec{x_N}, y_N) \}$$

A partir del conjunto $\mathcal{D}$, el aprendizaje supervisado construye un modelo o regla general cuyo propósito será clasificar entradas $\vec{x_i}$ nuevas de las cuales no se conoce su clase y_i, es decir la solución al problema [18, 19]

Algunos de los algoritmos de ML más importantes para aprendizaje supervisado según [3] son:

- *k-Nearest Neighbors*
- *Linear Regression*
- *Logistic Regression*
- *Support Vector Machines (SVMs)*
- *Decision Trees* y *Random Forests*
- *Neural Networks*

4.2.1.1. Clasificación

Cuando la variable y es categórica, se trata de un problema de clasificación o reconocimiento de patrones, entonces $y_i \in \{ c_1, c_2, \ldots, c_M \}$ $\forall_i = 1, \ldots, N$ para un problema con M clases distintas, donde estas clases son la solución esperada al problema. Si $|C| = 2$, se conoce como un problema de clasificación binaria (donde la mayoría de las veces se asume que $y_i \in \{ 0, 1 \}$ $\forall_i = 1, \ldots, N$.), en cambio, si $|C| > 2$, se le conoce como un problema de clasificación multiclase.

4.2.1.2. Regresión

Cuando la variable y es numérica, es muy similar al caso de clasificación, con la diferencia en que en el caso de la regresión en vez de tener un conjunto de clases C, se considera todo el conjunto de los números reales, es decir $y_i \in \mathbb{R}$ [20].

4.2.1.3. El problema de aprendizaje

Para comprender a un nivel introductorio la construcción del modelo, se procede a formalizar el problema de aprendizaje (*The learning problem*) para el caso de los problemas de clasificación, junto con un ejemplo sobre la aplicación del aprendizaje supervisado para realizar la detección Apps maliciosas en Android [1].

El problema de aprendizaje consiste en los siguientes elementos:

- **Entrada:** $\mathcal{X}$ (Conjunto de todos los vectores $\vec{x}$, en este caso características de tráfico de red de Android Apps)
- **Salida:** $\mathcal{Y}$ (Conjunto de todos los y, es decir la solución esperada al problema. En este caso *Benign, malicious*)
- **Función objetivo:** $f \colon \mathcal{X} \to \mathcal{Y}$ (Fórmula ideal para clasificar Android Apps)
- **Datos:** $\mathcal{D} = \{ (\vec{x_1}, y_1), (\vec{x_2}, y_2), \ldots, (\vec{x_N}, y_N) \}$ (Datos históricos de Android Apps)
- **Función hipótesis:** $g \colon \mathcal{X} \to \mathcal{Y}$ (Muy similar a f)
- **Algoritmo de aprendizaje:** $\mathcal{A}$
- **Conjunto de hipótesis:** $\mathcal{H}$ (Fórmulas candidatas para g)

La función objetivo (que es desconocida) $f \colon \mathcal{X} \to \mathcal{Y}$ tiene como dominio entrada el conjunto $\mathcal{X}$ y como codominio el conjunto de salida $\mathcal{Y}$ (que puede tomar un valor de *benign* para Apps benignas o de *malicious* para Apps maliciosas). Para todos los problemas de ML, la función f es desconocida, debido a que si se conociera, simplemente se realizaría su implementación y ya no tendría sentido el aprendizaje.

Lo que sí es posible conocer, es la función hipótesis $g \colon \mathcal{X} \to \mathcal{Y}$, $y_i = g(x_i)\ \forall_i = 1, ..., N$. Esta función se escoge a través de un algoritmo de aprendizaje $\mathcal{A}$ que haciendo uso del conjunto de entrenamiento $\mathcal{D}$, se basa para escoger a g de un conjunto de fórmulas candidatas $\mathcal{H}$ (El conjunto $\mathcal{H}$ podría ser por ejemplo, el conjunto de todas las fórmulas

lineales. El algoritmo de aprendizaje $\mathcal{A}$ escogerá la que mejor se ajuste a el conjunto de entrenamiento $\mathcal{D}$).

Finalmente, como el conjunto de entrenamiento $\mathcal{D}$ es un reflejo de la función ideal f (debido a que estos datos históricos son una muestra de la población total de Android Apps), esto implica que entre más significativa sea la muestra en $\mathcal{D}$, entonces $g \approx f$. Esta aproximación es el fin del aprendizaje [4].

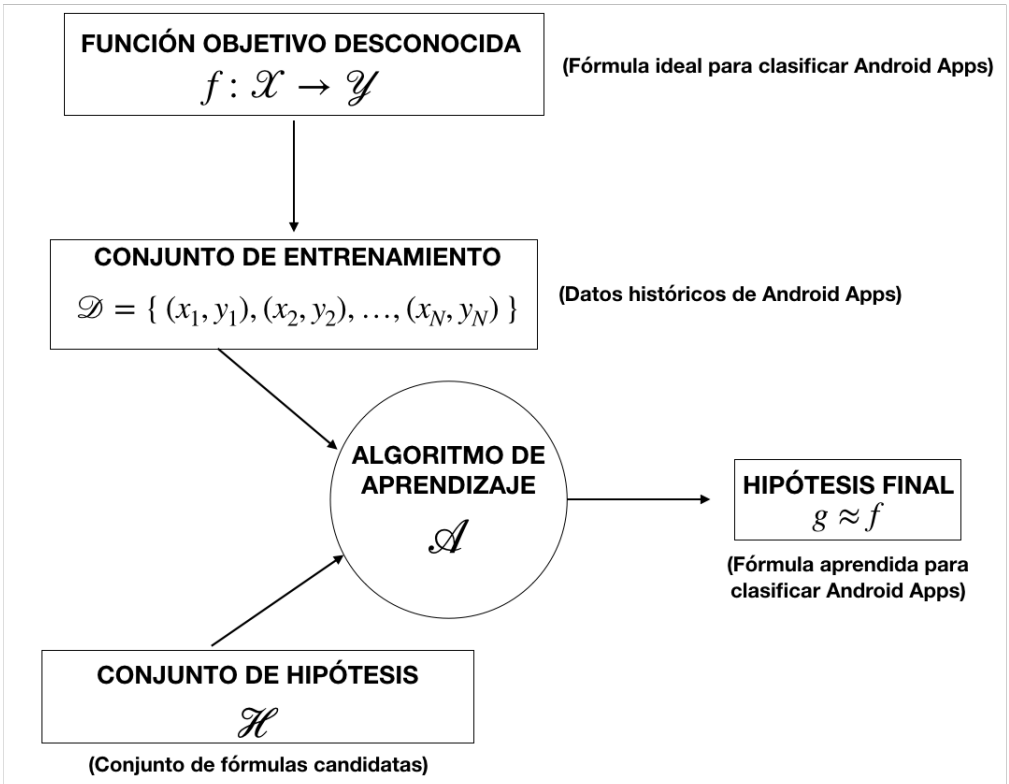


Figura 4.3: Esquema básico del problema de aprendizaje [4]

4.2.1.4. Protocolos de evaluación

Existen ciertos protocolos para realizar la validación de la capacidad de generalización de los modelos, con respecto a un conjunto de datos nuevos, para evitar el sesgo causado al realizar la evaluación con el mismo conjunto de datos históricos $\mathcal{D}$. Entre los que más se destacan son [9]:

- *Holdout*
- *K-fold cross-validation*

Hold-out, realiza una partición del conjunto de datos en dos partes. La primera, se llama "conjunto de entrenamiento" (Con el cual aprende el algoritmo de clasificación), y la otra llamada "conjunto de validación" o *test* (que está excluido del conjunto de entrenamiento). Los datos deben ser seleccionados de forma aleatoria para los dos conjuntos. Entre más datos, se mejorará el aprendizaje y por lo tanto mejorará la evaluación.

K-fold cross-validation, realiza una partición del conjunto de datos en K conjuntos disyuntos del mismo tamaño (El K permite hacer un balance entre sesgo y varianza). $K-1$ partes se utilizan para el entrenamiento, y 1 parte para la validación o *test*. Este proceso se repite K veces y finalmente se agregan las métricas de evaluación. Se estima que los mejores resultados se obtienen con un valor de K entre 5 y 10 [9].

