Universidad Zaragoza
1542

Trabajo Fin de Grado

"Análisis y detección de anomalías en tráfico de red con Machine Learning"

"Analysis and detection of anomalies in network traffic with Machine Learning"

Autor/es

Álvaro Huarte Albás

Director/es

Raquel Lacuesta Gilaberte

Santiago Hernández Ramos

Escuela Universitaria Politécnica de Teruel

2019

“Análisis y detección de anomalías en tráfico de red con Machine Learning”

RESUMEN

Cada vez usamos de forma más habitual Internet para realizar transferencias bancarias, compartir datos personales o simplemente mandar correos con contenido importante. Sin embargo, muchos de estos datos se transmiten sin protección frente a ataques informáticos. Estos ataques están en auge en los últimos años y es por ello que el presente trabajo se centra en mejorar la seguridad de los datos en dichas transmisiones. En concreto nos centraremos en el concepto de la prevención y detección de ataques de red gracias a las técnicas de “Machine Learning”.

El objetivo de este Trabajo de Fin de Grado es realizar un análisis exhaustivo sobre diferentes tráficos de red con la finalidad de detectar con gran seguridad indicios de ataques informáticos. Para ello utilizaremos algoritmos basados en técnicas de “Machine Learning”. Estás técnicas se basan en el estudio de reconocimiento de patrones y el aprendizaje de computadoras sin la necesidad de intervención del ser humano. Finalmente se realizará una evaluación de los ataques detectados y algoritmos implementados y de su eficiencia.

De forma desglosada, en un primer lugar se realizarán ataques informáticos de tipo red. Cada uno de ellos posee un comportamiento y objetivo diferentes, gracias a esto podremos observar la influencia que tiene tanto en la computadora que usaremos como víctima, así como en el tráfico de red que posteriormente tendremos que capturar. Se buscarán ataques muy dispares ya que cuanto mayor sea dicha desigualdad mayor será el enriquecimiento de nuestro trabajo y mejores resultados podremos obtener.

En segundo lugar se realizará una captura de datos obtenidos a través del tráfico de red. Diferenciando dos tipos de datos, infectados los cuales serán capturados durante la realización de ataques y datos no anómalos cuyo origen será tráfico normal. Se utilizarán diferentes técnicas para la recolección de los mismos para su posterior comparación gracias a la cual podremos saber qué modelo de herramienta es más efectiva.

En último lugar se procederá al análisis de los datos capturados en el paso anterior. Para esta función se estudiarán algoritmos basados en “Machine Learning” más apropiados para ello, concepto muy interesante y el cual está siendo muy utilizado en la seguridad informática. Gracias a estas técnicas se pretende mejorar el funcionamiento de los algoritmos de detección de ataques y que ellos mismos aprendan a optimizar la detección de datos infectados.

PALABRAS CLAVE

Detección de anomalías, algoritmos de datos, seguridad informática, ciberseguridad, aprendizaje automático, ataque, código malicioso, análisis de datos, prevención de ataques, detección de ataques.

“Analysis and detection of anomalies in network traffic with Machine Learning”

SUMMARY

We increasingly use the Internet to make bank transfers, share personal data or simply send emails with important content. However, many of these data are transmitted without any protection against computer attacks. These attacks are really common nowadays and in consequence, this work focuses on improving data security in transmissions. Specifically, we will focus on the concept of prevention and detection of network attacks with the help of "Machine Learning" techniques.

The objective of this final project is to carry out an exhaustive analysis of the net traffics in order to detect with great certainty signs of computer attacks. To achieve this, we will use algorithms based on "Machine Learning" techniques. These techniques are based on the study of pattern recognition and computer learning without the necessity of human intervention. Finally, an evaluation of the detected attacks and implemented algorithms and the efficiency that they have obtained.

In a broken down manner, in the first place computer attacks of network type will be carry out. Each of them has a different behaviour and objective, thanks to this we can observe the influence that has both on the computer to use as a victim, as well as on the network traffic that we have to capture. Very diverse attacks will be sought to obtain better results.

Secondly, data is captured through network traffic. Differentiating two types of data, infected data which will be captured during the attacks and non-anomalous data whose origin will be normal traffic. Different techniques will be used to collect them for a later comparison to know which tool model is most effective.

Finally, the data captured in the previous step will be analysed. For this purpose, algorithms based on "Machine Learning" will be studied. Thanks to these techniques, it is intended to improve the functioning of the attack detection algorithms and that they themselves learn to optimize the detection of infected data.

KEYWORDS

Detection of anomalies, data algorithms, computer security, cybersecurity, machine learning, attack, malicious code, data analysis, attack prevention, attack detection.

CONTENIDO

1. INTRODUCCIÓN

Debido al incremento tecnológico en las últimas décadas prácticamente todas las personas que viven en un país cuyo desarrollo es medio-alto pueden tener a su disposición diferentes dispositivos que proporcionan acceso a internet como por ejemplo un ordenador de mesa, una Tablet o simplemente un teléfono móvil.

Otra de las consecuencias derivabas de esta evolución es la temprana edad con la que los niños tienen la posibilidad de acceder a internet sin ningún tipo de control; según unos estudios realizados en el Instituto Nacional de Estadística [29], en los últimos años esta cifra se ve reducida hasta los 2 años, algo que escandaliza ya que se recomienda que ningún niño menor de 13 años puede disfrutar de dicho privilegio.

Enfocando el tema hacia la seguridad, la sociedad actual no es consciente de la facilidad con la cual se envían datos personales a la red, accesibles para cualquier persona desde cualquier lugar del mundo. Lo alarmante es que los conocimientos de seguridad informática son los mismos en un niño de 10 años que en personas adultas con trabajos, cuentas bancarias, etc.

El avance de las tecnologías también posee una cara negativa, se están desarrollando diferentes técnicas a través de las cuales se pueden realizar delitos cibernéticos cuyo abanico de acciones es muy amplio, es posible incluir dentro de los mismos ataques tan simples como robar datos personales (contraseñas de Facebook u otra red social) hasta grandes cantidades de dinero o incluso el control total de sistemas. De hecho en España un tercio de usuarios de internet fue víctima de algún tipo de ataque en 2018 [30].

Estos ataques o delitos suelen tener una gran cantidad de objetivos entre los cuales se puede destacar el robo de capital o datos importantes pertenecientes a grandes empresas, bancos o gobiernos. Su porcentaje de éxito es abrumador, dejando tras ellos un rastro que muchas veces es imposible de seguir. Como consecuencia de estos ataques la seguridad informática se ha visto en incremento en la última década como única medida de protección contra los mismos.

Es muy importante trabajar en la prevención y mitigación de estos ataques, algo que para mucha gente solo es alcanzable para expertos informáticos, cuando tomando una serie de medidas preventivas es posible evitar muchos de ellos.

El proyecto consiste en el análisis de diferentes conjuntos de datos obtenidos a través de herramientas específicas durante la ejecución de varios ataques de red, cada uno de los cuales posee un comportamiento y objetivos diferentes que serán explicados a lo largo del trabajo. Dicho análisis se llevará a cabo gracias a algoritmos basados en Machine Learning.

2. MOTIVACIÓN Y OBJETIVOS

El entorno que rodea a la seguridad informática resulta atractivo para todas las personas sin la necesidad de que posean algún tipo de conocimiento sobre el mismo, este interés suele surgir debido a la gran representación que ha tenido el concepto de hacker en el mundo del cine y ficción. Sin embargo, cuando se lleva a cabo en la realidad es muy diferente.

Por mi parte durante los primeros años de ingeniería informática no encontraba una rama específica en la cual centrarme, observaba que muchos de mis compañeros ya sabían el camino que querían seguir, pero no era en mi caso. Todo esto cambió cuando me fui de Erasmus a

Dublín. Durante mi estancia cursé varias asignaturas optativas relacionadas con ciberseguridad, algo que fue muy enriquecedor y que me ayudo a descubrir el mundo del hacker.

El universo de la seguridad es muy amplio y durante la carrera de ingeniería informática solo en una o dos asignaturas trabajan este concepto por lo que de alguna forma intente aumentar los conocimientos aprendidos en Irlanda buscando un TFG sobre este campo.

Por un lado la realización de este proyecto tiene un objetivo personal, como ya he dicho anteriormente no poseo mucho conocimiento, por lo que quiero aprender y mejorar mis habilidades y técnicas de seguridad gracias a este proyecto. Por otro lado posee una serie de objetivos académicos que son los siguientes:

- Entender el concepto de Machine Learning y saber cómo puede ayudar a los algoritmos en la detección de anomalías.
- Distinción de los diferentes ataques propuestos, conociendo su funcionamiento y objetivos.
- Distinción de los diferentes conjuntos de datos obtenidos, analizando su origen y captación, y que ataques poseen.
- Detección con un gran porcentaje de acierto aquellas anomalías que se encuentren en el tráfico de red.
- Comprender con una comparativa y análisis de los resultados, que algoritmo es más eficiente y que ataque es más complicado de detectar.

3. ANÁLISIS

3.1 ESTADO DEL ARTE

El objetivo de esta sección es presentar todas aquellas técnicas y tecnologías que poseen alguna relación con el proyecto. Se intenta facilitar al lector la comprensión del diseño de la solución y aclarar todos los conceptos para que se puedan evaluar de forma correcta todas las contribuciones del trabajo.

En este apartado se analizará la situación actual de conceptos como Machine Learning, ciberseguridad y técnicas de aprendizaje automático, para posteriormente mencionar y proporcionar una mínima explicación de algunos proyectos y aplicaciones que poseen unos objetivos y características similares a los expuestos anteriormente en este trabajo.

3.1.1 DETECCIÓN DE ANOMALÍAS

La detección de anomalías es una técnica utilizada para descubrir patrones poco comunes que no concuerdan con un comportamiento normal, aquellos denominados “Outliers”. Posee una gran cantidad de aplicaciones y es utilizada a su vez en numerosos campos como pueden ser detectar intrusiones de tráfico de red (trabajo actual), monitorizar un sistema de salud o encontrar patrones maliciosos en transacciones de tarjetas de crédito [18].

Este tipo de sistemas compara la actividad monitorizada con un modelo predefinido anteriormente, el cual ha sido generado a través de un proceso de entrenamiento. El sistema realizará las comparaciones con este modelo y cualquier actividad que no siga los patrones de

normalidad propuestos será definida como anomalía. Es posible encontrar diferentes métodos para generar un modelo [39]:

- **Basado en estadística**: utiliza "thresholds" (umbral de confianza entre normal y anómalo), y operadores estadísticos básicos como media, varianza, etc.
- **Basado en reglas y especificaciones**: en base a la experiencia de expertos, acerca del comportamiento de modelos.
- **Basado en heurísticas y aprendizaje automático**: se genera a través de algoritmos de Machine Learning, es el método más extendido ya que se obtienen buenos resultados. Durante los siguientes capítulos se explicará con mayor detalle los distintos algoritmos que utilizan, se pueden introducir distinguiendo las tres categorías que se pueden encontrar: *aprendizaje no supervisado, aprendizaje supervisado y aprendizaje semi-supervisado.*

3.1.2 MACHINE LEARNING

El Machine Learning es una disciplina científica del ámbito de la Inteligencia Artificial, a través del cual se crean sistemas que aprenden automáticamente. Posee la capacidad de detectar patrones dentro de un conjunto de datos algo que posteriormente permite predecir situaciones. Estos cálculos son los que permiten aprender para generar y obtener decisiones y resultados fiables [1].

Los sistemas tradicionales, basados en firmas, son eficaces en la detección de ataques o virus porque poseen datos que los identifican como por ejemplo cadenas de texto, sin embargo, los sistemas de Machine Learning basados en la detección de anomalías identifican patrones de comportamiento desviados de la normalidad. Esto hace que no exista necesidad de una actualización constante debido al aprendizaje que desarrollan.

3.1.3 PROYECTOS Y APLICACIONES SIMILARES

Una vez explicados los dos conceptos más importantes que abarca este trabajo se van a presentar una serie de proyectos con objetivos similares a este y se finalizará con la muestra de aplicaciones que se encargan de detectar anomalías para empresas.

Se pueden encontrar una gran cantidad de trabajos que podrían ser equitativos, en consecuencia se ha tenido que realizar una selección de los mismos reduciendo el número a tres. Todos ellos [13], [14] y [15] aplican técnicas que están basadas en Machine Learning para la detección de ataques usando una serie de algoritmos específicos, el mayor parecido es con [14] ya que se comparte el algoritmo de Isolation Forest aunque enfocado de distinta manera ya que los conjuntos de datos analizados son diferentes.

Las aplicaciones Netskope [17] y Cleverpy [16] ofrecen a empresas la posibilidad de detectar anomalías y localizar datos extraños en su sistema, algo que previene de forma muy eficaz los ataques que puedan recibir en un futuro.

Este trabajo posee un carácter innovador pese a la gran cantidad de trabajos parecidos que se pueden encontrar debido a dos situaciones, el análisis de datos capturados en tres entornos y situaciones diferentes (simulación de un ataque, conjunto de datos KDD'99 y NSL-KDD, realizando una comparativa de los mismos, sin embargo los otro trabajos visualizados suelen

ofrecer el análisis en un solo entorno específico) y la segunda razón es el número de ataques utilizados, tres en este caso, cuando normalmente se utiliza la comparativa de un máximo de dos. Los otros proyectos usan un gran número de algoritmos y un pequeño número de ataques.

4. CONCEPTO DE MACHINE LEARNING

En los últimos años ha evolucionado de una manera considerable, es un término muy relacionado con la robótica y la ciencia ficción por lo que la gente puede confundir las limitaciones reales que posee.

4.1 DEFINICIÓN

Machine Learning es una sub-rama de la gran conocida inteligencia artificial cuyo objetivo es proporcionar a las computadoras la capacidad de aprender sin que exista la necesidad de ser programadas explícitamente. En otras palabras crear programas que sean capaces de generalizar comportamientos a partir de información no estructurada.

Una forma de entender de forma más concisa este concepto es la comparación de su funcionamiento con el cerebro humano ya que son procesos semejantes. Cuando se visualiza una imagen el cerebro automáticamente absorbe una gran cantidad de datos complejos a través de los sentidos para posteriormente etiquetarlos a una definición almacenada previamente, ¿Pero qué se puede hacer para qué una computadora realice la misma tarea?

En la forma tradicional de programación, un programador construiría un modelo constituido por aquellas instrucciones que la computadora tendrá que seguir posteriormente para identificar dichos datos. Justamente esto es lo que se quiere evitar con el Machine Learning, la eclosión de una nueva forma de programación donde en lugar de proporcionar instrucciones específicas, la computadora usa ejemplos junto con el algoritmo donde deberá encontrar patrones en los datos para posteriormente convertirlos en instrucciones que el ser humano no sabía describir[1].

4.2 MODELOS

El aprendizaje automático de las computadoras se basa en la programación de ejemplos y en el uso del algoritmo que deberá encontrar los patrones. Es posible clasificar estos algoritmos en tres grupos en función del aprendizaje que poseen [2].

- **Aprendizaje supervisado:** este tipo de algoritmos se genera un modelo predictivo, basado en datos de entrada y salida. La palabra clave “supervisado” viene de la idea de tener un conjunto de muestra el cual ya se sabe a qué grupo pertenecen sus ejemplos. Este grupo de datos es el llamado datos de entrenamiento, de esta forma el algoritmo va aprendiendo a clasificar las muestras gracias al modelo inicial.
- **Aprendizaje no supervisado**: se intenta resolver problemas de los que no existe un conocimiento previo, no se tienen datos anteriores. Al contrario que el supervisado este modelo está basado solo en las entradas y se puede deducir la estructura de los datos en función de la relación que existe entre los mismos.
- **Aprendizaje semi-supervisado**: algoritmos que utilizan datos de entrenamiento tanto etiquetados como no etiquetados, normalmente la cantidad de datos etiquetados suele ser bastante pequeña.

El hecho de usar la inteligencia artificial para contraatacar a los ataques de ciberseguridad era una evolución natural, algo que era inevitable. La gota que colmó el vaso fue la automatización del malware [42] algo que impulso el uso del Machine Learning para contrarrestarlo.

4.3 IMPORTANCIA ACTUAL

El aprendizaje automático en el ámbito de red permite determinar de forma precisa anomalías en los patrones de tráfico, actividades de usuario y muchos otros aspectos. De esta forma los algoritmos pueden filtrar los patrones de tráfico para posteriormente aprender el "comportamiento" de la actividad y tomar decisiones. Por lo tanto, el Machine Learning se puede utilizar en una gran cantidad de ámbitos de la seguridad informática, en este caso se destacan dos situaciones donde es fundamental [3].