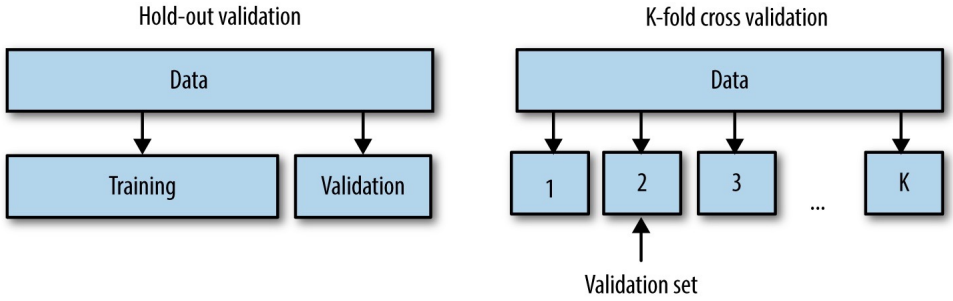


Figura 4.4: *Hold-out* y *K-fold cross-validation* [5]

4.2.1.5. Over-fitting y under-fitting

Al momento de entrenar un modelo con una buena capacidad predictiva, se debe tener un balance entre el sesgo (*bias*) y la varianza. Debido a que un desbalance entre ellos puede llegar a producir un *under-fitting* u *over-fitting*. Matemáticamente la relación entre el sesgo (*bias*) y la varianza, puede expresarse como:

$$\frac{db}{dc} = -\frac{dV}{dc}$$

Donde b es el sesgo (*bias*), c la complejidad del modelo, y finalmente V la varianza [21].

Over-fitting ocurre cuando el modelo se ajusta excesivamente con respecto a los datos del conjunto de entrenamiento. Este ajuste excesivo se enfoca en datos que son más bien "ruido", memorizándolos y haciendo que no se generalice el conocimiento con

respecto a un conjunto de datos nuevos. En este caso el *bias* es mucho menor que la varianza, y la complejidad del modelo (es decir, polinomios de un grado más alto) crece junto con la varianza [6, 19] (Ver figura 4.5, polinomio grado 9).

Para el caso de *under-fitting*, el modelo tiene un *accuracy* muy bajo, incluso con respecto al conjunto de entrenamiento. La mayoría de veces, cuando esto ocurre, se opta por buscar otro modelo que mejor se ajuste a los datos. En este caso el *bias* es mucho mayor que la varianza [6] (ver figura 4.5, polinomio grado 0).

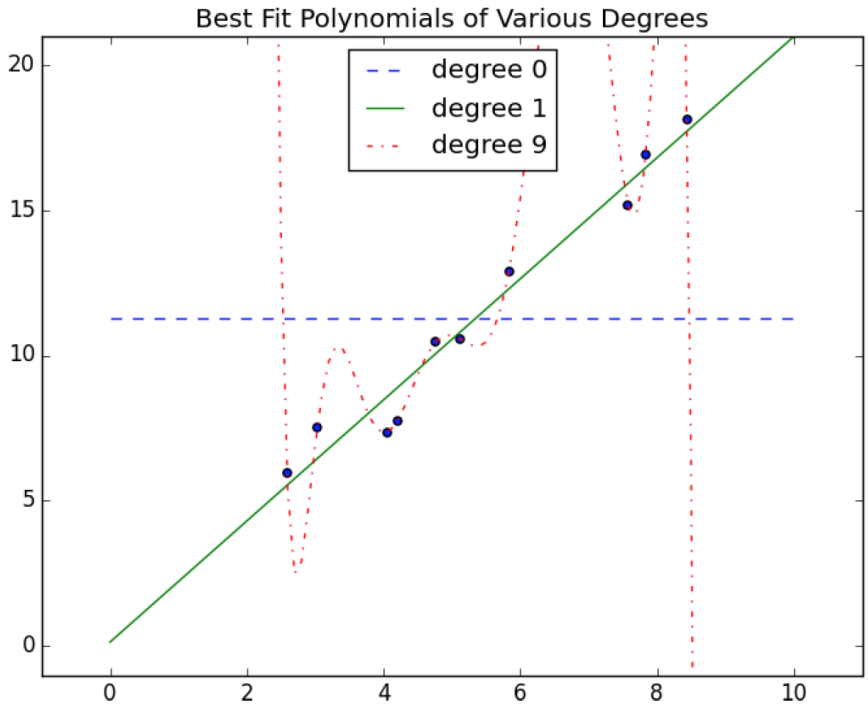


Figura 4.5: Representación gráfica de *over-fitting* y *under-fitting* [6]

En la figura 4.5 la línea con el polinomio de grado 1 expresa la cantidad de polinomios apropiados, evidenciando el punto medio entre el *over-fitting* y el *under-fitting*.

4.2.1.6. Métricas de evaluación

Una vez construido el modelo de ML, es importante conocer sus criterios de evaluación, para así determinar que tan bien realiza el trabajo de clasificación. La evaluación generalmente se realiza con la tasa de error del modelo y se calcula como:

$$T_E = \text{Número de errores/Número total de casos}$$

Existe una matriz, llamada la matriz de confusión, que ilustra mediante una tabla de contingencia la distribución de los errores cometidos por el modelo con respecto

Tabla 4.1: Matriz de confusión para problemas de clasificación [9]

		Clase predicha		
		Positivo(1)	Negativo(0)	Total
Clase verdadera	Positivo(1)	VP	FN	$VP + FN$
	Negativo(0)	FP	VN	$FP + VN$
	Total	$VP + FP$	$FN + VN$	N

a el distinto número de clases C [18]. En esta tabla de contingencia, se intersecta la variable predicha por el modelo con la variable que guarda la verdadera clasificación, como se puede apreciar en la Tabla 4.1.

Los elementos sombreados que se encuentran en la diagonal principal de la matriz de confusión (VP y VN) muestran la cantidad de instancias correctamente clasificadas. Mientras que las demás casillas muestran los diferentes tipos de error de clasificación:

- Tipo I: Falsos positivos (FP)
- Tipo II: Falsos negativos (FN)

De los elementos de la matriz de confusión, también se pueden extraer otro tipo de indicadores que son de gran utilidad al momento de evaluar el modelo [9], de los cuales se encuentran:

- **Tasa de correctitud (*Accuracy*)** $= (VP + VN)/(VP + VN + FP + FN)$
- **Probabilidad de error** $= (FP + FN)/(VP + VN + FP + FN)$
- ***Precision*** $= VP/(VP + FP)$: Probabilidad de dado a que la predicción es positivo, esta sea un Verdadero positivo.
- ***Recall*** $= VP/(VP + FN)$: Probabilidad de dado a que es un Verdadero positivo, la predicción sea positivo.
- ***Specificity*** $= VN/(VN + FP)$: Probabilidad de dado a que es un Verdadero negativo, la predicción sea negativo.
- **Coeficiente de concordancia Kappa:** $\mathcal{K} = (\text{O.A} - \text{E.A})/(1 - \text{E.A})$: Utilizado para *datasets* desbalanceados. Sustrae el efecto de concordancia por suerte (E.A) del valor del *Accuracy* (O.A).
- **F1-Score** $= 2 \times (\frac{Precision \times Recall}{Precision + Recall})$: Mide el balance entre *Precision* y *Recall*.

4.2.2. *Aprendizaje no supervisado*

El aprendizaje no supervisado a diferencia del aprendizaje supervisado, no cuenta con una variable objetivo $\mathcal{Y}$ (sus datos no se encuentran etiquetados). El aprendizaje no supervisado, es más aplicable que el aprendizaje supervisado, ya que no requiere de un humano experto para clasificar los datos [19].

La meta principal del aprendizaje no supervisado no es la predicción sino la búsqueda de una estructura, o de un nuevo punto de vista en los datos. Usualmente es realizado durante la parte exploratoria de los datos [9].

Algunas de las tareas que se pueden realizar con el aprendizaje no supervisado son:

- Segmentación (*Clustering*).
- Reglas de asociación.
- Reducción de dimensionalidad.
- Detección de anomalías.

4.2.3. *Reinforcement Learning*

"*Reinforcement Learning*, es un tipo de ML auto-evolutivo que nos lleva cada vez más cerca a lograr la verdadera inteligencia artificial"[22].

Reinforcement Learning, es una rama de ML, donde el aprendizaje ocurre a través de la interacción de un sistema o agente con su entorno. Este aprendizaje está orientado hacia una meta, en donde cada acción realizada por el sistema o a gente es recompensado con un *reward* (premio) por su entorno, este premio es positivo si la acción se acerca a la meta, y negativo cuando la acción se aleja de la meta. RL es completamente diferente en comparación con otros paradigmas pertenecientes al campo de ML, debido a que su diferencia más notoria radica en que RL no necesita datos históricos (*datasets*) para extrapolar y generalizar el conocimiento, como es el caso del aprendizaje supervisado, sino más bien aprende de su interacción directa con el entorno. [22].