- **Sistema de detección de intrusiones (IDS):** consiste en la detección de cualquier ataque que ya haya sido ejecutado, de este modo el aprendizaje automático se puede usar para aumentar la confiabilidad de según qué métodos de seguridad. Se distinguen dos ramas dentro de este concepto, por un lado están aquellos basados en heurística el cual verifica el comportamiento del tráfico y siempre y cuando encuentre una anomalía generará una alarma. Por otro lado están aquellos que están basados en reglas y funcionan con ciertas vulnerabilidades consideradas como ataques.
- **Amenazas de tipo día cero:** consiste en la explotación de una vulnerabilidad perteneciente a un programa o aplicación, se llama así debido al poco tiempo del que disponen los desarrolladores para defender sus sistemas y recuperar el control. Los sistemas de detección basados en comportamientos son muy importantes en este tipo de ataques ya que no solo se centran en una base de datos con amenazas, estudian el comportamiento del programa y estudian si sus acciones están siendo intencionadas o no.

5. DISEÑO

5.1 ALGORITMOS ESCOGIDOS

En este apartado se ilustra los tipos de algoritmos que han sido seleccionados para la realización de este trabajo, explicando su funcionamiento y una justificación de la elección de los mismos.

5.1.1 DISTRIBUCIÓN GAUSSIANA

Algoritmo es de tipo semi-supervisado basado en un modelo de distribución, los cuales cobran mucha importancia en el cálculo de probabilidades, en este caso permite describir variables aleatorias que aparecen en muchos fenómenos actuales como por ejemplo en el análisis de datos, el cual se ha trabajo en este proyecto. Este tipo de distribución está basada en una función que puede observarse en la ilustración 1, su funcionamiento depende de dos parámetros, "μ" y "σ", que son su media y su desviación típica.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(1/2)[(x-\mu)/\sigma]^2}$$

Ilustración 1. Función Distribución Gaussiana.

- *Media:* valor medio que se establece entre unos datos o valores.
- *Desviación típica*: es una medida de dispersión, esto quiere decir que es la mejor forma de calcular el error que se puede obtener cuando queremos medir algo. La cual posee la fórmula matemática de la ilustración 2.

$$\sigma = \sqrt{\frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \cdots (x_N - \mu)^2}{N}}$$

Ilustración 2. Desviación Típica.

En otras palabras, con la distribución de Gauss (también es conocida como la distribución normal), se calcula la probabilidad de que varios valores ocurran dentro de ciertos rangos o intervalos. Una gráfica muy común es la ilustración 3

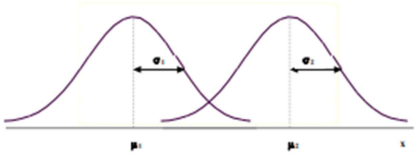


Ilustración 3. Gráfica Campana de Gauss.

Este tipo de "curva" generada por la distribución es la llamada Campana de Gauss donde es posible comprobar que la media es el punto más alto (posee el mayor número de repeticiones) y conforme nos alejamos de la misma el valor disminuye, [4].

Este tipo distribución basa su funcionamiento en la regla 68-95-99.7, también conocida como la regla empírica, donde se mide el porcentaje de valores que se encuentran dentro de la banda de cada color alrededor de una media. La medición de la ilustración 4 está realizada con valores de dos, cuatro y seis veces la desviación típica donde se puede comprobar el descenso del porcentaje de acierto conforme aumenta este valor [35].

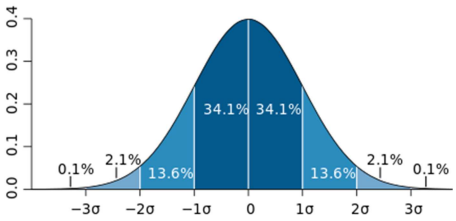
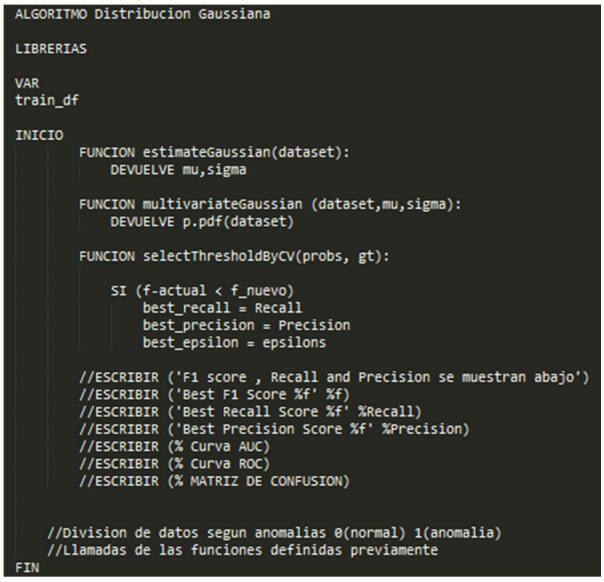


Ilustración 4. Gráfica Campana de Gauss con porcentajes.

Una vez explicado el funcionamiento de dicha distribución en la cual se basa el algoritmo, hay que entender cómo se va a aplicar al campo de la informática y a la detección de anomalías [5].

PSEUDOCÓDIGO DISTRIBUCIÓN GAUSSIANA

El código completo del algoritmo se puede encontrar en el **Anexo A.13**. A continuación hay una explicación del funcionamiento de las secciones más importantes del algoritmo a través de pseudocódigo estándar.



```
ALGORITMO Distribucion Gaussiana

LIBRERIAS

VAR
train_df

INICIO
        FUNCION estimateGaussian(dataset):
            DEVUELVE mu,sigma

        FUNCION multivariateGaussian (dataset,mu,sigma):
            DEVUELVE p.pdf(dataset)

        FUNCION selectThresholdByCV(probs, gt):

            SI (f-actual < f_nuevo)
                best_recall = Recall
                best_precision = Precision
                best_epsilon = epsilons

        //ESCRIBIR ('F1 score , Recall and Precision se muestran abajo')
        //ESCRIBIR ('Best F1 Score %f' %f)
        //ESCRIBIR ('Best Recall Score %f' %Recall)
        //ESCRIBIR ('Best Precision Score %f' %Precision)
        //ESCRIBIR (% Curva AUC)
        //ESCRIBIR (% Curva ROC)
        //ESCRIBIR (% MATRIZ DE CONFUSION)


    //Division de datos segun anomalias 0(normal) 1(anomalia)
    //Llamadas de las funciones definidas previamente
FIN
```

Ilustración 5. Pseudocódigo Distribución Gaussiana.

Este algoritmo es ejecutado a través de la terminal de Windows 10 y utiliza una librería de Python específica para Machine Learning llamada "Scikit-Learn". Se definen las variables necesarias, dentro de las cuales estarán almacenados los datos capturados (guardados previamente como fichero de tipo "csv").

Dentro del contenido del algoritmo se diferencian tres funciones importantes, que forman la columna vertebral del mismo [31]:

- **estimateGaussian**: averiguar el valor de "mu" y "sigma", variables que serán añadidas a la siguiente función ya que los parámetros utilizados para parametrizar este modelo.
- **multivariateGaussian**: calcula la probabilidad de distribución de cada fila (un ejemplo del conjunto de datos) gracias a los parámetros "mu" y "sigma".
- **selectThresholdByCV**: función más importante del algoritmo, en la cual se establecen dos pasos muy importantes:
    - Se definen los valores de épsilon (valor del umbral a través del cual el algoritmo de Distribución Gaussiana clasifica una transacción como anómala ya que al poseer un valor menor que el límite establecido (épsilon) será automáticamente considerada anómala) y averiguar si está funcionando correctamente el subconjunto de validación cruzada. Esta función está explicada de forma más detallada en el capítulo 8.1.
    - Calcula la **Precision, el Recall y el F1 score** de los datos en función del épsilon que está utilizando en cada momento, y si estos resultados son mejores que los que ya se tenía, estos últimos son sustituidos por los nuevos.

Posteriormente se imprimen por pantalla los resultados obtenidos de los valores anteriormente mencionados junto a las gráficas AUC (área bajo la curva ROC), ROC (muestra el rendimiento de un modelo de clasificación en todos los umbrales de clasificación) y la matriz de confusión (medición de falsos positivos y negativos).

El algoritmo recibirá un conjunto formado tanto por datos anómalos como por datos normales, ambos en el mismo fichero, deberá saber diferenciar si la transacción es anómala o no con un gran porcentaje de acierto. Esto servirá para demostrar que el algoritmo funciona correctamente

a la hora de detectar ciertos ataques de red y que es posible su adaptación a un entorno real y detectar el ataque.

MÉTRICAS

En este capítulo se explican de forma concisa las métricas que sirven para evaluar el modelo y como se utilizan para comprobar que resultado es mejor.

- **Precision:** es el número de resultados positivos correctos dividido por el número de todos los resultados positivos devueltos por el clasificador. Es decir, división entre los verdaderos positivos correctos entre todos los verdaderos positivos (verdadero positivo correcto + falso positivo).
- **Recall:** es el número de resultados positivos correctos dividido por el número de todos los datos relevantes (todos los datos que deberían haberse identificado como positivo).
- **F1 Score**: es el promedio armónico de la Precision y Recall, donde su mejor valor es alcanzado al llegar a 1 (Precision y Recall son perfectos). Para conseguir un buen resultado hay que tener en cuenta ambos valores, ya se puede tener una Precision de 1 (no tienes falsos positivos) pero tener Recall de 0.5 (había 10 verdaderos positivos pero solo has podido detectar 5).

Al ejecutar el algoritmo de Distribución Gaussiana, aparecerán estás tres métricas, una por cada épsilon seleccionado, existe una explicación más detallada en el capítulo 8.1. Para posteriormente poder realizar una comparación de las mismas con el resultado en función del épsilon previamente seleccionado y así poder validar que resultado es mejor.

5.1.2 ISOLATION FOREST

Algoritmo de tipo no supervisado cuyo objetivo es aislar los datos o instancias gracias a la realización de divisiones aleatorias del espacio.

En primer lugar hay que decir que el Isolation Forest identifica explícitamente las anomalías en lugar de perfilar los puntos de datos normales como hacen otro tipo de algoritmos. Se construye sobre la base de árboles de decisión, en dichos árboles las particiones se crean en primer lugar, seleccionando de forma aleatoria una característica para posteriormente escoger un valor de división aleatorio entre el valor mínimo y el máximo de la característica previamente seleccionada.

En principio los datos generados por las anomalías son menos frecuentes que aquellos generados por tráfico normal y son diferentes de ellos en términos de valores, es decir, se encuentran más alejados que los datos no anómalos en el espacio de características. Por esta razón al usar dicha partición aleatoria deben identificarse más cerca de la raíz del árbol (longitud de ruta promedio más corta, es decir, el número de bordes que una observación debe pasar en el árbol que va desde la raíz al nodo terminal), con menos divisiones necesarias [6].

Por lo tanto, se necesitan menos divisiones para aislar un dato anómalo, como se observa en la ilustración 6:

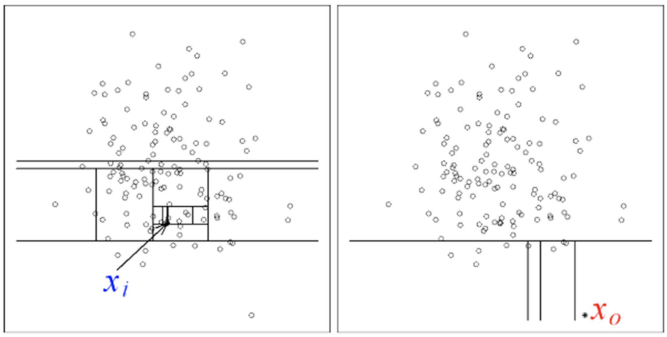


Ilustración 6. Divisiones en espacio por Isolation Forest.

"X0" al ser un dato anómalo solo es necesario realizar tres divisiones para aislarlo mientras que para "X1" necesita más del doble para conseguir aislarla.

PSEUDOCÓDIGO ISOLATION FOREST

El código completo del algoritmo se puede encontrar en el **Anexo A.14**. A continuación se explica el funcionamiento de las secciones más importantes del algoritmo a través de pseudocódigo estándar.

```
ALGORITMO isolation Forest

LIBRERIAS
//Importar las librerias necesarias de Sklearn y Pandas

VAR
X // cargar datos del tráfico almacenados en el csv que servirán de entrenamiento
y // cargar datos del tráfico almacenados en el csv  que serviran de prueba
X_outliers // cargar datos del tráfico maligno

INICIO

    FUNCION train_test-split(x,y)

    FUNCION IsolationForest()
    //Entrenando el Algoritmo con el conjunto de entrenamiento

    FUNCION GridSearchCV
    //Optimizar Hiperparámetros

    //ESCRIBIR (% Mejores_Parametros)

    FUNCION predict
    //predecir nuevos valores

    //ESCRIBIR (% MATRIZ DE CONFUSION)

    //Datos Anómalos
    ESCRIBIR ("Precision:",precision_score(Y_train, train_preds, average = "binary"))
    ESCRIBIR ("Recall:",recall_score(Y_train, train_preds, average = "binary"))
    ESCRIBIR ("F1-Score:",f1_score(Y_train, train_preds, average = "binary"))
FIN
```

Ilustración 7. Pseudocódigo Isolation Forest.

Al igual que sucede con el algoritmo de distribución Gaussiana, la ejecución se produce en la terminal de una computadora Windows 10 utilizándose la librería "Scikit-Learn". Se cargan en tres variables los tres conjuntos de datos que necesita este algoritmo, por un lado los datos que servirán de entrenamiento(x), los datos que servirán de prueba (y) y los datos con carácter maligno (X_outliers).

La columna vertebral del algoritmo está formada por cuatro funciones principales:

- **train_test_split:** función que permite dividir un conjunto en dos bloques, típicamente bloques destinados al entrenamiento y prueba del modelo. La función train_test_split añade al bloque de entrenamiento el 75% de los registros, y al bloque de pruebas el 25%

restante, es posible controlar estas cifras con los parámetros train_size y test_size como se verá en el capítulo 8.2.
    - Entrenamiento: resultado conocido, el modelo aprende con ellos.
    - Prueba: miden la predicción de nuestro modelo en este subconjunto.
- **isolationForest**: función más importante del algoritmo cuyo objetivo es entrenarlo gracias al conjunto de datos de entrenamiento.
- **GridSearchCV**: función que selecciona los hiperparámetros con los cuales se obtiene el resultado más positivo, realizando una comparativa entre los mismos.
- **predict**: función que predice nuevos valores.

Y por último imprime por pantalla las mismas métricas utilizadas para comprobar el correcto funcionamiento del algoritmo de Distribución Gaussiana la **Precision, el Recall y el F1-Score**.

Evaluar la capacidad predictiva de un modelo consiste en comprobar como de próximas son sus predicciones a los valores correctos. Para poder cuantificar de forma correcta el error que puede tener el modelo, es necesario poseer un conjunto de observaciones, de las que se conozca la variable respuesta, pero que el modelo no haya podido comprobar, es decir que no haya participado en el ajuste. Debido a esto, se separan los datos disponibles en tres conjuntos, de entrenamiento, validación y uno de test (prueba):

- Conjunto de entrenamiento, que servirá para entrenar al modelo (ajuste).
- Conjunto de datos de prueba (test), que servirá para probar el modelo previamente entrenado.
- Conjunto de validación, para encontrar los mejores hiperparámetros.

El tamaño adecuado de las particiones depende en gran medida de la cantidad de datos disponibles y la seguridad que se necesite en la estimación del error, 70%-15%-15% suele dar buenos resultados. Como se explicará posteriormente en el capítulo 8.2 si el número de transacciones es limitado generar tres particiones puede generar grupos pequeños y llegar a generar conclusiones contradictorias.

De esta manera, para la realización del proyecto al ser conjuntos con transacciones limitadas, se desechó la opción del conjunto de validación y directamente se dividió el conjunto principal entre los datos en entrenamiento y test, utilizándose la validación cruzada para garantizar que los datos de entrenamiento y prueba son independientes entre sí. La validación cruzada es una técnica sencilla que permite averiguar y garantizar que los conjuntos de entrenamiento y prueba obtenidos de la partición anterior son independientes, es decir, que los datos de prueba no estén repetidos en los de entrenamiento y viceversa. Se calcula a través de la repetición y cálculo de la media aritmética sobre diferentes particiones [50].

El algoritmo también recibirá un tercer conjunto formado por datos anómalos, transacciones generadas y capturadas durante la realización del ataque.

Estos conjuntos de datos deberán cumplir una serie de condiciones:

- Una característica importante del Isolation Forest es que no necesita un conjunto de datos muy extenso para funcionar correctamente.
- El conjunto de datos de prueba debe tener las mismas características que el de entrenamiento.

- No usar los datos de prueba en el conjunto de entrenamiento, ya que puede confundir al algoritmo.
- Los ejemplos anómalos deben distinguirse de los normales.
- Deben existir pocos ejemplos anómalos en relación a los datos normales.

5.2 JUSTIFICACIÓN

Existen una gran cantidad de algoritmos basados en Machine Learning cuyo objetivo entre otros es la detección de anomalías como por ejemplo Redes Bayesianas, Clustering o Redes Neuronales, por lo que fue necesario hacer una selección de los mismos. Básicamente se eligieron los algoritmos de distribución Gaussiana e Isolation Forest por dos razones.

La primera es por la sencillez que suponen ambos, como se comenta en el capítulo de motivación el número de asignaturas que se cursan relacionadas con la seguridad informática son muy reducidas por lo que es complicado entender según qué algoritmos y conceptos sin antes cursar alguna especialidad relacionada con la seguridad. Por esta razón se escogió desde un primer momento algoritmos sencillos, simples y fáciles de entender, que no requirieran un gran conocimiento previo del ciberseguridad o Python. Ambos algoritmos poseen estas cualidades.

La segunda razón es que son algoritmos que detectan anomalías de manera muy diferente. La Distribución Gaussiana es un algoritmo de detección de anomalías basado en aprendizaje semi-supervisado, y basado en densidades, sin embargo Isolation Forest es un algoritmo de aprendizaje no supervisado y basado en árboles.

La tabla 1 realiza una comparativa de las diferencias que se pueden encontrar entre estos dos algoritmos.

DISTRIBUCIÓN GAUSSIANA	ISOLATION FOREST
Semi-supervisado	No supervisado
Basado en densidades	Basado en árboles
Basado procedimientos simétricos	Basado en divisiones irregulares
Basado en estadísticas matemáticas	Utiliza datos de carácter aleatorio
Proporciona estabilidad	No proporciona estabilidad
Mayor lentitud	Mayor rapidez
Menor % de error	Mayor % de error

Tabla 1. Comparativa de algoritmos.

Gracias a esta tabla es posible encontrar grandes diferencias entre los dos algoritmos, las más importantes son:

- *Como aprenden:* si el algoritmo es supervisado, semi-supervisado o no lo es, otorgándole o no etiquetas de las características que poseen los datos.
    - En este caso la distribución Gaussiana utiliza datos de entrenamiento tanto etiquetados como no etiquetados (pequeña cantidad de etiquetados y una gran cantidad de no etiquetados). Normalmente el coste del proceso de etiquetado puede hacer al conjunto de datos inviable por eso se ha descubierto que los datos no etiquetados junto con una pequeña cantidad de etiquetados mejorar considerablemente la exactitud del aprendizaje, reduciendo el costo.
    - Mientras que el Isolation Forest utiliza solo datos de entrenamiento no etiquetados. La máquina se encarga de determinar las correlaciones y relaciones

de dichos datos disponibles, interpretándolos y dirigiéndolos en consecuencia. A medida que evalúa más datos, su capacidad para tomar decisiones sobre los mismos mejora gradualmente y se vuelve más acertada.

- *En que están basados*: la distribución Gaussiana está basado en densidades y estadísticas matemáticas, mientras que el Isolation Forest está basando en árboles y en divisiones de datos con carácter aleatorio.

Ninguno de los dos algoritmos utiliza un aprendizaje supervisado, donde todos los datos están etiquetados, a modo de ejemplo. De esta forma se puede comprobar que los algoritmos aprenden y actúan de formas muy dispares por lo que se pueden adatar de manera diferente al tipo de problema que les planteamos, algo que favorece la diversidad del proyecto.

6. IMPLEMENTACIÓN

6.1 CODIFICACIÓN

Para la realización del trabajo se ha necesitado el uso de algoritmos, los cuales están escritos en lenguaje de Python.

Python: es un lenguaje de programación interpretado cuya filosofía hace en una sintaxis que favorezca el código legible [33]. Es conocido por su variedad de librerías útiles para Machine Learning, la más famosa y conocida de ellas es Scikit-Learn la cual es de código abierto y es reutilizable en varios contextos, fomentando su uso académico y comercial. Dentro de la misma se incluyen varios algoritmos de clasificación, regresión, y análisis de grupos [34].

Esta librería está construida sobre SciPy (Scientific Python) y está formada por las siguientes librerías o paquetes:

- **NumPy**: librería de matriz n-dimensional.
- **Pandas**: estructura de datos y análisis.
- **SciPy:** librería fundamental para la informática científica.
- **Matplotlib**: trazado completo 2D.
- **Ipython:** consola interactiva mejorada.
- **SymPy:** matemática simbólica.

Las usadas principalmente por estos algoritmos son Numpy, Pandas, Scipy y Matplotlib.

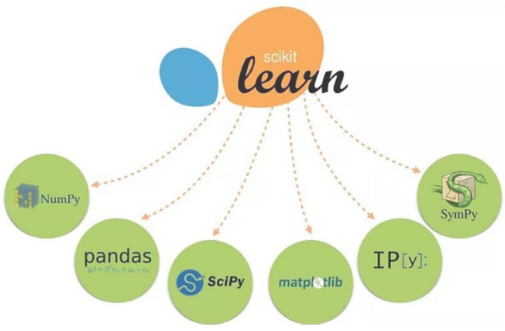


Ilustración 8. Scikit-Learn & Librerías.

Los algoritmos se están ejecutando a través de la terminal de una computadora con Windows 10, con la versión Python 3.

7. GENERACIÓN Y CAPTACIÓN DE LOS DATOS DE ANÁLISIS

Se ha dividido esta sección en tres fases, cada una de ellas con unas características y condiciones diferentes.

La primera de ellas consta de todo el proceso de explicación, mitigación y simulación del ataque de **Fuerza Bruta** así como la captura y optimización de los datos, todo ello en un

entorno virtualizado y controlado junto a la de simulación de tráfico normal; por lo tanto en esta primera fase se obtendrán transacciones de un solo ataque y de tráfico legítimo.

La segunda fase está formada por la definición y algunas pautas de mitigación de los ataques **Http-Flood y TCP-SYN** así como la explicación del conjunto de datos KDD'99 del cual se han extraído todos los datos necesarios de los dos ataques para su posterior optimización además de tráfico legítimo (también obtenido del KDD'99); por lo tanto en esta segunda fase se obtienen transacciones de datos pertenecientes a dos ataques y de tráfico no anómalo.

La tercera fase consta de la explicación del conjunto de datos NSL-KDD del cual se han extraído todos los datos necesarios de los tres ataques **(Fuerza Bruta, Http-Flood y TCP-SYN)** ya explicados en las fases anteriores además de tráfico normal; por lo tanto en esta tercera fase se obtienen transacciones de datos pertenecientes a todos los ataques utilizados en este proyecto además de tráfico legítimo que nos servirá posteriormente para analizar la validez de los algoritmos de detección de ataques.

7.1 PRIMERA FASE: SIMULACIÓN EN INFRAESTRUCTURA VIRTUAL

La primera fase consiste en la captación de los datos de tráfico ilegítimo obtenidos durante la simulación de un **ataque de Fuerza Bruta** realizado en una infraestructura controlada y virtualizada además de datos legítimos como se explicará en el capítulo 7.1.4.

En esta fase se utilizarán algoritmos de detección de anomalías para realizar una primera aproximación y determinar si dichos algoritmos seleccionados funcionan de forma correcta en un entorno virtual y simulado para detectar el ataque previamente realizado. Los resultados serán posteriormente comparados con resultados obtenidos al analizar datos capturados en la segunda fase (KDD'99) y capturados en la tercera fase (NSL-KDD) para comprobar la evolución y eficacia de los algoritmos en diferentes entornos y situaciones.

7.1.1 ATAQUE DE FUERZA BRUTA

El ataque simulado es el mundialmente conocido como Fuerza Bruta o en inglés "Brute Force", cuyo objetivo principal son los sistemas de autentificación donde es necesario introducir un usuario o contraseña para poder acceder a la plataforma o aplicación, algunos de estos sistemas comunes para este tipo de ataques pueden ser perfiles pertenecientes a redes sociales o cuentas de correo electrónico.

Por lo tanto la definición del ataque sería la siguiente, técnica que permite comprobar todas combinaciones posibles hasta encontrar la palabra o texto legible que fue cifrado para obtener el criptograma, lo que coloquialmente es conocido como contraseña.

En un primer lugar, es posible encontrar dos situaciones en las cuales puede estar situado un Hacker, la primera es que posea información acerca de la víctima como por ejemplo el usuario; en esta situación la complicidad se puede ver reducida para un ataque simple de Fuerza Bruta ya que normalmente los usuarios suelen utilizar palabras o fechas cercanas a ellos para acordarse de las contraseñas además de posiblemente utilizar la misma para diferentes aplicaciones.

La otra posibilidad es no poseer ningún tipo de información, en este caso el atacante deberá utilizar una serie de procesos que ayudarán a descifrar la contraseña y el usuario, y sobre todo a

reducir el tiempo que se requiere para hacerlo gracias a las *WordList de usuario* y *WordList de contraseña.*

Existen herramientas que generan este tipo de listas, como por ejemplo "Crunch" [43], dentro de la cual se introducen una serie de parámetros entre los que se pueden encontrar el intervalo de longitud, caracteres, etc. Sirve tanto para listas de usuarios como listas de contraseñas y cuanto mayor sea el tamaño de dichas listas mayor será la probabilidad de encontrar lo que se está buscando [7].

Otra técnica muy relacionada con el ataque de Fuerza Bruta es el llamado **ataque de diccionario,** su objetivo es similar pero en este caso consiste en utilizar todas las palabras del diccionario. Este tipo de ataques suele fallar para contraseñas fuertes, sin embargo, la mayoría de los usuarios usan contraseñas fáciles para poder recordarlas [37].

Al tener que realizar miles de comprobaciones, normalmente es un proceso costoso que puede llegar a durar días aun incluso con la ayuda de herramientas encargadas de realizar las comprobaciones de cada usuario y contraseña almacenadas previamente en las listas.

MÉTODOS DE MITIGACIÓN DEL ATAQUE DE FUERZA BRUTA

Una vez conocido y entendido el funcionamiento del ataque de Fuerza Bruta, a continuación se describen una serie de pautas que se deben y/o pueden tomar para ayudar a prevenir y mitigar este tipo de ataques en caso de ser una víctima de los mismos:

- **Contraseña fuerte**: es una medida bastante popular y recomendada por todos los sitios web donde necesites introducir contraseña para acceder al sitio, normalmente los usuarios no siguen este tipo de recomendaciones por lo que el atacante encuentra de manera fácil y rápida la contraseña. Esta es la primera línea de defensa, y probablemente la manera más eficaz para evitar que un atacante tenga éxito.
- **Restringir acceso a las URLs de autenticación**: una recomendación es restringir el acceso a la página de inicio de sesión solo a las direcciones IP autorizadas. Como por ejemplo con WordPress restringiendo el acceso a /wp-login a través del archivo .htaccess.
- **Cifrado de contraseña**: método universal usado por defecto y de forma automática por todos los accesos que requieran contraseña, dentro de los cifrados es posible encontrar diferentes tipos cada uno con mayor o menor dificultad para el atacante como el SHA-1 o SHA-256 o MD5.

7.1.2 SIMULACIÓN DE ATAQUE

En este capítulo se muestra el proceso seguido en este proyecto para la simulación del ataque de Fuerza Bruta. Como ya se ha explicado en el capítulo 7.1.1, este ataque posee muchas variaciones y se pueden utilizar diferentes herramientas y procesos para lograr el objetivo, en este caso se ha utilizado una máquina virtual llamada Kali-Linux la cual posee por defecto el software Hydra (muy utilizado en tareas de testeo en seguridad infromática).

En el **Anexo A.1** (Kali Linux y Metasploitable) está la explicación de la instalación completa que hay que realizar para configurar de forma correcta las distintas máquinas virtuales necesarias para este ataque.

Dentro de Kali-Linux es posible realizar el ataque a través de dos formas, interactuando con la interfaz que Hydra proporciona o directamente desde la terminal de comandos de Linux (este caso).

Hydra ofrece una serie de parámetros con los cuales poder atacar de la forma elegida, en este caso el método de ataque es **"http-post-form"**, como su propio nombre indica se realiza sobre el puerto http, cuyo correspondiente número es el 80, por lo tanto es necesario que este abierto [8].

Al ser una simulación se conoce la dirección IP de la misma forma que el puerto 80 está abierto. Esta situación es poco probable ya que en primer lugar se desconoce la IP de la víctima, y en segundo lugar el puerto 80 suele estar cerrado debido a la vulnerabilidad que conlleva tenerlo abierto, por lo que normalmente se necesitan una serie de herramientas para descubrirlos.

Una herramienta muy útil que se suele usar para este tipo de procesos es Nmap [44], la cual posee una gran variedad de opciones entre las cuales está el descubrir que direcciones IPs existen en un rango introducido previamente, para posteriormente detectar que puertos están abiertos y en función de eso seleccionar un tipo de variedad dentro del ataque de Fuerza Bruta.

Al conocer la dirección IP y el puerto abierto solo es necesario introducirlos en un buscador para poder acceder a la interfaz de la víctima → **192.168.56.101:80** (**IP** → 192.168.56.101) y (**Puerto** → 80).

El siguiente paso es realizar un "login" desde una computadora con usuario y contraseña erróneos, algo que no permitirá acceder pero será posible capturar con Burp Suite la dirección que posteriormente será utilizada en los comandos de Hydra. Antes de seguir, aclarar que es Burp Suite.

- **Burp Suite** [45]: herramienta utilizada principalmente para las auditorías de seguridad en aplicaciones web, que permite combinar pruebas tanto automáticas como manuales y que dispone de un gran número de funcionalidades.

Al realizarse el "login" erróneo una serie de datos son capturados por Burp Suite entre los que se encuentra el usuario y contraseña utilizados para el "login", el puerto ,la dirección IP y el "Referer", dato necesario para la ejecución de Hydra como puede verse a continuación:

http://192.168.168.10/mutillidae/index.php?page=login.php

El siguiente paso es utilizar Hydra a través de la línea de comandos de Linux que proporciona el sistema operativo [19]. En el **Anexo A.2 (comandos de Linux)** esta explicado con detalle que hace cada uno de los comandos escritos en la terminal de Kali-Linux. Se puede comprobar si el ataque está funcionando de forma correcta gracias al comando "-V" el cual saca por pantalla todos los intentos realizados.

7.1.3 RECOLECCIÓN DE DATOS

A continuación una vez explicado y simulado el ataque, el siguiente paso es capturar el tráfico generado durante la simulación. Para realizar esta tarea se ha requerido una serie de herramientas específicas que realizan una captura de los datos, algo que posteriormente permite

adaptarlos al algoritmo que se han necesitado en cada momento. Las herramientas utilizadas son las siguientes:

- **Tstat (TCP Statistic and Analisys Tool):** [10] es un rastreador pasivo capaz de proporcionar información sobre los patrones de tráfico tanto en la red como en los niveles de transporte. En lo que instalación requiere existieron una serie de problemas para configurarla de forma adecuada ya que en función de la versión de Windows puede variar, una vez solucionados su funcionamiento es a través de comandos específicos en la terminal: **Tstat –i eht0 –l–N net.private**
  En el **Anexo A.3 (Comandos de Tstat)** está explicado con detalle el funcionamiento de cada uno de los comandos usados para la ejecución del software.

- **Nprobe**: [11] es una aplicación software desarrollada por los mismos creadores de Ntop y Netflow, actúa de una forma muy similar a Tstat capturando el tráfico de red en vivo, pero de forma mucho más completa. Su ejecución se realiza a través de comandos en la terminal de Linux: **Nprobe -i eth0.**
  En el **Anexo A.4 (comandos de Nprobe)** está explicado con detalle el funcionamiento de cada uno de los comandos usados para la ejecución del software.

Después de unos minutos de intercepción de tráfico al mismo tiempo que se ejecutaba el ataque se comprobó que los datos obtenidos gracias a Tstat eran mucho más completos debido a la gran cantidad de "features" que reportaban ya que eran aproximadamente 130 mientras que con Nprobe no llegaban a 30 por lo que el estudio sería menos completo, a raíz de estas conclusiones se decidió usar los datos capturados con Tstat.

Una vez capturados los datos de carácter malicioso el siguiente paso es capturar una gran cantidad de tráfico de red generado por un usuario normal, también conocido como ruido (esta cantidad debe ser mayor que la cantidad capturada anómala dentro de unos límites específicos ya que como se explicará más adelante el conjunto de datos debe ser equilibrado).

Se encontraron una serie de opciones para generarlo, desde hacer ping a la víctima, la utilización de un software que permite generar tráfico TCP, UDP o SSL como Packet Sender [46] o establecer una puerta de enlace en la víctima algo que permitiría interceptar todo el tráfico generado por la misma.

El proceso más fiable es la puerta de enlace ya que con él se puede simular tráfico idéntico al que generaría un usuario normal por ejemplo realizar búsquedas en internet, mandar emails o actualizar Windows.