4.2.3.1. Elementos de RL

- **Agente:** Programa o programas que toman decisiones inteligentes (son aprendices).
- **Política** π : Define el comportamiento del agente en un entorno dado.
- **Función valor:** expresado como $\mathcal{V}(s)$, es igual a la cantidad total esperada de *reward* recibida por el agente desde un estado inicial. Denota que tan conveniente es para un agente permanecer en un estado particular.
- **Modelo:** El modelo es la forma del agente representar el entorno. El aprendizaje puede ser de dos maneras:
    1. **Aprendizaje basado en el modelo:** El agente utiliza una información previamente aprendida para llevar a cabo una tarea.
    2. **Aprendizaje libre del modelo:** El agente adopta una estrategia "ensayo-error" para llevar a cabo la acción correcta.

4.2.3.2. Procesos de decisión de Markov

Los procesos de decisión de Markov propuestos por Richard Bellman, proveen un marco matemático para modelar y dar solución al problema de RL [22]. Los PDM contienen los siguientes cinco elementos:

1. Un conjunto de estados (S) en los cuales el agente puede permanecer.
2. Un conjunto de acciones (A) que pueden ser ejecutadas por el agente, para ir de un estado a otro.
3. Una probabilidad de transición ($P^a_{ss'}$), que es la probabilidad de ir de un estado (s) a otro estado (s') ejecutando la acción (a).
4. Una probabilidad de *reward* ($R^a_{ss'}$), que es la probabilidad de que el agente obtenga un *reward*, por moverse de un estado (s) a otro estado (s') ejecutando la acción (a).
5. Un factor de descuento (γ), que controla la importancia inmediata y futura de los *reward*.

4.2.3.3. Q-Learning

El algoritmo Q-Learning es muy popular. En este algoritmo no es importante el valor de los estados, sino más bien la principal preocupación es el valor del par estado-acción. Es el efecto de ejecutar una acción A en un estado S. El valor Q, se guarda en una tabla llamada *Q-table*, y su valor se calcula de la siguiente manera:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma maxQ(s',a') - Q(s,a)) \tag{4.1}$$

Donde r es el reward, γ es el factor de descuento, y α es la razón de aprendizaje, que sirve para la convergencia. [22]

Aquí se trata de expresar de que por cada acción que tomamos, la *Q-table* se actualiza para incluir un *reward* positivo o negativo. Es así que a punto de "ensayo-error", la *Q-table* va llenando sus valores. Y dado un estado en el cual se encuentre el agente, siempre se busca realizar una acción tal que desde ese estado sea la máxima o la que mejor *reward* obtenga.

Para ilustrar un poco este proceso se tienen los siguientes pasos:

1. Se inicializa la *Q-table* con valores arbitrarios (generalmente son 0)
2. Se toma una acción desde un estado usando una política *epsilon-greedy* ($\epsilon > 0$) y se mueve al estado nuevo.
3. Se actualizan los datos de la *Q-table* con la ecuación (4.1).
4. Se repiten los pasos 2 y 3 hasta llegar a un estado terminal.

4.3. *Ciberseguridad*

La ciberseguridad tiene como pilares fundamentales la confidencialidad, integridad, y la disponibilidad de los sistemas informáticos. Debido a el rápido crecimiento de las TICS, nuevas vulnerabilidades en los sistemas informáticos están emergiendo con mucha frecuencia, requiriendo así de un esfuerzo constante para mitigar los riesgos (esto implica la no existencia de un sistema informático 100 % seguro) [23]. Se han propuesto métodos para la evaluación de amenazas cibernéticas, estos métodos han permitido de cierta forma crear defensas contra los ataques informáticos. Algunos de los más representativos son:

- Análisis estático.
- Análisis dinámico.
- Análisis híbrido.

4.3.1. *Malware*

Malware, abreviación de *Malicious Software*, es un *software* diseñado con el fin de hacer daño a un computador, *Smartphone*, servidor, entre otros [24]. No obstante los cibercriminales, en la mayoría de los casos los utilizan principalmente con ánimo de lucro, o activismo político [10]. Algunas de las familias más conocidas según Kaspersky lab [25] son:

- *Virus*
- *Ransomware*
- *Keylogger*
- *Spyware*
- *Trojan Horse*
- *Adware*
- *Rootkit*
- *Worm*
- *Miners*

4.3.2. *Análisis estático*

El análisis estático, se enfoca en encontrar comportamientos maliciosos en el código fuente, datos, o archivos binarios de la App, evitando ejecutarla de forma directa [26]. Sin embargo, [27] logra demostrar que es posible evadir este tipo de análisis, a partir de técnicas de ofuscación del código.

4.3.3. *Análisis dinámico*

El análisis dinámico, a diferencia del estático, se enfoca en analizar el comportamiento malicioso de la App en tiempo de ejecución, simulando gestos. Durante el análisis se tiene en cuenta los procesos en ejecución, la interfaz de usuario, conexiones de red, etc [28]. Sin embargo, existen técnicas que permiten evadirlo, ya que algunos *Malware* tienen la capacidad de detectar ambientes *sandbox* e inmediatamente comportarse de manera benigna [29].

4.3.4. *Análisis híbrido*

El análisis híbrido toma los elementos del análisis estático y el análisis dinámico para finalmente combinarlos [23].

4.3.5. *Exploit*

Un *exploit*, es cualquier ataque que que se aprovecha de ciertas vulnerabilidades en las aplicaciones, redes, o *hardware* [30].

4.3.6. *Adversarial Machine Learning*

Adversarial Machine Learning, es un método o técnica para engañar a los modelos de ML (aprendizaje supervisado) a través de entradas ($\vec{x}$) maliciosas [31]. Su principal propósito es atacar los modelos de ML y afectar su buen funcionamiento.

Los modelos de ML fueron diseñados con la premisa de que el *dataset* de entrenamiento y el *dataset* de evaluación pertenecen a la misma distribución estadística. Sin embargo, en la actualidad existen adversarios inteligentes y auto-adaptativos que de cierta forma pueden romper esta premisa haciendo que se comprometa la seguridad del modelo de ML [15].

4.3.6.1. Tipos de ataques en contra de ML

Existen tres grandes ramas que categorizan los diferentes ataques contra los modelos de ML [11].

1. ***Ataques influyentes***
    - **Ataques causativos:** Influyen sobre el aprendizaje sobre el modelo, ejerciendo control sobre el *dataset* de entrenamiento.
    - **Ataques exploratorios:** Realizan un *exploit* para que el modelo realice una clasificación errónea, pero no afectan el entrenamiento del modelo.
2. **Violación de seguridad**
    - **Ataques de integridad:** Comprometen la fiabilidad del modelo a través de falsos negativos.
    - **Ataques de disponibilidad:** Causan la denegación de servicio, usualmente a través de falsos positivos.
3. **Especificidad**
    - **Ataques focalizados:** Se enfocan en una instancia (registro) en particular.
    - **Ataques indiscriminados:** Encierran una amplia cantidad de instancias (registros).

4.3.6.2. Secure Learning

Uno de los riesgos en el uso de modelos basados en ML, consiste en que los atacantes pueden aprovecharse de la naturaleza auto-adaptativa de los modelos de ML y a través de un *exploit* intentar que estos fallen, es decir que los modelos de ML realicen mal una predicción o clasificación. [11] Define el término *Secure Learning* como la habilidad de un modelo de ML de tener un buen desempeño incluso en condiciones adversarias, es decir cuando Cibercriminales o atacantes intentan realizar un *exploit* en contra del modelo.

Para evaluar la seguridad de un sistema, se debe tener en cuenta [11]:

1. Se determinan los tipos de ataque al sistema.
2. Se evalúa la resiliencia del sistema con respecto a estos ataques.
3. Finalmente, se busca que el sistema se vuelva más robusto (es decir, que el modelo se vuelva menos vulnerable) ante estos ataques.

Estado del arte

Para el estado del arte del proyecto, se describen 3 trabajos previos que tienen propuestas similares.

5.1. *Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection*

En [14], se expone que aunque *Machine Learning* ha sido una gran ayuda para poder detectar y contrarrestar las crecientes y cambiantes amenazas en Android, también tiene ciertas vulnerabilidades y es propenso a recibir ataques por parte de los cibercriminales, haciendo que los modelos fallen al momento de hacer una clasificación.

El trabajo mencionado se basa en un *Framework* de ataque (Análisis estático) propuesto en un trabajo previo, para categorizar los potenciales escenarios de ataques en contra de los modelos de ML. Luego, se realiza la implementación de un conjunto de ataques evasivos contra "Drebin" (Un modelo de ML para detectar *Android Malware*), para evaluar de forma rigurosa la seguridad del mismo. Finalmente, se propone un paradigma simple y escalable de *Secure Learning*, que mitiga los ataques evasivos, aunque afectando un poco la tasa de detección sin los ataques.

5.2. *Poster: Towards Adversarial Detection of Mobile Malware*

En [7], se expone que los clasificadores de ML convencionales, fallan ante ciertos ataques. Toma en consideración los ataques a los modelos de ML llamados "*Poisoning attacks*", en donde el atacante tiene control sobre el *dataset*, y puede inyectar ciertos datos para confundir el modelo cuando está siendo entrenado, así generando una vulnerabilidad.

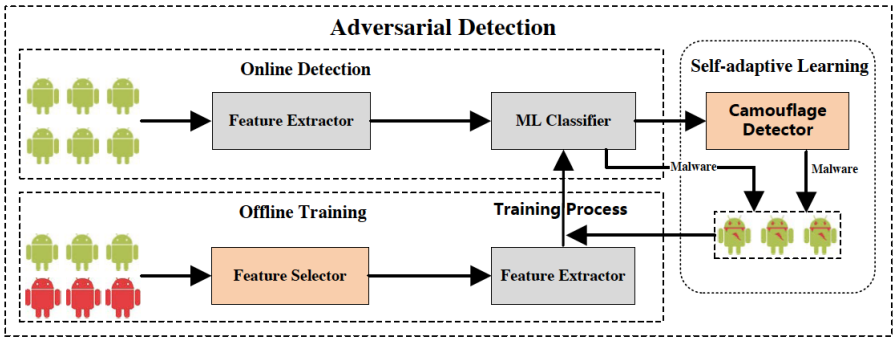


Figura 5.1: *KuafuDet Framework* [7]

En el trabajo mencionado, proponen un modelo adversario con tres tipos de ataques, para luego presentar un sistema iterativo de aprendizaje auto-adaptativo llamado "*KuafuDet*", que cuenta con dos fases, la primera (*offline*) en donde se seleccionan y se extraen las variables más significativas del conjunto de entrenamiento para entrenar el modelo, y la segunda (online) donde se realiza la clasificación con el modelo entrenado en la primera fase. Cuenta con un detector de camuflaje para realizar la detección de posibles inyecciones. Es el primer trabajo en detectar *Android Malware* en un ambiente adversario, reduciendo la tasa de falsos positivos, y mejorando el *accuracy* al menos 15 %.

5.3. *Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection*

En [32], se discute el papel tan importante que cumplen los modelos de ML para realizar la detección de *Android Malware*, sin embargo a estos modelos clasificatorios, les hace falta robustez con respecto a los datos adversarios, que son generados a través de pequeñas perturbaciones a los datos normales. Hasta el momento, los datos adversarios solo pueden engañar a modelos de ML cuyas variables de entrada sean sintáticas (es decir, los permisos de la App, llamados específicos del API,etc).

En el caso de los modelos cuyas variables de entrada sean semánticas (por ejemplo, Bytecode de Dalvik), los ataques con datos adversarios se vuelven más complejos, debido a que realizar perturbaciones en el bytecode no es trivial, además esto implica que una perturbación en el código pueda afectar directamente la funcionalidad original de la App.

En el trabajo mencionado, se propone un método de ataque para realizar perturbaciones óptimas en los archivos APK, generando datos adversarios. En contraste con los trabajos recientes, este método logra generar datos adversarios para algunos modelos cuyas variables de entrada son semánticas (por ejemplo, *control-flow-graph*, y también Dalvik). Al evaluar las manipulaciones hechas con los *datasets* de *Drebin* y *MaMadroid*, la tasa de detección decrementó de 96 % a 1 en MaMaDroid, y de 97 % a 1 % en Drebin.

5.4. *Matriz de estado del arte*

En la siguiente matriz, se realiza una resumen de lo que se encuentra en el estado del arte. Las columnas representan el número de la sección de los trabajos discutidos, y las filas los criterios de comparación entre estos.

Dentro de los criterios de comparación se encuentran:

- **Reinforcement Learning:** Este criterio dice si se utilizó RL como técnica fundamental para la generación de datos adversarios.
- **Adversarial Training:** Este criterio dice si se implementó un AT para mejorar la robustez de los modelos.
- **Attack framework:** Este criterio dice si se propone un marco de trabajo para realizar un ataque adversario al modelo.
- **Network features:** Este criterio dice si se utilizaron variables de tráfico de red para discriminar entre una Android App bengina o maligna.
- **Recomendaciones de Secure Learning:** Este criterio dice si se proponen recomendaciones de SL para mejorar la seguridad en los modelos de ML (en este caso de detección de Android Malware).

Tabla 5.1: Matriz resumen del estado del arte

Criterios / Papers	5.1	5.2	5.3	**Proyecto de grado**
Reinforcement Learning	NO	NO	NO	Sí
Adversarial Training	NO	NO	NO	SÍ
Attack framework	NO	SÍ	SÍ	SÍ
Network features	NO	NO	NO	SÍ
Recomendaciones de Secure Learning	NO	NO	NO	SÍ

Metodología

Para la metodología del proyecto a nivel global, se siguen los lineamientos de CRISP-DM (*Cross-industry standard process for data mining*) [8]. Debido a que es una metodología especialmente diseñada, para resolver problemas que conciernen a la ciencia de datos. Esta metodología cuenta con seis fases.

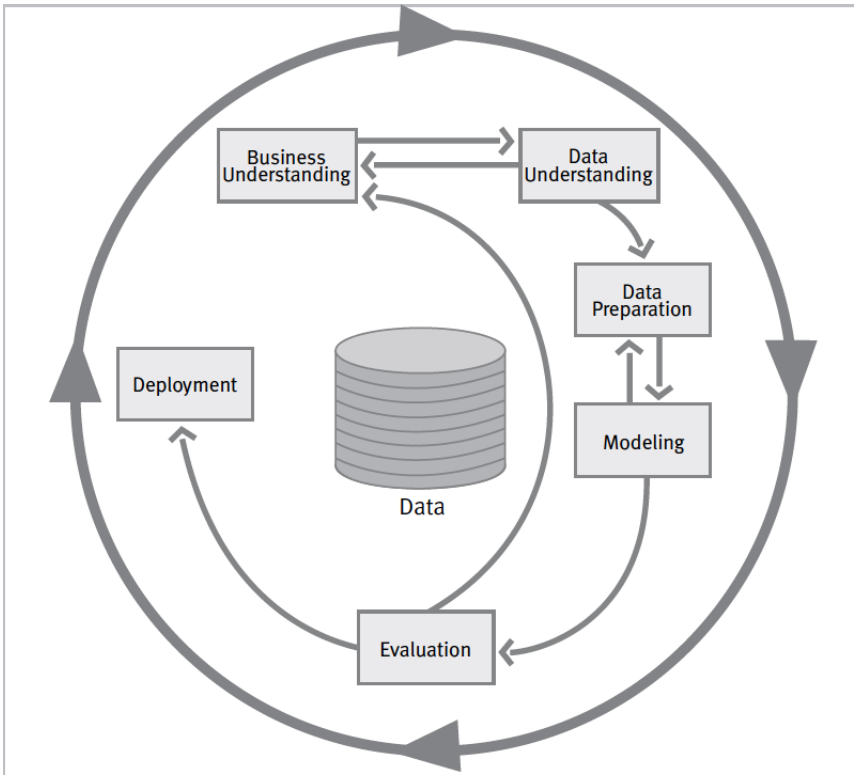


Figura 6.1: Fases de CRISP-DM [8]

6.1. *Modelo de defensa (CRISP-DM)*

6.1.1. Entendimiento del negocio

Para el entendimiento del negocio, se procedió a realizar un estudio de estado del arte, y se realizó la revisión de un trabajo previo [1].

6.1.2. Entendimiento de los datos

En la fase de entendimiento de los datos, se tomó el *dataset* de características de red utilizado en un trabajo previo [1], cuya cardinalidad es de 7845 registros, para luego realizar una exploración y descripción de los datos. El *dataset* cuenta con 10 variables:

- (R1): Paquetes TCP
- (R2): Paquetes distintos TCP
- (R3): IP externas
- (R4): Volumen de bytes
- (R5) Paquetes UDP
- (R6) Paquetes de la aplicación fuente
- (R7) Paquetes de la aplicación remota
- (R8) Bytes de la aplicación origen
- (R9) Bytes de la aplicación remota
- (R10) Consultas DNS, número de consultas DNS.

Incluyendo además una variable adicional llamada *type*, que puede tomar valores de *malicious* o *benign*.

6.1.3. Preparación de los datos

En la fase de la preparación de los datos, se procedió a realizar la limpieza de los datos, eliminando los datos atípicos, y aquellas columnas donde tuviese 0's o NaN's en más del 50 % de los datos. Finalmente se procede a realizar un escalamiento (Normalización) de los datos, para facilitar su manejo. También se realizó un análisis descriptivo de los datos.