Para esta sección se cambió el S. O. de la víctima ya que desde el usado anteriormente (explicado en el **Anexo A.1**) no era posible realizar búsquedas en internet o mandar emails debido a que es una terminal y sus funcionalidades estaban muy limitadas, por lo que se eligió un sistema operativo Windows 8.1. En el **Anexo A.5** aparece toda la instalación necesaria para la máquina virtual de Windows 8.1 en VirtualBox.

PUERTA DE ENLACE

El objetivo de este apartado es explicar de forma más detallada en que consiste el proceso Puerta de Enlace. Este tipo de conexión [26], que permite interconectar redes con protocolos y

arquitecturas diferentes a todos los niveles de comunicación. Su objetivo es traducir información proveniente del protocolo de inicio usado en una red al protocolo destino.

La puerta de enlace, también conocida como "Default Gateway", es la ruta definida por defecto que tiene asignada un equipo y cuya es función es enviar cualquier paquete del que se desconozca la interfaz al cual enviarlo. Se aprovecha esta ruta para definir en la máquina virtual Windows 8.1 a la máquina de ataque como ruta por defecto a través de una serie de configuraciones que están explicadas con detalle en el **Anexo A.6**. De esta forma es posible capturar todo el tráfico de red que la víctima está generando realizando al gracias a operaciones cotidianas.

El proceso de captura es igual al anteriormente realizado con el comando Tstat para capturar el tráfico y solamente hay que realizar búsquedas por internet, mandar algún correo o realizar actualizaciones en Windows con la máquina cliente, esto generará una gran cantidad de tráfico no anómalo el cual será capturado por la herramienta Tstat.

7.1.4 OPTIMIZACIÓN DE LOS DATOS CAPTURADOS

En este apartado se explican una serie de serie modificaciones y optimizaciones a los datos capturados previamente que mejorarán el resultado final del algoritmo.

Una de las características más destacables de Tstat es su capacidad para almacenar datos en función de su origen y protocolo, pudiendo distinguir entre datos de tipo TCP, UDP o incluso multimedia, gracias a esto se dedujo que al realizar el ataque un gran porcentaje datos era tipo TCP (más de 90%). Otra de las facilidades que posibilita seguir este proceso es la no necesaria adaptación de los datos maliciosos y normales debido al uso de la misma herramienta para su captura, en caso contrario si se hubieran elegido dos herramientas diferentes esta adaptación habría sido obligatoria, comparado características e incluso añadir algunas nuevas.

En primer lugar al ser un algoritmo semi-supervisado solo recibe de la etiquetación de una parte de los datos, en este caso de los datos legítimos, los cuales gracias al añadido de la "feature" "Class" son clasificados con el número "0". Posteriormente algoritmo calculará un modelo que modeliza los datos etiquetados (normales) obteniendo un umbral, de esta forma cuando analice un dato malicioso los resultados obtenidos de dicho análisis se encontrarán en un extremo de la distribución (ilustración 4), donde menos porcentaje de datos normales exista y por debajo del umbral establecido anteriormente.

Una vez clasificadas las transacciones hay que diferenciar que características son útiles para las distinción de los datos y cuales no; los datos poseen 130 "features" de las cuales muchas no poseen distinción entre los valores anómalos y normales, por lo tanto el algoritmo no podrá realizar la detección de anomalías de la mejor forma, como consecuencia todas estas características deben ser eliminadas.

Esta distinción de características se realiza a través de un pequeño algoritmo que posee el mismo funcionamiento que los anteriores explicados con pseudocódigo, donde los datos son almacenados a través de un "csv" en un variable. El código del algoritmo se puede encontrar en el **Anexo A.7**, donde está escrito el código y explicado con todo detalle el funcionamiento del mismo.

La ejecución del algoritmo visualiza 130 gráficas, una por cada una de las características de los datos capturados, que a su vez posee dos trazas internas, una para los datos con el valor de "Class" 0 (normales) cuyo color es azul y otra para el valor 1 (anómalo) cuyo color es naranja. En el **Anexo A.9** está cada una de las gráficas obtenidas, además de una explicación de la función cuyo objetivo es eliminar las características sobrantes.

Existe un algoritmo capaz de indicar cuales son las características más importantes y cuales desechar. Posee una función característica que permite clasificar todas las variables según su importancia (la explicación completa del algoritmo se encuentra en el **Anexo A.8**). Sin embargo, algunos datos superaban el límite de caracteres que este algoritmo podía soportar (float32), algo que hizo imposible la utilización del mismo, por lo que la única opción restante era realizar la selección de forma manual. En el **Anexo A.9** se encuentran explicadas las características más importantes, con una definición de cada una y en qué consisten.

7.2 SEGUNDA FASE: CONJUNTO DE DATOS DE KDD'99

La segunda fase consiste en la captura de transacciones ilegítimas pertenecientes a un conjunto de datos sencillo, llamado KDD-99, cuyo año de captación fue en 1999. En esta fase de captación de datos no se han realizado los ataques en un entorno virtualizado y controlado ni ninguna simulación de los mismos, sino que se han obtenido los datos del conjunto KDD'99 seleccionando específicamente las transacciones pertenecientes a dichos ataques.

Los ataques seleccionados para esta sección son **Http-Flood y TCP-SYN** que al ser de tipo DoS como se explica a continuación poseen una gran diferencia de comportamiento con respecto al ataque de Fuerza Bruta**.** Al evaluar los resultados obtenidos del análisis de datos perteneciente a esta fase se debe poder comprobar si los algoritmos escogidos funcionan de forma eficaz con estas condiciones específicas y poder deducir si se obtiene mejor o peor resultado al comparar con resultados obtenidos del análisis de datos de otras fases.

7.2.1 ATAQUES HTTP-FLOOD Y TCP-SYN

En este apartado por un lado se explica el origen común que poseen los dos ataques mencionados anteriormente ya que ambos forman parte de un grupo llamado de *negación de servicio* por lo que comparten muchas características, por otro lado también se realizará una explicación individual de cada uno.

¿QUÉ SON LOS ATAQUES DE NEGACIÓN DE SERVICIO?

También conocidos como ataque DoS [21] (debido a sus siglas en inglés, Denial Of Service), van dirigidos a sistemas de computadoras o de red causando la inaccesibilidad del servicio a recurso a usuarios legítimos. Normalmente al atacar la red de la víctima, esta consume una gran cantidad de ancho de banda, algo que provoca la pérdida de la conectividad de la red o sobrecarga de los recursos computacionales del sistema atacado.

Este tipo de ataques se generan mediante la saturación de los puertos con múltiples flujos de información, provocando una sobrecarga en el servidor impidiendo que siga prestando servicio, debido a esto se le denomina *denegación,* provocado por demasiadas solicitudes para el servidor.

Existe una ampliación de este tipo de ataques llamado denegación de servicio distribuido (por sus siglas en inglés, Distributed Denial of Service (DDoS)), se lleva a cabo generando un enorme flujo de información al igual que los ataques DoS, pero en este caso desde varios puntos de conexión a un mismo punto de destino. Uno de los métodos más eficaces para realizar esta ampliación con gran éxito es el uso de Bots, los cuales actúan de forma concurrente y muy eficaz.

HTTP-FLOOD

En este apartado se explica de forma más detallada el funcionamiento del ataque Http-Flood, primer ataque seleccionado del conjunto de datos KDD'99, cuya forma de saturar el tráfico es a través de solicitudes de HTTP [20]. Este tipo de ataques, se encuentra dentro de la capa 7, la cual hace referencia a la capa de aplicación del modelo OSI referida a los protocolos de internet como el HTTP. Es posible encontrar dos variaciones:

- **Ataque HTTP GET:** coordinación de un gran número de computadoras que envían de forma simultanea múltiples solicitudes de imágenes, archivos o algún otro activo desde un servidor específico, provocando la negación de servicio. Técnica más efectiva con un escenario de Bots.
- **Ataque HTTP POST:** normalmente, cuando se produce un envío de un formulario en un sitio web el servidor debe procesar la solicitud que le ha llegado para posteriormente enviar los datos, normalmente a una base de datos. El proceso nombrado anteriormente es más intenso que la cantidad de potencia necesaria para procesar el ancho de banda requerido para una solicitud POST.

TCP-SYN

En este apartado se explica de forma más detallada el funcionamiento del ataque TCP-SYN, segundo ataque seleccionado del conjunto de datos KDD'99. Para que se realice este ataque es necesario que se establezca una conexión TCP/IP entre dos máquinas, enviándose una serie de datos junto a una petición real [22].

Estos datos conforman la cabecera de la solicitud, dentro de la cual se encuentran las flags (banderas) encargadas de iniciar una conexión, cerrarla, reiniciar…etc., Están incluidas tanto en la respuesta como en la solicitud. El proceso es el siguiente:

- El cliente establece una conexión con un servidor enviando una solicitud SYN.
- El servidor responde con un paquete SYN/ACK que es validado por el paquete ACK (reconocimiento).

Tienen que haber finalizado estas vías para que se establezca la conexión TCP, una vez finalizado ese proceso entra en juego este tipo de ataque ya que envía una gran cantidad de solicitudes a cada puerto del servidor desde

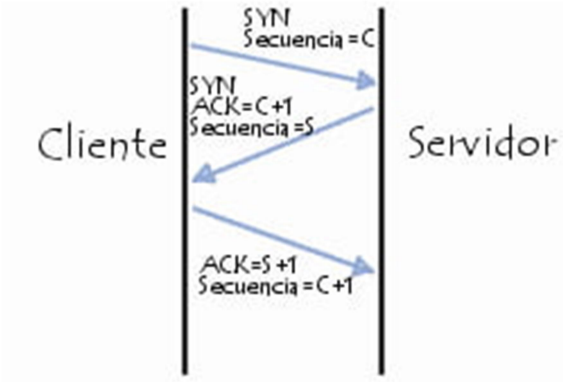


Ilustración 9. Proceso TCP-SYN

uno o varios ordenadores que poseen IPs falsas o inexistentes por lo que el equipo destino no puede recibir el paquete ACK quedando el canal en una estructura de memoria de datos esperando la recepción del paquete ACK.

Aquellos puertos que estén cerrados contestarán con un paquete RST (reset). Antes de que pueda producirse un tiempo de espera otro paquete SYN llega desde el cliente hostil, una conexión de este tipo se denomina conexión medio abierta. En estas condiciones, el servidor se vuelve completamente o casi completamente ocupado con el cliente hostil por lo tanto la comunicación con los clientes legítimos es difícil o imposible.

MITIGACIÓN DE LOS ATAQUES DE TIPO DOS

En este apartado se encuentran algunas medidas que se pueden usar para evitar ataques de este tipo en caso de poder ser víctimas de los mismos. Ambos ataques pertenecen al grupo DoS (ataques de negación de servicio) por lo que la mayoría de las medidas propuestas servirán como protección para ambos ataques, excepto en alguna situación específica. Se puede diferenciar tres entornos donde poder añadir protección contra este tipo de ataques [28]:

- **Es recomendable disponer de gran cantidad de ancho de banda**, tanto por parte de nuestro sistema como por parte del proveedor de servicios. De esta manera sería más difícil que se realizara la saturación que ataques de este tipo quieren conseguir.
- **Uso de CDN (Content Delivery Network),** una red formada por servidores alejados geográficamente los cuales son copias exactas de los mismos, de esta manera se pueden ofrecer respuestas más rápidas a las peticiones web, algo que aumenta la capacidad cache de nuestra red y descongestionar el servicio.
- La **desactivación de todos los puertos** que no sean necesarios en los servidores. Si el objetivo es alojar un servicio web deben estar abiertos el puerto 80 para peticiones HTTP o el puerto 443 para peticiones HTTPs, si el servicio es DNS el puerto abierto debería ser el 53(TCP/UDP).
- En el caso de servidores web, es muy recomendable la **instalación de un WAF** (Web Application Firewall), especializado en controlar todas las conexiones de un sitio web, filtrarlas, bloquearlas, etc. Es posible encontrarlos de tipo “hardware” o “software” y pueden ser instalados en nuestro servidor, integrado en la red o en servicio remoto (“cloud”).

7.2.2 RECOLECCIÓN DE DATOS (KDD’99)

En este capítulo se explica el conjunto de datos del cual se han extraído los datos para los posteriores análisis de los mismos. A diferencia de la anterior fase, las transacciones obtenidas en esta pertenecen a dos ataques (Http-Flood y TCP-SYN) y se han obtenido directamente del conjunto de datos KDD’99 que será explicado a continuación, en lugar del ataque de Fuerza Bruta obtenido a través de una simulación.

Durante las últimas décadas la detección de anomalías se ha convertido en un objetivo primordial para los investigadores que intentan mejorar los sistemas IDS (sistemas de detección de intrusos) provocando el descubrimiento de nuevos ataques. EL conjunto de datos KDDCUP’99 [23] fue uno de los conjuntos de datos más utilizados para esta tarea.

Este conjunto está construido basado en los datos capturados gracias a Tcpdump (herramienta similar a Tstat), en el programa DARPA'98 el cual está compuesto por 4 gigabytes de datos binarios sin procesar pertenecientes a siete semanas de tráfico de red que aproximadamente equivalen a 5 millones de conexiones, cada una de ellas posee una equivalencia de 100 bytes.

Los datos de prueba consisten en 2 millones de registros de conexión durante dos semanas., mientras que los datos de entrenamiento consisten en aproximadamente 4.900.000 millones de vectores de conexión cada uno de los cuales posee 41 características.

Los ataques simulados están clasificados en las siguientes categorías [23]:

- **Ataque de denegación de servicio (Dos):** ataque que satura un tipo de servicio a través del envío de una gran cantidad de solicitudes falsas, impidiendo manejar aquellas solicitudes que sí son legítimas.
- **User to Root Attack (U2R):** el atacante consigue el acceso a una cuenta de usuario a través de ataques secundarios (sniffing o Fuerza Bruta) y es capaz de explotar alguna vulnerabilidad interna para conseguir ser root.
- **Ataque remoto a local (R2L):** proceso que consiste en el envío de paquetes desde el atacante a una máquina a través de una red, intentando explotar alguna vulnerabilidad para tener acceso como usuario a dicha máquina ya que previamente no lo tenía.
- **Probing Attack**: es un intento de recopilar información sobre una red de computadoras para el propósito de eludir controles de seguridad.

Estos conjuntos están formados por un total de 24 tipos de ataques de entrenamiento, con 14 tipos adicionales en el conjunto de datos de prueba. Se puede encontrar en [36], un listado de todos los ataques que componen este conjunto de datos además de una descripción de tallada de los mismos.

Las características de KDD'99 están clasificadas en tres grupos: básicas (TCP/IP), tráfico (intervalo de ventana diferenciándose dos tipos: mismo host y mismo servicio) y contenido (específicas para los ataques R2L y U2R).

7.2.3 PROBLEMAS EN EL KDD'99

Antes de pasar a la última fase del proyecto hay que destacar una serie de problemas que con el paso del tiempo han sido encontrados dentro del conjunto de datos KDD-99 [23]. Esta serie de errores fueron el causante de la pérdida de fiabilidad que poseía el KDD-99 y la necesidad de generar un nuevo conjunto de datos que lo sustituyera.

Como consecuencia, en lugar de generar un conjunto de datos totalmente nuevo, se mejoró el actual surgiendo así el conjunto de datos conocido como NSL-KDD'99, el cual ya no posee los errores de su antecesor y es mucho más fiable.Es por esta serie de razones por las que el proyecto ha ido avanzando en un grado de dificultad donde se empezó con un entorno virtualizado para acabar con un conjunto de datos totalmente fiable.

Como se menciona en el capítulo 7.2.2, KDD'99 está basado en los datos capturados en DARPA'98, la cual fue criticada por las características de datos sintéticos. Como resultado algunos de estos problemas fueron heredados por KDD'99. A continuación se explica los problemas que poseía DARPA'98 y la posible existencia de los mismos en el actual conjunto de datos de KDD'99 [23].

- Se detectó que estos datos no parecían ser similares a los datos capturados en tráfico en redes normales.
- Recolectores de datos como TCPdump pueden sufrir sobrecargas y pérdidas de paquetes cuando el tráfico es muy pesado. Por lo que existe la posibilidad que esto ocurriera aunque no hay evidencia de ello.
- No hay una definición exacta de los ataques, se puede dar la situación de que un paquete que causa sobrecarga no siempre sea perteneciente a un ataque por lo que puede llegar a confundir, además, debería existir un acuerdo de definiciones de los ataques usados durante la captura de los datos.
- En [23] son mencionadas cinco tipos de anomalías que posteriormente serían usadas para la detección de ataques. Sin embargo un estudio reveló que muchos de los ataques no encajaban en ninguna de las categorías propuestas. Afortunadamente este problema no afecto al conjunto de datos KDD ya que sus 41 características no están relacionadas con ninguna de las debilidades mencionadas.