6.1.4. Modelado

En la fase de modelado, se entrenaron seis modelos de ML, entre los cuales se encuentran:

- *Naive Bayes*
- *Random Forest*
- *KNN*
- *SVM*
- *Logistic Regression*
- *Decision Tree*

Para entrenar dichos modelos, se realizó un *hold-out* con un 75 % del *dataset* para entrenamiento, mientras que el otro 25 % se utilizó para evaluar la capacidad de generalización de cada modelo.

6.1.5. Evaluación

En la fase de evaluación, se tuvo como uno de los criterios de selección que el *Accuracy* fuese mucho mayor que el *baseline*. Además se obtuvieron y se tuvieron en cuenta durante este proceso otras métricas como:

- Matriz de confusión
- *Precision*
- *Recall*
- *F1-score*
- *Accuracy*
- *Kappa*

Finalmente, teniendo en cuenta estos criterios se selecciona el mejor modelo para dicho problema, y se serializa en un archivo con extensión ".sav".

6.1.6. Despliegue

En la fase de despliegue, el modelo de ML seleccionado, se lleva a producción. Pero en el caso particular del presente proyecto, esta fase no aplica debido a que el propósito del mismo es mejorar la robustez del modelo antes de llegar a esta fase.

6.2. *Ataque al modelo de defensa (Exploit)*

Para realizar el *exploit* al modelo de defensa, se utilizó un enfoque *white-box*, es decir en este caso se tiene acceso al *dataset*, con el cual el modelo fue entrenado, a la interacción con el modelo, y las variables. Como método de programación se utilizó *Reinforcement Learning*. El modelo de defensa y las variables del *dataset* de entrenamiento, representan el medio y las acciones en el algoritmo de RL.

Al momento del ataque, se generaron un total de 500 datos adversarios a través de un *exploit* que incorpora RL. El procedimiento para realizarlo es el siguiente:

1. Se seleccionan las variables más significativas a atacar del modelo de defensa (Teniendo en cuenta *Feature importances* del modelo de defensa) para disminuir la complejidad computacional al momento del ataque. *Feature importances* (método que hace parte de la librería Scikit-learn para python), evalúa la importancia de cada variable del modelo al momento de que el modelo realizara la clasificación [33].

2. Se ajustan los parámetros correspondientes del algortimo de RL, tales como: `greedy_factor=0.9, learning_rate=0.1, discount_factor=0.9`. Y se crea un método (`choose_action`) con el cual se selecciona la acción (variable a atacar) según los valores en la Q-table.

3. Se procede a cargar el mejor modelo de defensa (desde un archivo serializado ".sav"), y el *dataset* con cual el modelo fue entrenado.

4. Se crea un *dataset* auxiliar, con la misma estructura que el *dataset* de entrenamiento del modelo de defensa, con una muestra aleatoria de 500 datos maliciosos (*malicious*) tomados del *datset* de entrenamiento del modelo de defensa y eliminando la columna de clasificación (*type*) del *datset* auxiliar.

5. Se modifica el algoritmo de RL para realizar perturbaciones (es decir, reemplazos en los valores) a las variables del *dataset* auxiliar con los 500 datos maliciosos. Estas perturbaciones tienen un rango específico por cada variable, en donde el mínimo es el dato mínimo del *dataset* de entrenamiento y el máximo, es el dato máximo del *dataset* de entrenamiento de cada variable respectivamente. Las perturbaciones se aplican reemplazando de forma aleatoria el valor original de las variables del *dataset* auxiliar, dichas variables (acciones), como se mencionó anteriormente, son escogidas según los valores de la Q-table. Se debe aclarar, que la aleatoriedad está limitada al respectivo rango de cada variable.

6. Se procede a crear un método (`get_env_feedback`), cuyo propósito es asignar el respectivo *reward* para cada acción (variable a atacar) al momento de realizar una perturbación. Positivo (+2) cuando el modelo de defensa previamente cargado, clasifica un registro del *dataset* auxiliar como *benign*, y Negativo(−0) cuando es clasificado como la verdadera clase, es decir *malicious*.

7. Se procede a crear un método que imprime la cantidad de pasos que le tomó al algoritmo de RL para que el modelo clasificara un registro del *dataset* auxiliar como *benign* (es decir, convertir el registro en un dato adversario).

8. Finalmente se ejecuta el algoritmo, se guardan los valores de la Q-table y los 500 datos adversarios generados en un archivo ".sav" y un archivo ".csv" respectivamente.

6.3. *Adversarial training*

Adversarial training, incorpora un *dataset* híbrido, que incluye los datos normales de entrenamiento con la adición de datos adversarios [34]. Para realizar el *Adversarial training*, se tuvo en cuenta un trabajo realizado [34], donde se expone que este método fue uno de los primeros para contrarrestar los ataques adversarios en contra de los modelos de ML. Además, se ha llegado a un consenso que este método puede mejorar la robustez de un de un modelo de ML con respecto a los ataques adversarios. El procedimiento para realizar *Adversarial training* es el siguiente:

1. Se carga tanto el *dataset* de entrenamiento, y el *dataset* de los 500 datos adversarios.
2. Se procede a adicionar la columna de clasificación (*type*) en el *dataset* de los datos adversarios con la etiqueta *malicious*.
3. Se realiza un *hold-out* con los 500 datos adversarios, dejando el 75 % para entrenamiento y el otro 25 % para evaluar.
4. Se agregan los datos adversarios (el 75 %) al *dataset* de entrenamiento.
5. Se realiza el entrenamiento de un modelo de ML (*Random forest*) con la agregación de los datos adversarios.
6. Se realiza la evaluación con el conjunto de evaluación del modelo de defensa para verificar que el nuevo modelo siga clasificando de manera correcta y no tenga un *bias* muy alto.
7. Finalmente, se evalúa utilizando el conjunto de evaluación (25 %) de los datos adversarios con la técnica *cross-validation* (k=5), para verificar que su robustez con respecto a nuevas muestras adversarias.

6.4. *Cuadro resumen de la metodología de ataque y AT*

A continuación se ilustra a través de un gráfico la metodología para realizar el entrenamiento del modelo de defensa, y el *Adversarial Training* para mejorar la robustez del modelo.

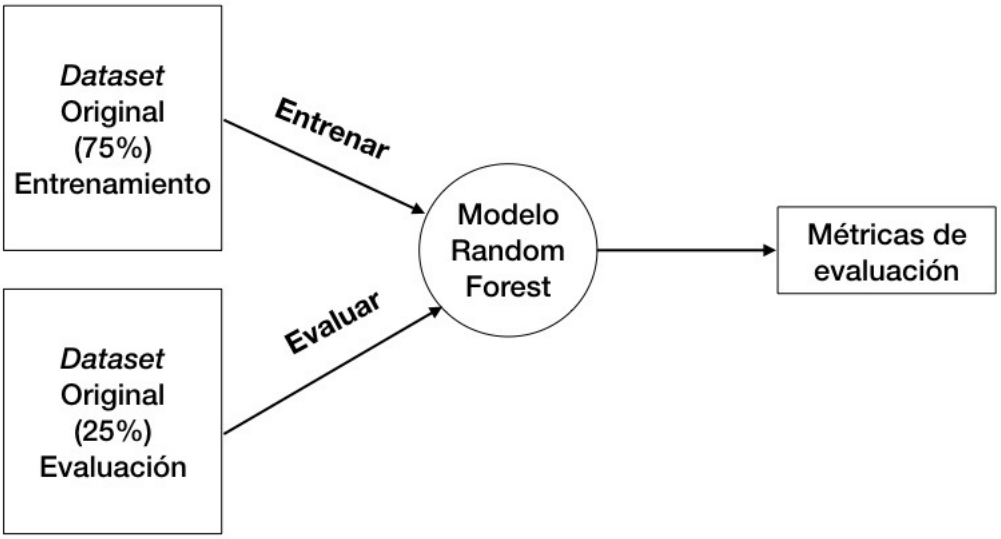


Figura 6.2: Entrenamiento y evaluación del modelo de defensa (ilustración propia)

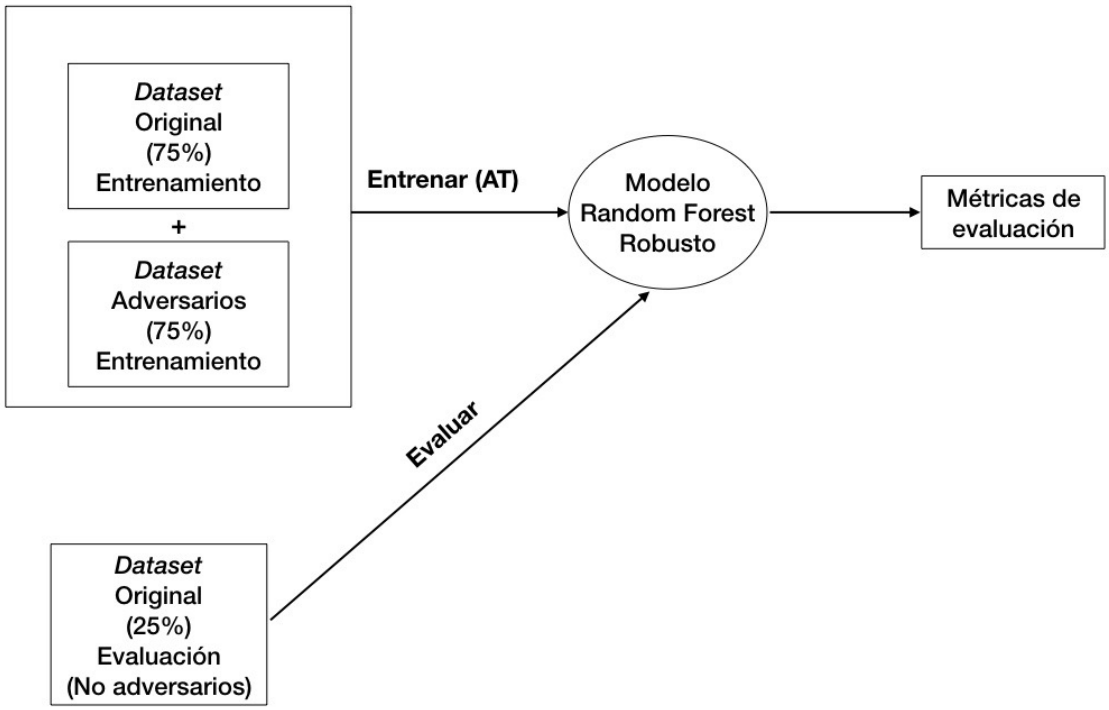


Figura 6.3: *Adversarial training* y verficación con el conjunto de evaluación del *dataset* original (ilustración propia).

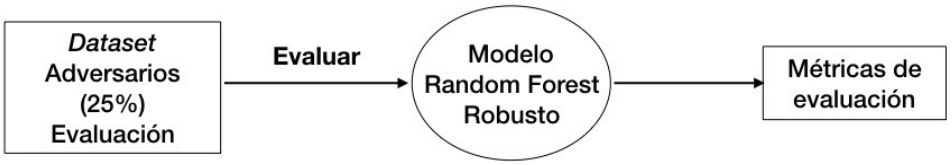


Figura 6.4: Evaluación con solo datos adversarios (ilustración propia)

Resultados y experimentos

La sección de experimentos del proyecto se dividirá en tres categorías. La primera, las tecnologías empleadas, la segunda, el modelo de defensa y finalmente el *exploit* al modelo.

7.1. *Tecnologías empleadas*

Durante el proyecto se utilizó como lenguaje de programación Python 3.7.1 junto con Jupyter notebook y la librería de *Machine Learning* scikit-learn 0.20.1.

7.2. *Modelo de defensa (CRISP-DM)*

En el fase de preparación de los datos se realizaron las siguientes actividades:

- En la actividad de limpieza de los 7845 registros, se eliminaron 13 datos atípicos, quedando con un total de 7832 registros. De los cuales 4691 eran de tipo *benign* y 3141 eran de tipo *malicious*.
- Se utilizó el método RobustScaler de la librería scikit-learn para realizar la normalización de todas las variables independientes.
- Con base en los datos obtenidos, y en un trabajo previo [1], se evidencia en las capturas de tráfico de red que el promedio de la cantidad de paquetes TCP enviados es de 1,72 y 0,55 para la clase *benign* y *malicious* respectivamente. Por otra parte, se evidencia también que otra variable (`remote_app_packets`) en promedio tenía 2,35 y 0,86 para la clase *benign* y *malicious* respectivamente.

Posteriormente, en la fase de modelado, se reentrenaron y evaluaron seis modelos de *Machine Learning* clasificatorios de *Android Malware* con las 10 características de red mencionadas anteriormente como variables predictivas a partir de los datos utilizados en un trabajo previo [1].

Tabla 7.1: Resultado en el desempeño individual de los seis modelos

Algoritmo	Métricas de desempeño							
	Precision		Recall		F1-score		Accuracy	Kappa
	benign	malicious	benign	malicious	benign	malicious	-	-
Naive Bayes	0.81	0.41	0.12	0.96	0.20	0.58	0.44	0.06
Random Forest	0.93	0.90	0.94	0.88	0.93	0.89	0.91	0.82
KNN: K=4	0.89	0.89	0.93	0.83	0.91	0.86	0.89	0.77
SVM	0.72	0.79	0.92	0.45	0.81	0.57	0.73	0.39
Logistic Regression	0.72	0.68	0.86	0.47	0.78	0.56	0.70	0.34
Decision Tree	0.91	0.86	0.91	0.85	0.91	0.86	0.88	0.76

En el caso del actual proyecto, el mejor modelo fue *Random Forest*, y se procedió a realizarle un ajuste a ciertos parámetros tales como: `n_estimators=250, max_depth=50,random_state=45`. Se logró mejorar el desempeño previamente encontrado por [1], pasando de un *accuracy* de 0.89 a 0.91, y aumentando el Kappa de 0.75 a 0.82.

Del mejor modelo para el problema en cuestión (*Random Forest*), se obtuvo sus *feature importances*, es decir, las variables más importantes tenidas en cuenta al momento de realizar la clasificación (ver Tabla 7.2). También se obtuvo la matriz de confusión y la curva ROC (ver figura 7.1 y figura 7.2 respectivamente).

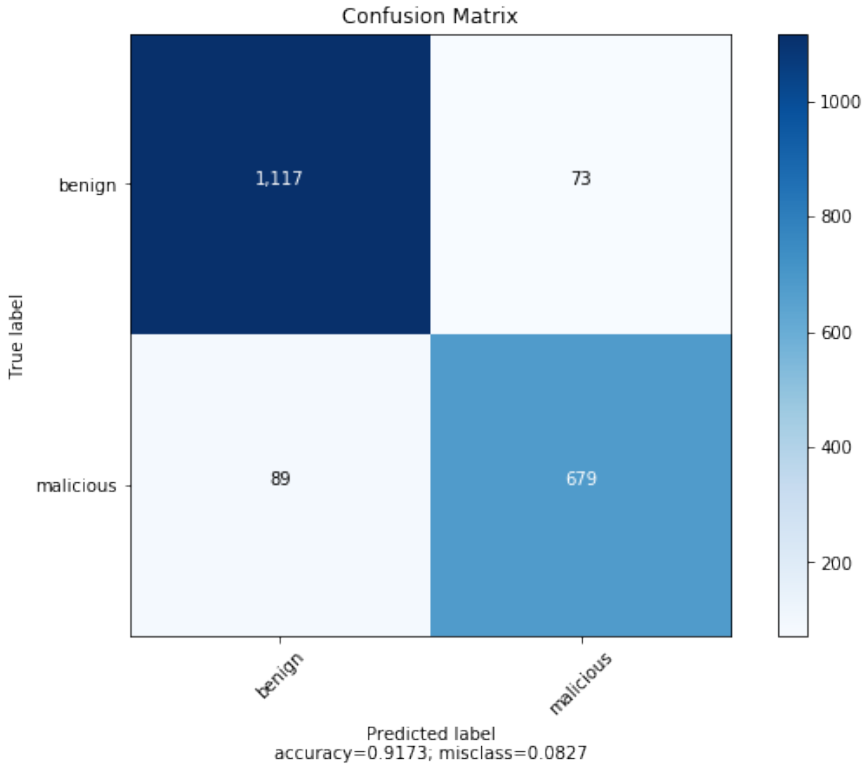


Figura 7.1: Matriz de confusión modelo de defensa

Como se había mencionado anteriormente, la matriz de confusión ayuda a ver los errores de clasificación del modelo. En este caso, observamos que en la diagonal principal (1117 y 69) fueron de la clase *benign* y *malicious* clasificadas de manera correcta respectivamente. Mientras que la otra diagonal (73 y 79) fueron de la clase *benign* y *malicious* clasificadas de manera errónea respectivamente.