Sin embargo, KDD posee una serie de problemas adicionales que no existían anteriormente en el conjunto de datos DARPA. Un estudio dividió el conjunto de datos de KDD en 10 porciones cada una de ellas con 490000 transacciones aproximadamente, posteriormente observaron que la distribución de los ataques no era la correcta algo que impedía la realización de la validación cruzada.

- Por ejemplo se encontró que las porciones 5th o 6th poseían solo ataques de tipo "Smurf".
- Otro ejemplo es que el 71% de los datos del conjunto de entrenamiento estaba formado por ataques de tipo DoS algo que afecta completamente a la evaluación.

A continuación se muestran una serie de pruebas o experimentos que verifican los errores comentados anteriormente, destacando la redundancia y el nivel de dificultad del conjunto.

REDUNDANCIA DEL KDD'99

Una de las mayores deficiencias que tiene este conjunto de datos es la gran cantidad de registros redundantes que posee, algo que favorece el enfoque hacia los datos frecuentes y no prestar atención a aquellos que no lo son (Ataques como U2R y R2L ya que poseen menos transacciones de datos). Para solucionar este problema se deben eliminar todos los registros repetidos, las siguientes tablas ilustran las estadísticas de la reducción de registros repetidos en los conjuntos de prueba y entrenamiento.

La **tabla 2** equivale a las estadísticas que se poseen de datos redundantes en el conjunto de datos de entrenamiento del KDD, donde se puede comprobar que la reducción de los datos de ataque es del 93.32%, esto quiere decir que 93 de cada 100 datos pertenecientes a este conjunto estaban repetidos.

	Original Records	Distinct Records	Reduction Rate
Attacks	3,925,650	262,178	93.32%
Normal	972,781	812,814	16.44%
Total	4,898,431	1,074,992	78.05%

Tabla 2. Estadística del conjunto de datos de entrenamiento.

La **tabla 3** equivale a las estadísticas que se poseen de los datos redundantes en el conjunto de datos de prueba del KDD, donde se puede comprobar que la reducción de los datos de ataque es del 88.26%, muy parecida al anterior.

	Original Records	Distinct Records	Reduction Rate
Attacks	250,436	29,378	88.26%
Normal	60,591	47,911	20.92%
Total	311,027	77,289	75.15%

Tabla 3. Estadística del conjunto de datos de entrenamiento.

NIVEL DE DIFICULTAD DEL KDD'99

El proceso que sigue para realizar una detección de anomalías consiste entrenar a un algoritmo para que comprenda y diferencie el comportamiento entre actividades normales y maliciosas. A continuación se muestran una serie de ejemplos donde se puede comprobar la facilidad que tenían algunos algoritmos para detectar dichas anomalías en este conjunto.

Para comprobar la dificultad de detección de dichos ataques se dividió el conjunto en tres secciones iguales y se seleccionó siete tipos de algoritmos entre los que se encuentran árbol de decisiones, redes bayesianas, NBtree, Random Tree, etc. Cada uno de los siete métodos se encargó analizar las tres secciones del conjunto de datos, obteniendo de esta forma 21 resultados.

En las siguientes estadísticas se puede observar los resultados obtenidos para cada una de las 5 millones de transacciones de datos, de las cuales cerca del 96 por ciento fueron clasificadas por todas las pruebas que se realizaron (7 algoritmos * 3 divisiones de conjuntos de datos), esto quiere decir que solo el 4 % de las transacciones no fue clasificada de forma correcta por todas las pruebas.

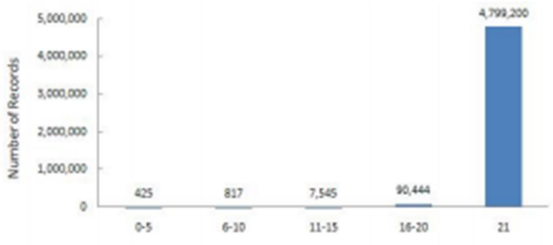


Ilustración 10. Resultados al analizar las transacciones.

Posteriormente se seleccionaron al azar 1.074.992 transacciones de datos del conjunto de entrenamiento, realizándose exactamente el mismo proceso anterior obteniéndose el resultado de la ilustración 11.

	Distinct Records	Percentage	Selected Records
0-5	407	0.04	407
6-10	768	0.07	767
11-15	6,525	0.61	6,485
16-20	58,995	5.49	55,757
21	1,008,297	93.80	62,557
Total	1,074,992	100.00	125,973

Ilustración 11. Resultado análisis datos de entrenamiento.

Como se puede comprobar más del 90 por ciento de los datos pertenecientes al conjunto de datos de entrenamiento fueron capturados y clasificados de forma correcta por los métodos de captación de anomalías, por lo que se puede deducir que la dificultad que aporta este conjunto de datos es muy baja ya que resulta bastante sencillo determinar si es anomalía o no.

Como pequeña conclusión a esta sección, se deduce que el conjunto de datos no era fiable debido a la repetición de los datos y su mala distribución y variación. Por este motivo se desarrolló un nuevo conjunto en el cual se solucionan los problemas anteriores y aumenta la diversidad del mismo.

7.1.4 OPTIMIZACIÓN DE LOS DATOS CAPTURADOS

Al igual que se realizó para el conjunto de datos en un entorno virtualizado, hay que realizar una optimización de este último conjunto, el proceso es el mismo seguido en el caso anterior y la explicación total y detallada del algoritmo se encuentra en el **Anexo A.7**. En este caso hay que hacer la optimización para el conjunto perteneciente a **Http-Flood** y a **TCP-SYN.** En los **Anexos A.10 y A.11** están explicados los procesos seguidos para cada uno de estos dos ataques respectivamente.

7.3 TERCERA FASE: CONJUNTO DE DATOS REAL (NSL-KDD)

Durante la tercera fase se recolectan los datos de los tres ataques (Fuerza Bruta, Http-Flood y TCP-SYN) a través de un conjunto de datos llamado NSL-KDD que como se verá en la posterior explicación es una mejora del utilizado en la segunda fase (KDD'99).

Este conjunto de datos posee un carácter mucho más realista que el utilizado en las dos anteriores fases, con un número de ataques más elevado y una mayor cantidad de tráfico, según análisis realizados para este conjunto con otro tipo de algoritmos [24], los resultados han de ser muy positivos por lo que se usarán de referencia para las otras dos fases.

Una vez realizados todos experimentos se deberá poder comprobar la eficacia de los dos algoritmos en las tres situaciones propuestas: entorno virtualizado con poco tráfico, entorno controlado y simple pero real y finalmente con un entorno con mucho tráfico y totalmente real.

7.3.1 RECOLECCIÓN DE DATOS

Como ya se comentó anteriormente este conjunto de datos es una mejora del KDD'99 utilizado en la evaluación anterior por lo que se podría decir que poseen el mismo origen. Este conjunto

proporciona un análisis de varias técnicas de aprendizaje automático para la detección de intrusiones, y una serie de ventajas con respecto a su antecesor, el KDD'99 [25]:

- No incluye registros redundantes en el conjunto de entrenamiento, por lo que los clasificadores no tenderán a analizar los registros más frecuentes.
- No hay registros duplicados en los conjuntos de prueba propuestos; por lo tanto, el rendimiento de los estudios no está sesgado por los métodos que tienen mejores tasas de detección en los registros frecuentes.
- El número de registros seleccionados de cada grupo de nivel de dificultad es inversamente proporcional al porcentaje de registros en el conjunto de datos KDD original. Como resultado, las tasas de clasificación de los distintos métodos de aprendizaje automático varían en un rango más amplio, lo que hace que sea más eficiente tener una evaluación precisa de diferentes técnicas de aprendizaje.
- El número de registros en el entrenamiento y los conjuntos de pruebas es razonable, lo que hace que sea asequible ejecutar los experimentos en el conjunto completo sin la necesidad de seleccionar al azar una pequeña parte. En consecuencia, los resultados de la evaluación de diferentes trabajos de investigación serán consistentes y comparables.

El conjunto [24] de entrenamiento está formado por 21 tipos de ataques diferentes respecto a los 37 presentes en el conjunto de datos de prueba. Aquellos que son conocidos están situados en el conjunto de datos de entrenamiento, mientras que los ataques nuevos son adicionales en el conjunto de datos de prueba, es decir, no están disponibles en los conjuntos de datos de entrenamiento. En la siguiente tabla es posible comprobar los ataques pertenecientes a cada una de las categorías.

Attacks in Dataset	Attack Type (37)
DOS	Back,Land,Neptune,Pod,Smurf, Teardrop,Mailbomb,Processtable,Udpstor m,Apache2,Worm
Probe	Satan,IPsweep,Nmap,Portsweep,Mscan,Sa int
R2L	Guess_password,Ftp_write,Imap,Phf,Multi hop,Warezmaster,Xlock,Xsnoop,Snmpgue ss,Snmpgetattack,Httptunnel,Sendmail, Named
U2R	Buffer_overflow,Loadmodule,Rootkit,Perl ,Sqlattack,Xterm,Ps

Tabla 4. Tipos de ataques en el conjunto NSL-KDD.

En las siguientes figuras se puede comprobar con detalle el número de registros de cada uno de los ataques, donde se puede verificar que los ataques DOS poseen un mayor número de transacciones en comparación con los demás ataques.

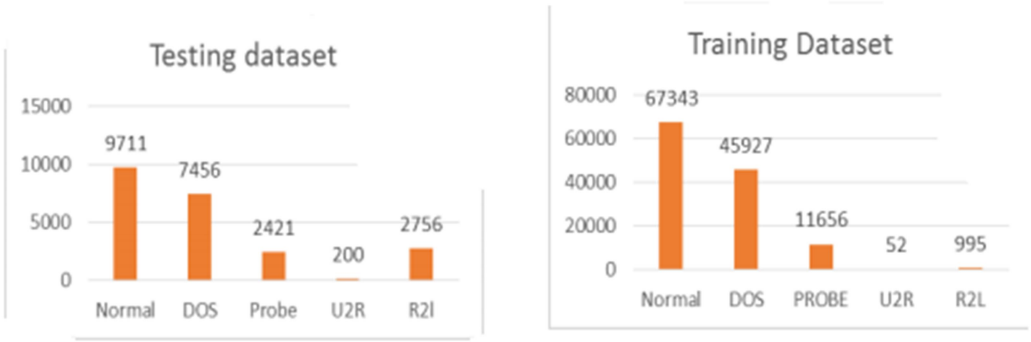


Ilustración 12. Número de ataques en conjuntos de prueba y entrenamiento.

Se puede comprobar que el conjunto de datos NSL-KDD resuelve algunos de los problemas encontrados en los datos de KDD'99, es un conjunto ideal para comparar diferentes modelos de detección de intrusiones.

7.3.2 OPTIMIZACIÓN DE LOS DATOS

Al igual que se ha realizado para las dos fases anteriores hay que optimizar las características, en este caso de los tres conjuntos de datos ya que del NSL-KDD se obtuvo un conjunto por cada ataque. Hay que seguir el mismo proceso utilizado en los algoritmos anteriores eliminando una serie de características que pueden perjudicar nuestro resultado, en el **Anexo A.12** se muestran las imágenes con las gráficas correspondientes junto con el resultado final y una breve explicación del proceso seguido para cada uno de los tres conjuntos pertenecientes a la fase tres.

8. TESTEO CON ALGORITMOS

Durante este capítulo se mostrarán todos los resultados obtenidos al analizar los diferentes conjuntos de datos previamente capturados en la fase de generación y captación. Estará dividida en dos secciones, una para cada algoritmo (Distribución Gaussiana e Isolation Forest) que a su vez estarán compuestas por tres evaluaciones, una para cada situación y/o entorno propuesto anteriormente (entorno virtualizado Fuerza Bruta), KDD'99 (Http-Flood y TCP-SYN) y NSL-KDD (todos los ataques).

8.1 ANÁLISIS CON ALGORITMO DE DISTRIBUCIÓN GAUSSIANA

Como se explica en el capítulo 5.1.1 este algoritmo utiliza una función llamada "selectThresholdByCV" a través de cual se comprueba que valores de épsilon funcionan mejor para cada conjunto de datos. El valor de los épsilon es utilizado como umbral para comprobar si la transacción es considerada anómala o no, si el valor de la transacción está por debajo del umbral seleccionado será clasificada como ilegítima.

Para obtener un conjunto correcto de épsilon se ejecutó este algoritmo con al menos 100 valores de épsilon y así comprobar cuál de ellos funcionaba de mejor manera con cada conjunto de datos, para finalmente realizar una selección de tres épsilon finales con los cuales posteriormente se realizará en análisis en los capítulos siguientes.

Inicialmente los 100 épsilon eran comunes para todos los conjuntos de datos a analizar con este algoritmo aunque posteriormente cada conjunto puede tener tres épsilon específicos sin

embargo como se verá a continuación algunos épsilon son utilizados por más de un conjunto. Se mostrarán los análisis producidos por este algoritmo en cada uno de los tres entornos explicados anteriormente, en el orden siguiente: primera, segunda y tercera evaluación.

Dentro del análisis individual de cada ataque seleccionado se mostrará capturas de las gráficas producidas de los siguientes valores, en el siguiente orden: **Precision, Recall y F1-Score** siguiendo con una selección del épsilon que mejor detecta anomalías para cada ataque dependiendo de la situación que se pueda dar.

8.1.1 PRIMERA EVALUACIÓN: ENTORNO VIRTUAL (FUERZA BRUTA)

Para el ataque de Fuerza Bruta en un entorno virtualizado (primera fase) los épsilon seleccionados son los siguientes: **1.0527717316e-72, 1.0527717316e-76, 1.0527717316e-15.** El objetivo de esta sección es determinar cuál de los tres épsilon obtiene un mejor resultado a la hora de detectar el ataque como se muestra en la tabla 5.

Epsilon	1.0527717316e-72	1.0527717316e-76	1.0527717316e-15
Recall	0.898305	0.864407	0.932203
Precision	0.883333	0.879310	0.705128
F1-Score	0.890756	0.871795	0.802920

Tabla 5. Resultados obtenidos con Distribución Gaussiana vs Fuerza Bruta.

El resultado más importante para verificar que épsilon funciona mejor es el F1-Score (promedio armónico entre el Recall y la Precision), lo que finalmente se quiere comparar con los demás.

De la tabla anterior de deduce que el épsilon que mejor F1-Score obtiene contra un ataque de Fuerza Bruta generado en un entorno virtualizado es el **1.0527717316e-72,** con un porcentaje de acierto del **89.0756**, esto quiere decir que dentro del total de anomalías que detecta, un 89 por ciento pertenece al ataque de Fuerza Bruta, en otras palabras solo 11 de cada 100 datos son considerados falsos positivos, transacciones legitimas clasificadas como anómalas.

Las ilustración 13 muestra una comparativa de la Precision y Recall respectivamente, la primera muestra la Precision obtenida en cada uno de los tres épsilon seleccionados donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-72**, mientras que la segunda hace lo mismo para el Recall donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-15**.

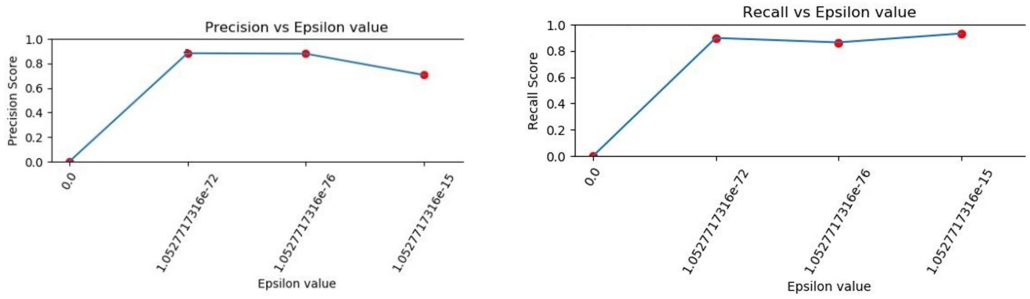


Ilustración 23. Gráficas de la Precisión y Recall, Distribución Gaussiana VS ataque de Fuerza Bruta.

La ilustración 14 muestra el F1-Score obtenido donde el mejor resultado corresponde al épsilon **1.0527717316e-72**.

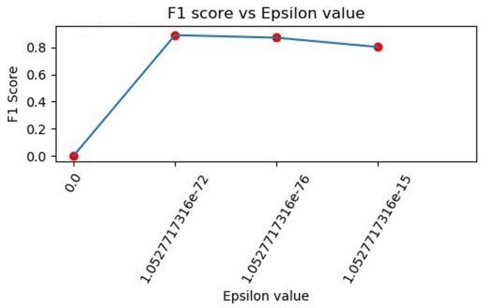


Ilustración 14. Gráfica del F1-Score, Distribución Gaussiana vs ataque de Fuerza Bruta.

8.1.2 SEGUNDA EVALUACIÓN: KDD'99

Esta segunda evaluación corresponde al análisis de los conjuntos de datos capturados durante la segunda fase (Http-Flood y TCP-SYN) extraídos del KDD'99 con el algoritmo de Distribución Gaussiana.