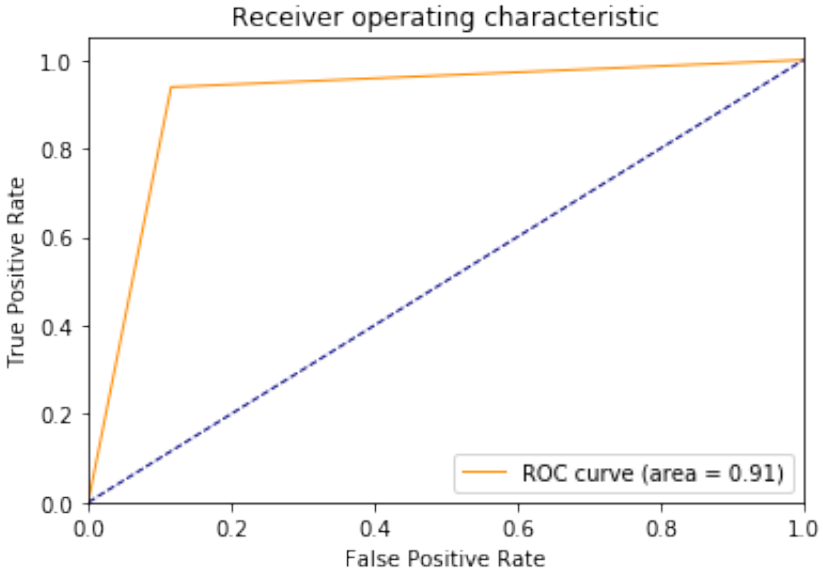


Figura 7.2: Curva ROC modelo de defensa

La curva ROC (*Receiver Operating Characteristics*), es una métrica donde se grafica la razón de los FP y la razón de los FN. Ayuda al momento de determinar que tan bien el modelo realiza la clasificación. En este caso un 91 % es una excelente medida, puesto que entre más cerca este valor se encuentre del 1 mucho mejor será el modelo [35].

Tabla 7.2: *Feature importances* de todas la variables (*Random forest*)

Feature	*Importance*
(1) source_app_bytes	0.221418
(2) remote_app_bytes	0.147509
(3) tcp_packets	0.142017
(4) remote_app_packets	0.134524
(5) vulume_bytes	0.127661
(6) source_app_packets	0.086725
(7) dns_query_times	0.050267
(8) dist_port_tcp	0.045439
(9) external_ips	0.043016
(10) udp_packets	0.001425

Como se había mencionado anteriormente, *feature importances*, permite visualizar qué variables son muy importantes para que *Random Forest* pueda tomar una decisión al momento de realizar la clasificación. *Feature importances* asigna un peso a cada variable y la suma en conjunto del peso de cada variable debe ser igual a 1. La tabla 7.2, se puede observar que la variable más importante es `source_app_bytes` y la menos importante es `udp_packets` (Están ordenadas de mayor a menor peso), por lo que en el proyecto se tomaron las 6 primeras variables más importantes para reducir costos computacionales como se había mencionado anteriormente.

7.3. *Ataque al modelo de defensa (Exploit)*

Para realizar un *exploit* a el modelo previamente entrenado (siguiendo con la metodología previamente presentada), se procede a generar un *dataset* auxiliar de 500 instancias, basado en perturbaciones de las 6 variables más importantes de la tabla 7.2.

Se encontró que luego de la generación de 20 datos adversarios, que la cantidad de pasos necesarios para realizar una perturbación exitosa fue sólo de 5, manteniéndose estable hasta llegar a la generación de los 500 datos adversarios. En los datos generados anteriormente, estos pasos variaban entre 6 y 15 pasos, pudiendo evidenciar la mejora en la eficiencia y eficacia de la Q-table al momento de elegir una acción, a medida que se iban generando nuevos datos adversarios hasta llegar a 500.

7.4. *Adversarial training*

Luego de realizar el *adversarial training*, se obtuvo sus respectivas métricas de evaluación, con respecto a el conjunto de evaluación del modelo de defensa para verificar que aún seguía clasificando de manera correcta.

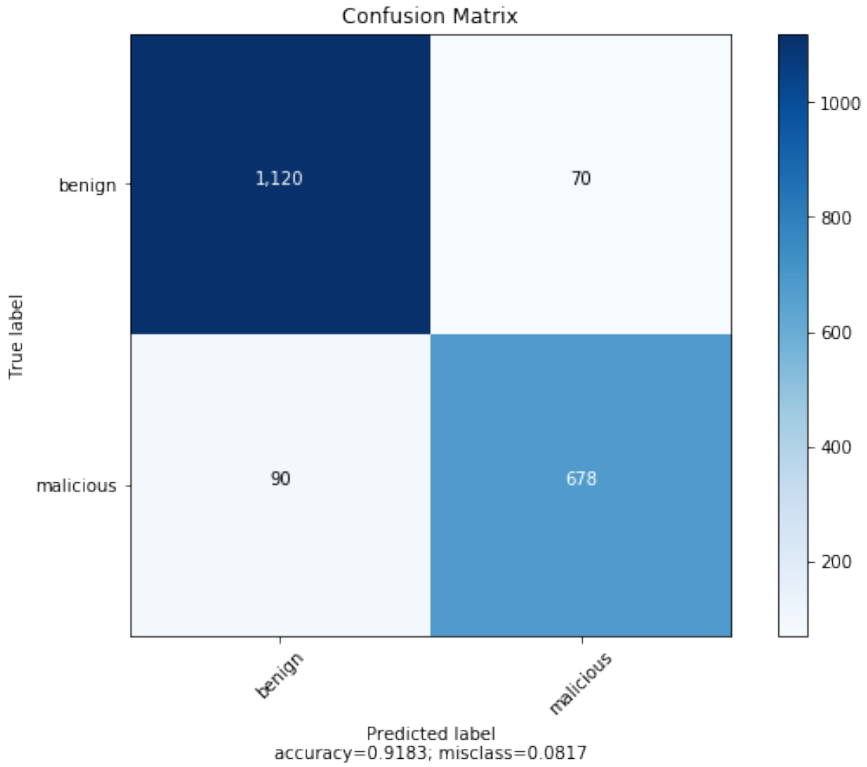


Figura 7.3: Matriz de confusión luego de realizar *Adversarial training*

Tabla 7.3: Resultado en el desempeño de *Random Forest* luego de realizar *Adversarial training*

Algoritmo	Métricas de desempeño							
	Precision		Recall		F1-score		Accuracy	Kappa
	benign	malicious	benign	malicious	benign	malicious	-	-
Random Forest Robusto	0.93	0.91	0.94	0.88	0.93	0.89	0.91	0.82

Efectivamente se puede observar, que las métricas son casi iguales al modelo de defensa (ver tabla 7.1 y 7.3), lo que implica que el modelo sigue teniendo un buen desempeño realizando la clasificación, incluso se puede observar una mejora muy pequeña en el *accuracy* (0,9183) con respecto a el modelo de defensa donde se obtuvo (0,9173). También podemos observar mejoras en la matriz de confusión donde hay 1120 clases de tipo benignas clasificadas correctamente con respecto al modelo de defensa donde se obtuvo 1117, y 90 clases de tipo malicioso clasificadas correctamente con respecto al modelo de defensa donde se obtuvo 89.

Luego de realizar la verificación de que seguía siendo un buen modelo, se procedió a

realizar una evaluación con respecto a el conjunto de evaluación de los datos adversarios (25 %), utilizando la técnica *cross-validation* con un $k = 5$.

Tabla 7.4: *cross-validation* para la evaluación con respecto a los datos adversarios

cross-validation	*Accuracy*
k=1	0.77
k=2	0.73
k=3	0.75
k=4	0.71
k=5	0.59
$\overline{k}$	0.71

Como se puede observar en la tabla 7.4, de reconocer 0 % de los datos adversarios, ahora reconoce hasta aproximadamente el 71 % en promedio de ellos, teniendo en cuenta que este 25 % de los datos adversarios no se encontraba en el *dataset* híbrido de entrenamiento. Se puede concluir por lo tanto que el modelo ahora es mucho más robusto ante ataques adversarios.

Recomendaciones de *Secure Learning*

En el desarrollo del actual proyecto, se trabajaron dos objetivos de la ciberseguridad, los cuales fueron la integridad porque mejorar la robustez del modelo nos ayuda a reducir la cantidad de falsos positivos, falso negativos y la disponibilidad porque si el modelo es un poco más seguro no estará fuera de servicio por realizar clasificaciones erróneas. Sin embargo, cuando es llevado a producción (*Deployment*), se deben tener en cuenta los tres objetivos de la seguridad mencionados anteriormente (confidencialidad, integridad, disponibilidad), esto implica muchas más cosas (como el sistema, el medio donde este interactúa) no solamente el modelo en cuestión. Para que un modelo de ML sea menos proclive a ataques adversarios, se deben tener en cuenta las siguientes recomendaciones:

1. Se recomienda agregar una nueva actividad en la fase de evaluación de la metodología *CRISP-DM*, donde se aplique la defensa proactiva. Es decir, en este caso el diseñador del modelo de ML, desarrolla estrategias de defensa (también robustez) ante posibles ataques adversarios antes de desplegar el modelo de ML, y así que este no sea tan vulnerable.

2. Se recomienda que el *dataset* de entrenamiento, junto con la interacción del modelo de ML no esté disponible al público en general, para evitar ataques de *white-box* y *poisoning attacks*.

3. Se recomienda el entrenamiento de modelos de ML, con algunos datos adversarios (que se pueden generar con el método desarrollado en el presente proyecto), para así mejorar la robustez del modelo [34] (confidencialidad).

4. Se recomienda que al llevar un modelo de ML a producción (Deployment), el sistema debe también estar seguro, como el medio en que este interactúa.

Contribuciones y entregables

9.1. *Contribuciones*

La idea del presente proyecto nace de un trabajo previo [1], donde se propone como trabajo a futuro la exploración de *Adversarial Machine Learning* para el mejoramiento de la seguridad en los modelos clasificatorios de ML. Dentro de los aportes del proyecto se encuentra:

- Creación de un método de programación (*Reinforcement Learning*) para descubrir vulnerabilidades y generar datos adversarios para un modelo de ML, desde el enfoque *white-box*.
- Mejorar la robustez de un algoritmo de clasificación de *Android Malware* basado en características de tráfico de red, propuesto en un trabajo previo [1].
- Propuesta de una nueva actividad que involucra evaluar y mirar los posibles ataques adversarios de un modelo de ML, en la fase de evaluación de la metodología *CRISP-DM*.

9.2. *Entregables*

Como entregables se tienen:

- Los *dataset* utilizados durante el proyecto.
- La Q-table con sus respectivos valores.
- El modelo de defensa.
- *Jupyter Notebook* con el algortimo de RL para realizar el ataque al modelo de defensa.
- *Jupyter Notebook* con las fases del ciclio de vida de CRISP-DM para el entrenamiento del modelo de defensa.
- *Jupyter Notebook* con el procedimiento de *Adversarial Training*.

Conclusiones y trabajo futuro

En el presente proyecto se realiza la implementación de *Secure Learning* a un modelo que permite predecir la clasificación de APKS *Android* como benignas o maliciosas (*benign,malicious*) a partir de variables de tráfico red propuestas en un trabajo previo [1]. Como parte de las conclusiones se encuentran:

- El modelo de ML basado en *Random Forest* mantuvo sus métricas luego del *Adversarial training*, es decir, hubo mejoras en la matriz de confusión luego de realizar el *Adversarial Training*, donde se obtuvo 1120 clases de tipo benignas clasificadas correctamente con respecto al modelo de defensa donde se obtuvo 1117, y 90 clases de tipo malicioso clasificadas correctamente con respecto al modelo de defensa donde se obtuvo 89.

- Luego del *adversarial training* el modelo pasó de reconocer 0 % de los datos adversarios, al 77 % de ellos (se debe aclarar que el conjunto de evaluación con los datos adversarios (la partición del 25 %)no se encontraba en el *dataset* híbrido de entrenamiento). se puede concluir por lo tanto, que el modelo ahora es mucho más robusto ante ataques adversarios.

- Es posible realizar un *exploit* a un modelo clasificatorio de Android Malware entrenado con características de tráfico de red utilizando RL.

- A través del ajuste de ciertos parámetros como: `n_estimators,max_depth,random_state` se pudo mejorar el *accuracy* del modelo de defensa.

Finalmente como trabajo futuro se propone lo siguiente:

- Mejorar el algoritmo de RL para realizar perturbaciones más precisas utilizando *Deep Reinforcement Learning* o *Generative Adversarial Networks* (*GANS*) propuestas en [36].

- Porponer un método de programación para realizar un *exploit* de un modelo de ML con un enfoque *black-box*

- Llevar a producción el modelo robusto (*Random forest*) del presente proyecto desplegándolo en un servidor del grupo de investigación i2t para que pueda realizar clasificación de *Android malware* a través del trafico de red, incorporado en el aplicativo móvil Snif.

Referencias

[1] C. Urcuqui, J. Delgado, A. Perez, A. Navarro, and J. Diaz, "Features to Detect Android Malware," *2018 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2018.

[2] "What is The OSI Model — Cloudflare."

[3] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2017.

[4] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning From Data. A short course*. 2012.

[5] A. Zheng, "Evaluating Machine Learning Models," 2015.

[6] J. Grus, *Data Science from scratch book*. O'Reilly Media, 2015.

[7] S. Chen, M. Xue, and L. Xu, "Poster: Towards adversarial detection of mobile malware," pp. 415–416, 2016.

[8] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, "CRISP-DM 1.0 Step-by-step," *ASHA presentation*, p. 73, 2000.

[9] J. Díaz, "Notas de clase, Aprendizaje Automático. Análisis de grandes volúmenes de datos.," Universidad Icesi, 2017.

[10] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The Evolution of Android Malware and Android Analysis Techniques," *ACM Computing Surveys*, vol. 49, no. 4, pp. 1–41, 2017.

[11] M. Barreno, A. D. Joseph, and J. D. Tygar, "Can Machine Learning Be Secure?," pp. 16–25, 2006.

[12] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, 2010.

[13] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," pp. 1–11, 2014.

[14] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–14, 2017.

[15] I. Goodfellow, P. McDaniel, and N. Papernot, "Making machine learning robust against adversarial inputs," *Communications of the ACM*, vol. 61, no. 7, pp. 56–66, 2018.

[16] L. M. Arboleda, *Programación en red con JAVA*. Universidad Icesi, 2004.

[17] A. F. PÉREZ, *SISTEMA DE ANÁLISIS DE TRÁFICO WEB PARA LA DETECCIÓN DE MALWARE EN DISPOSITIVOS ANDROID*. Proyecto de grado, universidad Icesi, 2018.

[18] B. Sierra Araujo, *Aprendizaje Automático: Conceptos básicos y avanzados*. PEARSON EDUCACIÓN, S.A, 2006.

[19] K. Murphy P., *Machine Learning: A Probabilistic Perspective*. The MIT Press Cambridge, Massachusetts. London, England, 2012.

[20] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*.

[21] S. Fortmann-Roe, "Understanding the Bias-Variance Tradeoff," 2012.

[22] S. Ravichandiran, *Hands-On Reinforcement Learning with Python: Master reinforcement and deep reinforcement learning using OpenAI Gym and TensorFlow*. 2018.

[23] C. Urcuqui, M. García, J. Osorio, and A. Navarro, *Ciberseguridad: Un enfoque desde la ciencia de datos*. Universidad Icesi, 2018.

[24] S. M. Robert Moir, "Defining Malware: FAQ — Microsoft Docs."

[25] Kaspersky Lab, "Types of known threats, General information."

[26] L. Batyuk, M. Herpich, S. A. Camtepe, K. Raddatz, A. D. Schmidt, and S. Albayrak, "Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications," *Proceedings of*

the 2011 6th International Conference on Malicious and Unwanted Software, Malware 2011, pp. 66–72, 2011.

[27] M. Sharif, A. Lanzi, J. Giffin, and W. Lee, "Impeding Malware Analysis Using Conditional Code Obfuscation," *Proceedings of the 15th Network and Distributed System Security Symposium - NDSS '08*, pp. 321–333, 2008.

[28] J. J. Drake, P. O. Fora, Z. Lanier, C. Mulliner, S. A. Ridley, and G. Wicherski, *Android™ Hacker's Handbook*, vol. 3. John Wiley & Sons, Inc., Indianapolis, Indiana, 2014.

[29] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage against the virtual machine: Hindering Dynamic Analysis of Android Malware," *Proceedings of the Seventh European Workshop on System Security - EuroSec '14*, pp. 1–6, 2014.

[30] AVAST Software, "What are exploits and how to protect yourself from them — Avast."

[31] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial Machine Learning at Scale," pp. 1–17, 2016.

[32] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection," 2018.

[33] "Feature importances with forests of trees — scikit-learn documentation."

[34] X. Wang, J. Li, X. Kuang, Y.-a. Tan, and J. Li, "The security of machine learning in an adversarial setting: A survey," *Journal of Parallel and Distributed Computing*, vol. 130, pp. 12–23, 2019.

[35] "Understanding AUC - ROC Curve – Towards Data Science."

[36] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," pp. 1–9, 2014.

Anexos

11.1. *Link del repositorio de los entregables*

Los entregables se encuentran disponibles en el repositorio del grupo i2t de la universidad icesi, en la carpeta dentro de la carpeta Android, donde se se encuentra otra carpeta con el título del proyecto. El link (privado) es el siguiente: `https://github.com/i2tResearch/Ciberseguridad.git`