HTTP-FLOOD

Para el conjunto de datos Http-Flood pertenecientes al KDD'99 los épsilon seleccionados son los siguientes: **1.0527717316e-70, 1.0527717316e-50, 1.0527717316e-24.** El objetivo de esta sección es determinar cuál de los tres épsilon obtiene un mejor resultado a la hora de detectar el ataque como se muestra en la tabla 6.

Epsilon	1.0527717316e-70	1.0527717316e-50	1.0527717316e-24
Recall	0.805945	0.805000	0.806305
Precision	0.808889	0.786704	0.784530
F1-Score	0.802504	0.791166	0.799832

Tabla 6. Resultados obtenidos con Distribución Gaussiana vs Http-Flood (KDD'99).

El resultado más importante para verificar que épsilon funciona mejor es el F1-Score (promedio armónico entre el Recall y la Precision), lo que finalmente queremos comparar con los demás.

De la tabla anterior de deduce que el épsilon que mejor F1-Score obtiene contra el ataque Http-Flood extraído del KDD'99 es el **1.0527717316e-70,** con un porcentaje de acierto del **80.2504**, esto quiere decir que dentro del total de anomalías que detecta, un 80 por ciento pertenece al ataque de Http-Flood, en otras palabras, el algoritmo de cada 100 transacciones detectadas como anómalas 20 son transacciones legítimas por lo tanto son falsos positivos.

La ilustración 15 muestra una comparativa de la Precision y Recall respectivamente, la primera gráfica muestra la Precision obtenida en cada uno de los tres épsilon seleccionados donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-70**, mientras que la segunda hace lo mismo del Recall donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-24**.

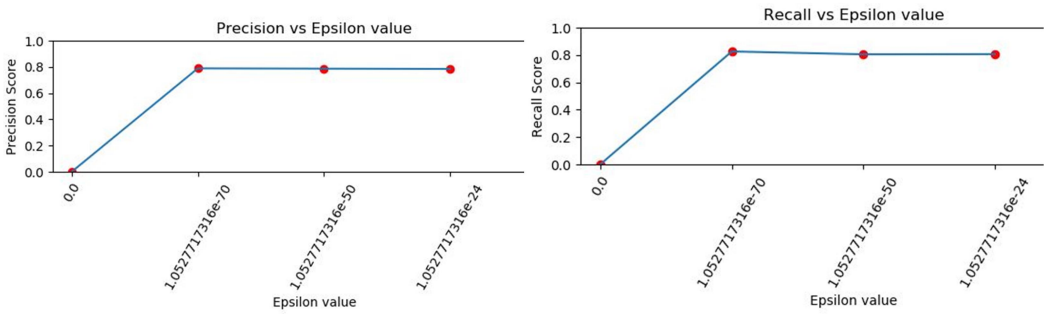


Ilustración 15. Gráficas de la Precisión y Recall, Distribución Gaussiana VS ataque Http-Flood (KDD'99).

En este caso la línea perteneciente a la ilustración 15 que une los resultados es prácticamente horizontal por lo que los resultados obtenidos con cada épsilon en referencia a la Precision y al Recall han sido muy similares.

La ilustración 16 muestra el F1-Score donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-70**.

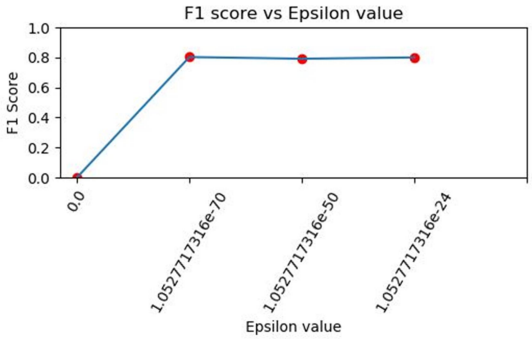


Ilustración 16. Gráficas de F1-Score, Distribución Gaussiana VS ataque de Http-Flood (KDD'99).

TCP-SYN

Para el conjunto de datos TCP-SYN pertenecientes al KDD'99 los épsilon seleccionados son los siguientes: **1.0527717316e-70, 1.0527717316e-76, 1.0527717316e-15,** hay que determinar que épsilon obtiene un mejor resultado a la hora de detectar el ataque.

Epsilon	1.0527717316e-70	1.0527717316e-76	1.0527717316e-15
Recall	0.850001	0.845643	0.823559
Precision	0.836734	0.842944	0.809090
F1-Score	0.841123	0.842678	0.818912

Tabla 7. Resultados obtenidos con Distribución Gaussiana vs TCP-SYN (KDD'99).

El resultado más importante para verificar que épsilon funciona mejor es el F1-Score (promedio armónico entre el Recall y la Precision), lo que finalmente queremos comparar con los demás.

De la tabla anterior de deduce que el épsilon que mejor F1-Score obtiene contra un ataque de TCP-SYN extraído del KDD'99 es el **1.0527717316e-76,** con un porcentaje de acierto del **84.2678**, esto quiere decir que dentro del total de anomalías que detecta, un 84 por ciento

pertenece al ataque de TCP-SYN, en otras palabras 16 de cada 100 datos son considerados falsos positivos ya que al conjunto de transacciones legítimas en lugar de transacciones maliciosas.

La ilustración 17 muestra una comparativa de la Precision y Recall respectivamente, la primera gráfica muestra la Precision obtenida en cada uno de los tres épsilon seleccionados donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-76**, mientras que la segunda hace lo mismo del Recall donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-70**.

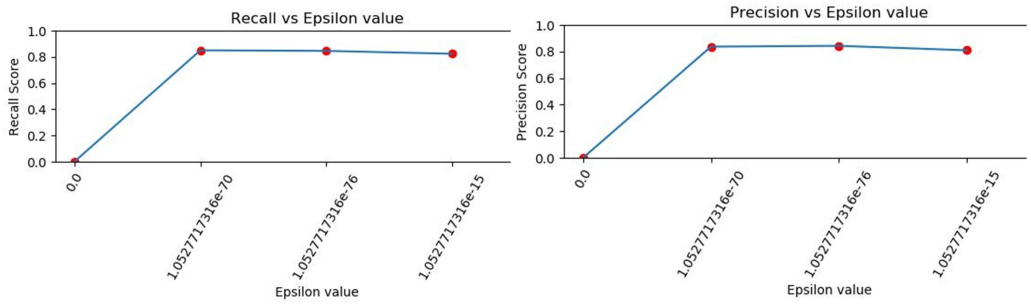


Ilustración 17. Gráficas de la Precisión y Recall, Distribución Gaussiana VS ataque TCP-SYN (KDD'99).

La ilustración 18 hace referencia al F1-Score obtenido con cada uno de los tres épsilon donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-76**.

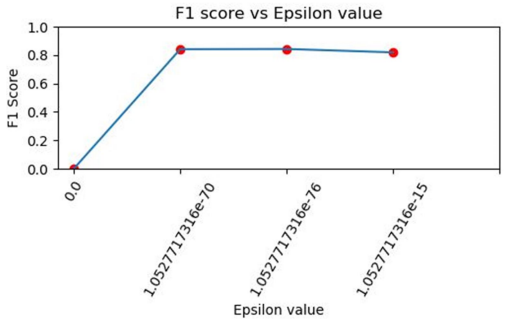


Ilustración 18. Gráficas del F1-Score, Distribución Gaussiana VS ataque TCP-SYN (KDD'99).

8.1.3 TERCERA EVALUACIÓN: CONJUNTO DATOS REAL (NSL-KDD)

En este capítulo se muestra cada uno de los resultados obtenidos al analizar los tres conjuntos de datos (uno por ataque) obtenidos a partir del NSL-KDD con el algoritmo de Distribución Gaussiana.

ATAQUE DE FUERZA BRUTA

Para el conjunto de datos de Fuerza Bruta pertenecientes al NSL-KDD los épsilon seleccionados son los siguientes: **1.0527717316e-72, 1.0527717316e-76, 1.0527717316e-15**, habrá que deducir que épsilon es más favorable para el algoritmo.

Epsilon	1.0527717316e-72	1.0527717316e-76	1.0527717316e-15
Recall	0.926544	0. 989811	0. 899323
Precision	0.960129	0. 940023	0. 931021
F1-Score	0.940002	0. 969822	0. 914569

Tabla 8. Resultados obtenidos con Distribución Gaussiana vs Fuerza Bruta (NSL-KDD).

El resultado más importante para verificar que épsilon funciona mejor es el F1-Score (promedio armónico entre el Recall y la Precision), lo que finalmente se quiere comparar con los demás.

De la tabla anterior de deduce que el épsilon que mejor F1-Score obtiene contra un ataque de Fuerza Bruta es el **1.0527717316e-76,** con un porcentaje de acierto del **96.9822**, esto quiere decir que dentro del total de anomalías que detecta, un 96 por ciento pertenece al ataque de Fuerza Bruta, en otras palabras solo 4 de cada 100 datos son considerados falsos positivos (transacciones legítimas consideradas como anómalas).

La ilustración 19 muestra una comparativa de la Precision y Recall respectivamente, la primera gráfica muestra la Precision obtenida en cada uno de los tres épsilon seleccionados donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-76**, mientras que la segunda hace lo mismo del Recall donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-72**.

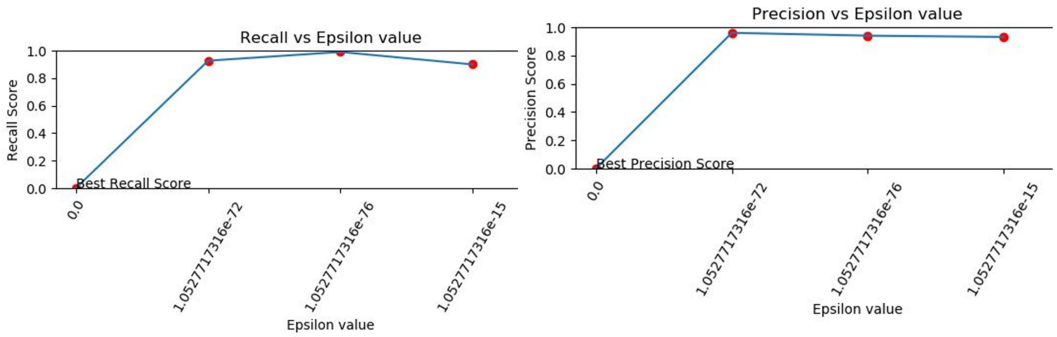


Ilustración 19. Gráficas de la Precisión y Recall, Distribución Gaussiana VS ataque Fuerza Bruta (NSL-KDD).

La ilustración 20 muestra el F1-Score obtenido con cada uno de los tres épsilon, donde el mejor resultado obtenido corresponde al épsilon 1**.0527717316e-76**.

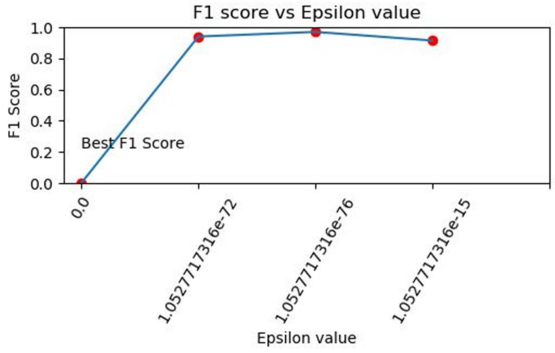


Ilustración 20. Gráficas del F1-Score, Distribución Gaussiana VS ataque Fuerza Bruta (NSL-KDD).

ATAQUE HTTP-FLOOD

Para el conjunto de datos de Http-Flood pertenecientes al NSL-KDD los épsilon seleccionados son los siguientes: **1.0527717316e-70, 1.0527717316e-50, 1.0527717316e-24**, para finalmente averiguar que épsilon funciona mejor.

Epsilon	1.0527717316e-70	1.0527717316e-50	1.0527717316e-24
Recall	0.959832	0.948988	0.930985
Precision	0.886892	0.862434	0.852399
F1-Score	0.928453	0.907657	0.883245

Tabla 9. Resultados obtenidos con Distribución Gaussiana vs Http-Flood (NSL-KDD).

El resultado más importante para verificar que épsilon funciona mejor es el F1-Score (promedio armónico entre el Recall y la Precision), lo que finalmente se quiere comparar con los demás.

De la tabla anterior de deduce que el épsilon que mejor F1-Score obtiene contra un ataque Http-Flood es el **1.0527717316e-70,** con un porcentaje de acierto del **92.8453**, esto quiere decir que dentro del total de anomalías que detecta, un 96 por ciento pertenece al ataque de Http-Flood, en otras palabras solo 3 de cada 100 datos son considerados falsos positivos (transacciones legítimas consideradas como anómalas).

La ilustración 21 muestra una comparativa de la Precision y Recall respectivamente, la primera gráfica muestra la Precision obtenida en cada uno de los tres épsilon seleccionados donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-70**, mientras que la segunda hace lo mismo del Recall donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-70**.

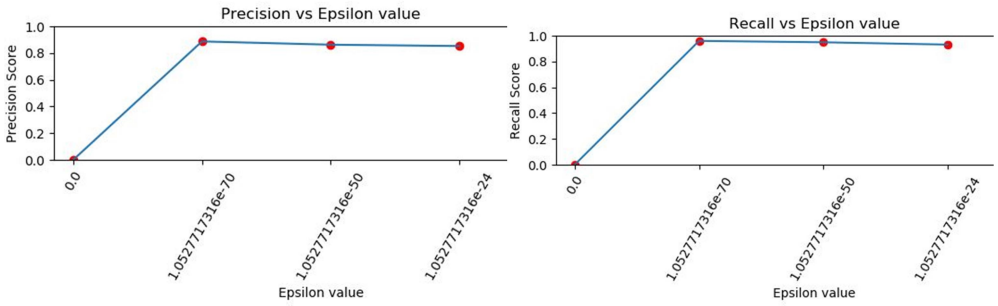


Ilustración 21. Gráficas de la Precisión y Recall, Distribución Gaussiana VS ataque Http-Flood (NSL-KDD).

La ilustración 22 muestra el F1-Score obtenido con cada uno de los tres épsilon, donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-70**.

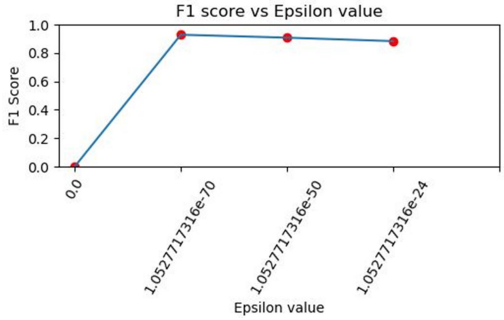


Ilustración 22. Gráficas del F1-Score, Distribución Gaussiana VS ataque Http-Flood (NSL-KDD).

ATAQUE TCP-SYN

Para el conjunto de datos de TCP-SYN pertenecientes al NSL-KDD los épsilon seleccionados son los siguientes: **1.0527717316e-70, 1.0527717316e-76, 1.0527717316e-15**.

Epsilon	1.0527717316e-70	1.0527717316e-76	1.0527717316e-15
Recall	0.906867	0.947855	0.949826
Precision	0.946724	0.929275	0.920002
F1-Score	0.921347	0.938888	0.934492

Tabla 10. Resultados obtenidos con Distribución Gaussiana vs TCP-SYN (NSL-KDD).

El resultado más importante para verificar que épsilon funciona mejor es el F1-Score (promedio armónico entre el Recall y la Precision), lo que finalmente queremos comparar con los demás.

De la tabla anterior de deduce que el épsilon que mejor F1-Score obtiene contra un ataque TCP-SYN es el **1.0527717316e-76,** con un porcentaje de acierto del **93.8888**, esto quiere decir que dentro del total de anomalías que detecta, un 96 por ciento pertenece al ataque de TCP-SYN, en otras palabras solo 7 de cada 100 datos son considerados falsos positivos (transacciones legítimas consideradas como anómalas).

La ilustración 23 muestra una comparativa de la Precision y Recall respectivamente, la primera gráfica muestra la Precision obtenida en cada uno de los tres épsilon seleccionados donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-76**, mientras que la segunda hace lo mismo del Recall donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-15**.

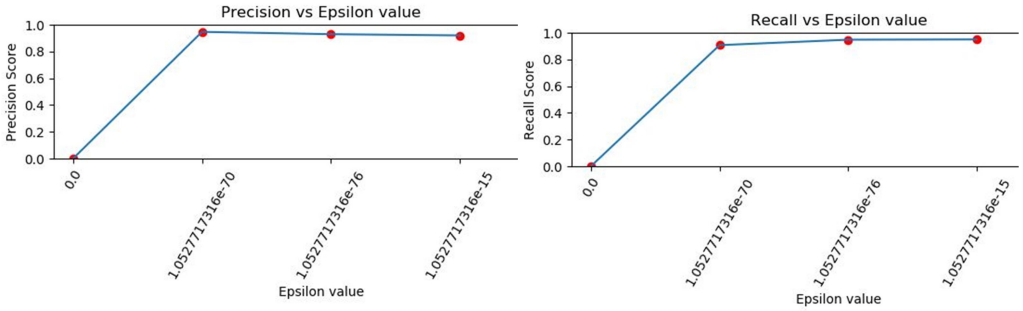


Ilustración 23. Gráficas de la Precisión y Recall, Distribución Gaussiana VS ataque TCP-SYN (NSL-KDD).

La ilustración 24 muestra el F1-Score obtenido con cada uno de los tres épsilon seleccionados, donde el mejor resultado obtenido corresponde al épsilon **1.0527717316e-76.**

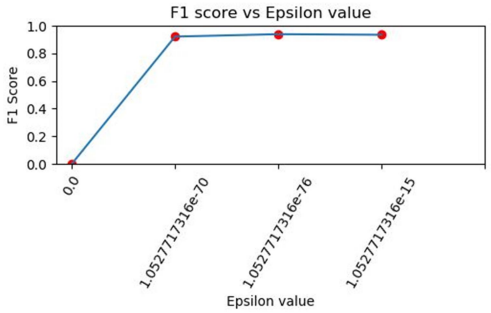


Ilustración 24. Gráficas del F1-Score, Distribución Gaussiana VS ataque TCP-SYN (NSL-KDD).

8.2 ANÁLISIS CON ALGORITMO ISOLATION FOREST

En este capítulo se mostrarán los análisis producidos por este algoritmo en cada uno de los tres entornos explicados anteriormente, en el orden siguiente: primera evaluación (entorno virtualizado y controlado), segunda evaluación (entorno con poco tráfico pero real correspondiente al conjunto KDD'99) y tercera evaluación (correspondiente al conjunto NSL-KDD), pudiéndose distinguir tres métricas a través de las cuales medir la efectividad del algoritmo con cada conjunto de datos: Precision, Recall y F1-Score, explicados de forma más detallada en el capítulo 5.1.1 (métricas).

En el algoritmo de Distribución Gaussiana se comprueba que el resultado varía en función del épsilon que se elija, en este caso se requiere el uso de hiperparámetros, variables que influyen en el resultado final, por lo que serán expuestos unos ejemplos donde se podrá comprobar el efecto de los mismos.

Muchos modelos contienen parámetros que no pueden aprenderse a partir de datos de entrenamiento y que por lo tanto se deben establecer de forma manual por el analista, estos son conocidos como hiperparámetros. Como se comenta en el anterior párrafo los resultados de un modelo pueden variar en función de los hiperparámetros que se elijan pero es muy difícil saber cuál es el adecuado. Con la práctica, muchos especialistas en Machine Learning aprenden a intuir que valores puede ser los mejores para cada modelo y algoritmo, sin embargo existe un método a través del cual encontrar los valores óptimos llamado "Grid Search".

El "Grid Search" es un proceso que consiste en realizar un ajuste de hiperparámetros para determinar los valores óptimos para un modelo dado. Esto es significativo ya que el rendimiento de todo el modelo se basa en los valores de hiperparámetros especificados.

Antes de continuar con el proceso hay que definir los hiperparámetros del Isolation Forest [47]:

- **N_estimators:** el número de estimadores base en el conjunto. Su valor por defecto es 100.
- **Max_samples:** número de muestras a extraer para entrenar cada estimador base. Valor por defecto automático, media entre 256 y el número de ejemplos que posee el conjunto.
- **Contamination:** cantidad de datos malignos que posee el conjunto de datos que está analizando el algoritmo. Su valor por defecto es 0.1 (de cada 10 transacciones 1 es ilegítima).
- **Max_features:** número de características a analizar por el algoritmo. Por defecto coge todas.
- **Bootstrap:** ajusta los árboles individuales en conjuntos aleatorios de datos de datos de entrenamiento muestreados con reemplazo. Por defecto es falso por lo que lo hace sin reemplazo.
- **N_jobs:** Número de trabajos que se ejecutarán en paralelo para ajustar y predecir. Por defecto ninguno.
- **Behaviour:** Comportamiento de la función decisión. Por defecto es "old". Por lo que no se adaptará para coincidir con otro algoritmo de detección de anomalías API.

Gracias a [47] es posible determinar que los hiperparámetros más importantes son *N_estimators, Max_samples, Contamination y Max_features* por lo tanto los demás no se tendrán en cuenta a

la hora de evaluar los resultados obtenidos y su valor siempre será el que poseen por defecto. El hiperparámetro Max_features tampoco se manipulará ya que siempre se quiere usar todas las características de los conjuntos en cada uno de los análisis.

Hay otros factores a tener en cuenta que también pueden influir de forma menos importante sobre el resultado final, por lo que es recomendable dejarlos en sus valores por defecto, ya que así poseen un buen ajuste de los datos:

- **Test_size:** cantidad de datos de prueba que posee el conjunto de datos que está analizando el algoritmo, su valor por defecto es 0.75.
- **Train_size:** cantidad de datos de entrenamiento que posee el conjunto de datos que está analizando el algoritmo, su valor por defecto es 0.25.

Se van a usar los mismos valores de los hiperparámetros para todos los conjuntos de datos, de esta forma todos los análisis estarán en las mismas condiciones, los valores de los hiperparámetros utilizados son los siguientes:

- n_estimators: list (range (100, 200, 400, 800, 1600).
- max_samples: list (range (100, 200, 400, 800, 1600).
- contamination: [0.001, 0.01, 0.1, 0.2, 0.5]

Dentro de cada evaluación se muestran dos tipos de resultados, el primero equivale a las métricas obtenidas como resultado de un análisis con los hiperparámetros por defecto, es decir sin que sufran ninguna modificación, mientras que el segundo equivale al resultado obtenido al utilizar el "Grid Search" donde se podrá observar que tipo de hiperparámetro ha sido elegido en cada evaluación y que modificación positiva o negativa ha sufrido el resultado final.

8.2.1 PRIMERA EVALUACIÓN: ENTORNO VIRTUAL (FUERZA BRUTA)

Se realiza una primera evaluación al conjunto de datos capturado en un **entorno virtualizado** simulando el ataque de **Fuerza Bruta** obteniendo el siguiente resultado (*con los hiperparámetros por defecto*).

Precision	Recall	F1-Score
0.858484	0.888936	0.870069

Tabla 11. Resultados Fuerza Bruta con hiperparámetros por defecto.

Los mejores hiperparámetros para el modelo y el resultado final de las métricas son los siguientes (*con modificaciones en los hiperparámetros*).

Mejor *n_estimator*: **800** Mejor *max_samples*: **1600** Mejor *contamination*: **0.01**

Precision	Recall	F1-Score
0.858489	0.889777	0.875005

Tabla 12. Resultados Fuerza Bruta con los mejores hiperparámetros.

Como es posible comprobar ha mejorado un **0.005% la Precision, un 0.12% el Recall y un 0.5050% el F1-Score** final, aunque las mejorías son mínimas ha favorecido el uso de los hiperparámetros para medir la eficacia del modelo.

La ilustración 25 es la representación de los datos en un espacio bidimensional formando un árbol. Como se puede apreciar los datos de entrenamiento y prueba forman la raíz del árbol (color blanco y verde), sin embargo los datos anómalos están situados de forma aleatoria alejándose de la raíz (color rojo). De esta forma cuanto más cercanas están situadas las transacciones anómalas de la raíz más complicado es para el algoritmo clasificar cada transacción de forma correcta.

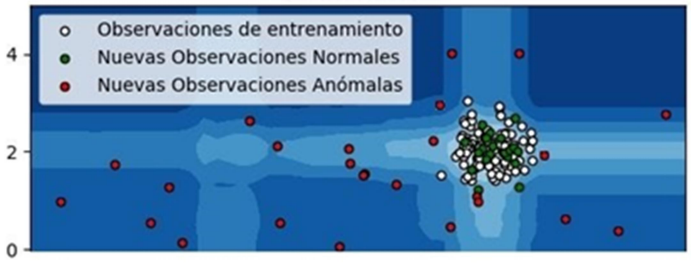


Ilustración 25. Representación Árbol Fuerza Bruta.

8.2.2 SEGUNDA EVALUACIÓN: KDD'99

En este capítulo se muestran los resultados obtenidos al analizar los ataques pertenecientes a la segunda fase de generación y/o captación de datos extraídos del conjunto de datos de KDD'99 con el algoritmo de Isolation Forest.

HTTP-FLOOD (KDD'99)

En un primer lugar se analiza el conjunto perteneciente al ataque **Http-Flood** obteniéndose el siguiente resultado (*con los hiperparámetros por defecto*).

Precision	Recall	F1-Score
0.9286954	0.949263	0.935911

Tabla 13. Resultados Http-Flood con hiperparámetros por defecto.

Los mejores hiperparámetros para el modelo y el resultado final de las métricas son los siguientes (*con modificaciones en los hiperparámetros*).

Mejor *n_estimator*: **800** Mejor *max_samples*: **1600** Mejor *contamination*: **0.1**

Precision	Recall	F1-Score
0.940127	0.969922	0.956200

Tabla 14. Resultados Http-Flood con los mejores hiperparámetros.

Como es posible comprobar ha mejorado un **2.8% la Precision, un 2.1% el Recall y un 2.4% el F1 Score**, de esta forma se puede comprobar la mejoría de los resultados gracias a los hiperparámetros.

La ilustración 26 hace referencia a la representación de datos anómalos, de entrenamiento y de prueba en un espacio bidimensional formando un árbol. Como se puede apreciar los datos de entrenamiento y prueba forman la raíz (color blanco y verde), sin embargo los datos anómalos (color rojo) no siguen ningún patrón por lo que están situados de forma aleatoria en el árbol. De esta forma cuanto más cercanas están situadas las transacciones anómalas de la raíz más complicado es para el algoritmo clasificar cada transacción de forma correcta.

Ilustración 26. Representación Árbol Http-Flood (KDD'99).

TCP-SYN (KDD'99)

En un segundo lugar se analiza el conjunto perteneciente al ataque **TCP-SYN** obteniéndose el siguiente resultado (*con los hiperparámetros y la contaminación por defecto*).

Precision	Recall	F1-Score
0.869002	0.878883	0.872231

Tabla 15. Resultados TCP-SYN con hiperparámetros por defecto.

Los mejores hiperparámetros para el modelo y el resultado final de las métricas son los siguientes (*con modificaciones en los hiperparámetros*).

Mejor *n_estimator*: **800** Mejor *max_samples*: **800** Mejor *contamination*: **0.01**

Precision	Recall	F1-Score
0.865001	0.898700	0.884008

Tabla 16. Resultados TCP-SYN con mejores hiperparámetros.

Como es posible comprobar ha empeorado un **0.4% la Precision,** sin embargo **el Recall y el F1-Score poseen una mejoría del 2% y 1.3%** respectivamente, por lo tanto aun teniendo un decremento en la Precision, se elige este hiperparámetro ya que el dato importante es el F1-score (promedio armónico de los anteriores).

La ilustración 27 es la representación de los datos en un espacio bidimensional formando un árbol. Como se puede apreciar los datos de entrenamiento y prueba (color blanco y verde) forman una agrupación (raíz), sin embargo los datos anómalos están situados de forma aleatoria en el árbol (color rojo). De esta forma cuanto más cercanas están situadas las transacciones anómalas de la raíz más complicado es para el algoritmo clasificar cada transacción de forma correcta.

Ilustración 27. Representación Árbol TCP-SYN (KDD'99).

8.2.3 TERCERA EVALUACIÓN: CONJUNTO DATOS REAL (NSL-KDD)

La siguiente tabla muestra los resultados al analizar los tres conjuntos obtenidos durante la tercera fase de generación y/o captación de datos, pertenecientes al conjunto NSL-KDD con el algoritmo Isolation Forest.

Al contrario que en las evaluaciones anteriores, no ha sido necesario realizar pruebas con los hiperparámetros ya que los resultados obtenidos desde un primer momento han sido óptimos.

Tipo de Ataque	Fuerza Bruta	Http-Flood	TCP-SYN
Recall	0.923335	0.992990	0.954478
Precision	0.904334	0.991209	0.930045
F1-Score	0.913988	0.991221	0.940099

Tabla 17. Resultados obtenidos al analizar cada uno de los tres ataques de NSL-KDD con Isolation Forest.

Como se puede observar de la tabla anterior, el ataque perteneciente al NSL-KDD más fácil para detectar para el algoritmo de Isolation Forest es el Http-Flood con un 99% de eficacia.

Las tres ilustraciones que aparecen a continuación muestran la posición en un espacio bidimensional (árbol) de cómo están situados los tres tipos de datos que analizamos en este algoritmo: datos normales, de entrenamiento y maliciosos para cada uno de los tres ataques cuyos datos han sido sustraídos del conjunto de datos NSL-KDD.

FUERZA BRUTA *TCP-SYN*

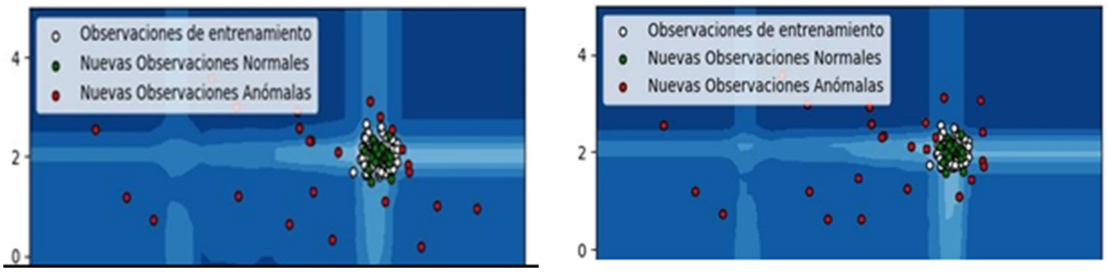


Ilustración 28. Representaciones de Árboles pertenecientes a Fuerza Bruta y TCP-SYN, ambos del NSL-KDD.

HTTP-FLOOD

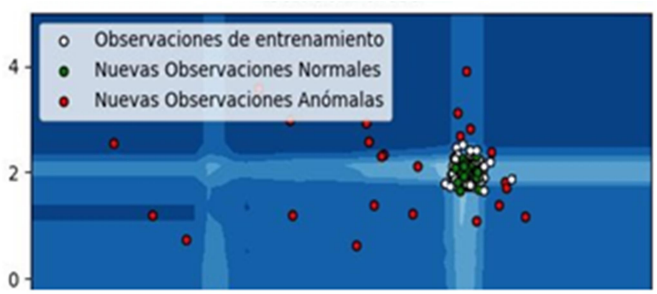


Ilustración 29. Representación Árbol Http-Flood (NLS-KDD).

Si se realiza una comparación con las gráficas obtenidas para la primera y segunda evaluación se puede observar que la raíz formada por los datos de entrenamiento y prueba posee una menor agrupación comparado con las gráficas expuestas en la tercera evaluación, por lo tanto se

puede hacer una pequeña conclusión de que los resultados obtenidos con el conjunto de datos NSL-KDD son mejores que los obtenidos con el entorno virtualizado (1ª evaluación) y KDD'99 (2ª evaluación),algo que era de esperar.

9. COMPARATIVA DE LOS RESULTADOS

En este último capítulo de análisis de resultados, se van a llevar a cabo tres comparaciones diferentes.

- En un primer lugar se compararán los resultados obtenidos de la **Distribución Gaussiana**, eligiendo que ataque se ha detectado con mayor exactitud, este proceso se verificará gracias a las curvas **ROC y AUC**, procesos que serán explicados más adelante. Posteriormente se verificará que los resultados no son engañosos y poseen fiabilidad gracias a la matriz de confusión, proceso que será explicado en las siguientes secciones.
- En un segundo lugar se compararán los resultados obtenidos de **Isolation Forest**, eligiendo que F1-Score es mejor en función de los hiperparámetros elegidos gracias al método "Grid Search". Posteriormente se verificará que los resultados no son engañosos y poseen fiabilidad gracias a la matriz de confusión, proceso que será explicado en las siguientes secciones.
- Por último, se realizará una comparativa entre los dos resultados seleccionados anteriormente(los mejores de cada algoritmo) con los resultados que se obtiene gracias al conjunto de datos reales NSL-KDD (cuyo resultado es 100% fiable) para determinar si los resultados que han sido obtenidos en este trabajo tienen alguna validez ya que deberían ser similares a los últimos mencionados.

9.1 MEJOR RESULTADO DISTRIBUCIÓN GAUSSIANA

Para este tipo de algoritmos se usaron en un principio nueve valores de épsilon para detectar las anomalías (tres para cada ataque), como es posible apreciar en la tabla 12.

Fuerza bruta	1.0527717316e-72	1.0527717316e-76	1.0527717316e-15
Http-Flood (KDD'99)	1.0527717316e-70	1.0527717316e-50	1.0527717316e-24
TCP-SYN (KDD'99)	1.0527717316e-70	1.0527717316e-76	1.0527717316e-15

Tabla 18. Selección inicial de todos los épsilon utilizados por el algoritmo de Distribución Gaussiana.

Posteriormente tras una comparación de los resultados obtenidos con cada uno de ellos, el número se redujo a tres seleccionando el mejor épsilon para cada ataque.

Tipo de ataque	**Fuerza Bruta**	**Http-Flood (KDD'99)**	**TCP-SYN (KDD'99)**
Mejor Épsilon	1.0527717316e-72	1.0527717316e-70	1.0527717316e-76

Tabla 19. Épsilon seleccionados finalmente para cada ataque.

En esta sección se elegirá para qué ataque tiene mayor eficiencia este algoritmo gracias a las comparativas realizadas entre las curvas ROC y AUC generadas por cada uno de los tres épsilon finalmente seleccionados.

La **curva ROC** (*Receiver Operating Characteristic*) es una representación gráfica donde la tasa de falsos positivos está situados en el eje X y la tasa de verdaderos positivos en el eje Y.

Tasa de verdaderos positivos (TPR) es sinónimo de exhaustividad →

$$TPR = \frac{VP}{VP + FN}$$

Ilustración 30. Tasa Verdaderos

Tasa de falsos positivos (FPR) se define de la siguiente manera →

$$FPR = \frac{FP}{FP + VN}$$

Ilustración 31. Tasa de Falsos Positivos.

Esta curva constituye un método estadístico para determinar la exactitud diagnóstica del algoritmo, siendo utilizadas con dos propósitos específicos: determinar el punto de corte en el que se alcanza la sensibilidad y especificidad más alta y evaluar la capacidad discriminativa del algoritmo, es decir, su capacidad de diferenciar datos legítimos y anómalos [40].

La **curva AUC** (*Área Under Curve*), como su nombre indica, es el área que está debajo de la curva ROC, cuando más grande es esa área, mejor resultado se obtiene. La ilustración 32 hace referencia a la representación típica que suele tener este tipo de áreas.

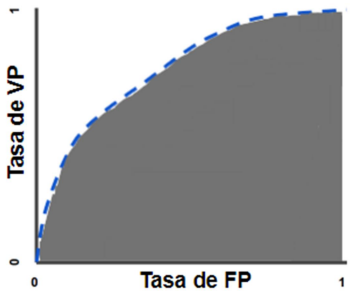


Ilustración 32. Ejemplo Gráfica Curva AUC.

Una forma de interpretar el AUC es como la probabilidad de que el modelo clasifique un ejemplo positivo aleatorio más alto que un ejemplo negativo aleatorio [41]. Por lo tanto se puede deducir, midiendo la gráfica de derecha a izquierda como veremos en las siguientes ilustraciones que el mejor resultado se obtendría situando el extremo de la gráfica en la esquina superior izquierda, donde el % de falsos positivos sea 0 y el % de verdaderos positivos sea 100%, la curva AUC poseerá el valor 1(máximo).

A modo de guía para interpretar las curvas ROC se han establecido los siguientes intervalos para los valores de AUC [48]:

[0.5]: Es como lanzar una moneda.
[0.5, 0.6): Test malo.
[0.6, 0.75): Test regular.
[0.75, 0.9): Test bueno.
[0.9, 0.97): Test muy bueno.
[0.97, 1): Test excelente.

9.1.1 ATAQUE DE FUERZA BRUTA

La curva **ROC** y **AUC** obtenidas durante el análisis del **ataque de fuerza ruta** son las siguientes:

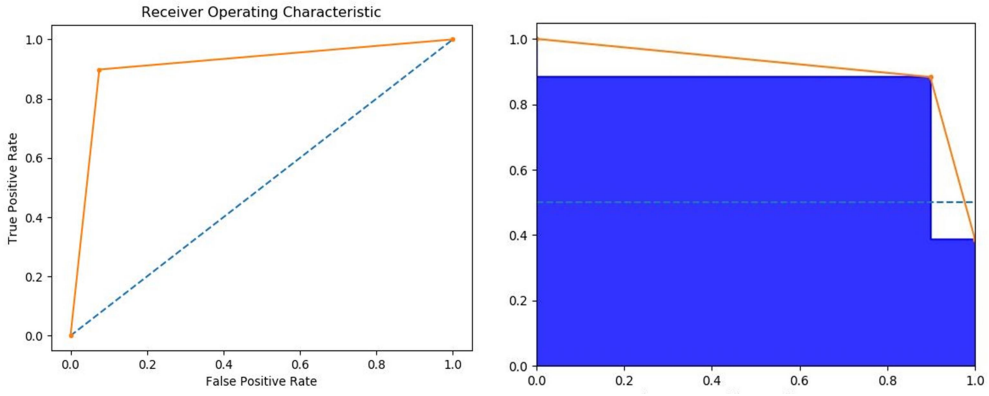


Ilustración 33. Curvas ROC y AUC en análisis de Fuerza Bruta.

El valor de la curva AUC es: **0.911,** por lo tanto según la guía estándar obtenida en [48] es posible clasificar este test como **muy bueno**.

9.1.2 ATAQUE HTTP-FLOOD

La curva **ROC** y **AUC** obtenidas durante el análisis del **ataque Http-Flood** son las siguientes:

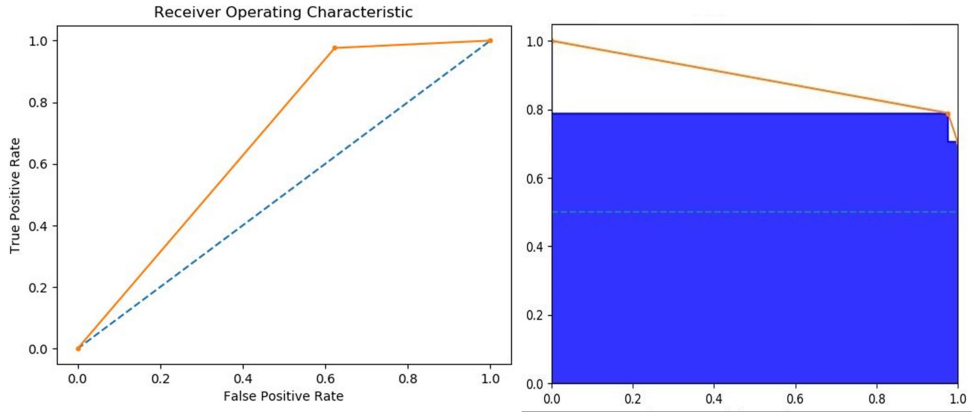


Ilustración 34. Curvas ROC y AUC en análisis de Http-Flood.

El valor de la curva AUC es: **0.768**, por lo tanto según la guía estándar obtenida en [48] es posible clasificar este test como **bueno**.

9.1.3 ATAQUE TCP-SYN

La curva **ROC** y **AUC** obtenidas durante el análisis del **ataque TCP-SYN** son las siguientes:

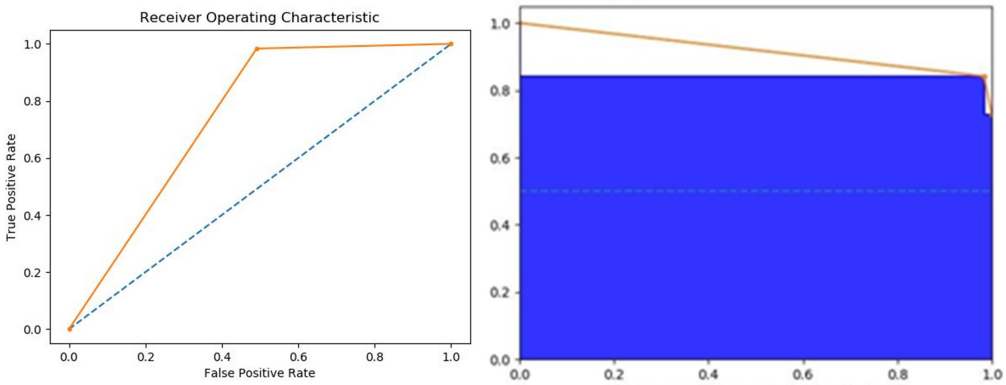


Ilustración 35. Curvas ROC y AUC en análisis de TCP-SYN.

El valor de la curva AUC es: **0.816**, por lo tanto según la guía estándar obtenida en [48] es posible clasificar este test como **bueno**.

Como se puede comprobar en la tabla 20, el ataque con mejor curva AUC es el ataque de Fuerza Bruta con más de 0.9, por lo tanto ese es el ataque que mejor detecta el algoritmo de distribución Gaussiana.

Tipo de Ataque	Fuerza Bruta	Http-Flood	TCP-SYN
Valor Curva AUC	0.911	0.768	0.816

Tabla 20. Resultados de las curvas AUC.

Gracias a la curva ROC y AUC se puede comprobar el ratio de verdaderos positivos y falsos positivos y aportar un mayor grado de fiabilidad a los resultados pero según expertos en la materia de Machine Learning para detección de anomalías existe un método de comprobación que es obligatorio usar, **la matriz de confusión.**

9.1.4 MATRIZ DE CONFUSIÓN

Recordemos que el algoritmo de Distribución Gaussiana no es un algoritmo de clasificación y que gracias al modelado que realiza es capaz de detectar el tráfico malicioso como anomalía. Gracias a este algoritmo obtenemos *la Precision (Número de verdaderos positivos / Número de verdaderos Positivos + Falsos positivos)* y *el Recall (Número de verdaderos positivos / Número de verdaderos positivos + Falsos negativos)*, a través de los cuales se obtiene el F-1 score, dato utilizado junto con la curva AUC para comprobar que resultado es mejor.

Sin embargo existen una serie de situaciones donde los parámetros anteriores aun siendo muy positivos pueden ser erróneos como se puede comprobar con los siguientes ejemplos.

Se puede dar la situación de poseer 100 transacciones pertenecientes al ataque de red y 1000 legítimas, y clasificar dos maliciosas como anómalas mientras que las transacciones restantes son clasificadas como legítimas (1098), en este caso la Precision será del 100% ya que todos los clasificados como anómalos eran maliciosos. Por lo tanto, existe una tasa muy alta de falsos negativos, una gran cantidad de transacciones clasificadas como legítimas cuando no lo son, por lo que en definitiva la clasificación del ejemplo propuesto es bastante negativa.

Sin embargo puede existir un caso en especial en el cual no importe que se dejen de detectar transacciones como maliciosas pero es fundamental no detectar algo legítimo como malicioso, este caso puede suceder en según qué herramientas que al detectar tráfico malicioso cortan el tráfico, por lo que es muy importante la Precision, y no puedes permitirte la existencia de falsos positivos.

Por otra parte, el Recall es todo lo contrario, en el ejemplo anterior propuesto el Recall habría sido muy negativo (cerca del 5%). Esta medición es fundamental en casos de uso donde el objetivo es detectar un ataque de red, es decir, que los falsos negativos sean los mínimos posibles.

Otro matiz a tener en cuenta a la hora de valorar si un resultado es eficaz o no es el equilibrio de los datos, para obtener un análisis fiable es necesario que la cantidad de datos legítimos no sea excesivamente superior a la cantidad de datos maliciosos, por ejemplo si solo se posee 5 transacciones maliciosas mientras que 1000 son legítimas ningún resultado será fiable ya que aunque no se detectará ninguna transacción maliciosa como anómala el Recall seguirá siendo muy alto (1000/1000+5).

Como pequeña conclusión a esta sección, todos los resultados obtenidos deben ser comprobados de forma minuciosa ya que dependen de muchos factores, para realizar esta comprobación existe la mencionada anteriormente matriz de confusión [38]. Dicha matriz es una herramienta que permite la visualización del desempeño del algoritmo trabajado. Mostrando el siguiente resultado:

	N (modelo)	S (modelo)
n (real)	Negativos Reales	Falsos Positivos
p (real)	Falsos Negativos	Positivos reales

Ilustración 36. Representación Matriz de confusión.

De esta forma la siguiente gráfica hace referencia a la matriz de confusión del mejor resultado obtenido con la curva AUC, en este caso **al ataque de Fuerza Bruta cuya curva AUC es 0.911**.

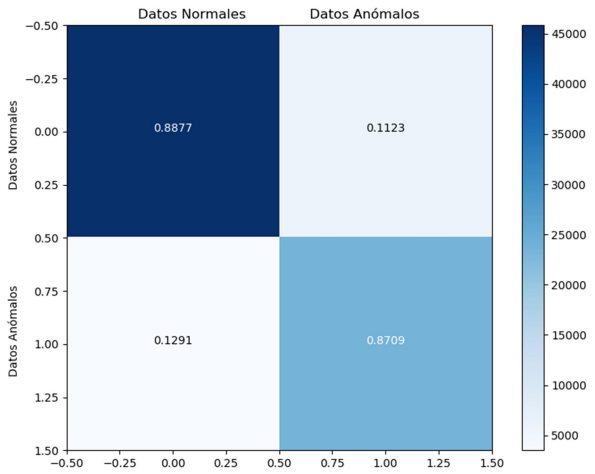


Ilustración 37. Matriz de Confusión para el ataque de Fuerza Bruta con Distribución Gaussiana.

De la ilustración anterior se deducen los siguientes resultados:

- Datos normales detectados como legítimos (verdaderos negativos [TN]) → **88.77%**
- Datos normales detectados como maliciosos(falso positivo [FP]) → **11.23%**
- Datos maliciosos detectados como ilegítimos (verdaderos positivos [TP]) → **87.09%**
- Datos maliciosos detectados como legítimos (falso negativo [FN]) → **12.91%**

Accuracy = (TP + TN) / (TP + FP + FN + TN) = (87.09 + 88.77) / (87.09 + 11.23 + 12.91 + 88.77) = **87.93 %**
Precision = TP / (TP + FP) = 87.09 / (87.09 + 11.23) = **88.57%**
Recall = TP / (TP + FN) = 87.09 / (87.09 + 12.91) = **87.09%**
F1 = (2 * Precision * Recall) / (Precision + Recall) = (2 * 88.57 * 87.09) / (88.57 + 87.09) = **87.82%**

Se puede comprobar gracias a los datos de falsos positivos y falsos negativos y al F1-score que la clasificación de los datos es bastante correcta, por lo que el análisis sobre el ataque de fuerza bruta por parte del algoritmo de Distribución Gaussiana ha resultado ser muy positivo.

Aun obteniendo muy buenos resultados con los algoritmos siempre hay que analizarlos ya que existen situaciones donde estos resultados puedan ser engañosos. Hay diferentes métodos de comprobarlo, uno de los más eficaces es la matriz de confusión donde es posible distinguir el número de falsos positivos o falsos negativos y comprobar si los resultados son fiables o no.

Si se diera la situación de que gracias a la matriz de confusión se determina que los resultados obtenidos son erróneos habría que seleccionar el segundo mejor resultado que en este caso sería el TCP-SYN o incluso volver a capturar tráfico ya que este podría estar desequilibrado.

9.2 MEJOR RESULTADO ISOLATION FOREST

Una vez analizado que ataque detecta de mejor forma el algoritmo de Distribución Gaussiana, se realiza de la misma manera con el Isolation Forest. En este caso al utilizar las mismas métricas que el algoritmo de Distribución Gaussiana el dato que mayor importancia posee es el F1-Score (media armónica entre los dos valores anteriores, Recall y Precision).

Tipo de Ataque	Fuerza Bruta	Http-Flood(KDD-99)	TCP-SYN (KDD-99)
Recall	0.889777	0.969922	0.898700
Precision	0.858489	0.909127	0.865001
F1-Score	0.875005	0.956200	0.884008

Tabla 21. Resultados obtenidos al analizar los datos pertenecientes al entorno virtualizado y KDD'99 con Isolation Forest.

Los datos están bastante diferenciados numéricamente hablando por lo que se puede comprobar que el ataque mejor ha detectado el algoritmo es el **Http-Flood** con un 0.956% en el valor de F1-Score.

Los otros ataques también poseen resultados buenos, muchos análisis estarían satisfechos con esos datos también. Al igual que sucede con el algoritmo de Distribución Gaussiana, es necesario hacer una comprobación de los datos de exactitud obtenidos para comprobar los % de falsos negativos o falsos positivos de este algoritmo con la matriz de confusión. En este caso la necesidad es mayor ya que no existe una curva ROC/AUC en este algoritmo que ayude a comprobar la fiabilidad de los resultados.

9.2.1 MATRIZ DE CONFUSIÓN

Para realizar la comprobación se ha seleccionado el conjunto de datos que mejor resultado ha obtenido, **en este caso el ataque Http-Flood** y utilizado la matriz de confusión para poder evaluar el modelo de clasificación utilizado.

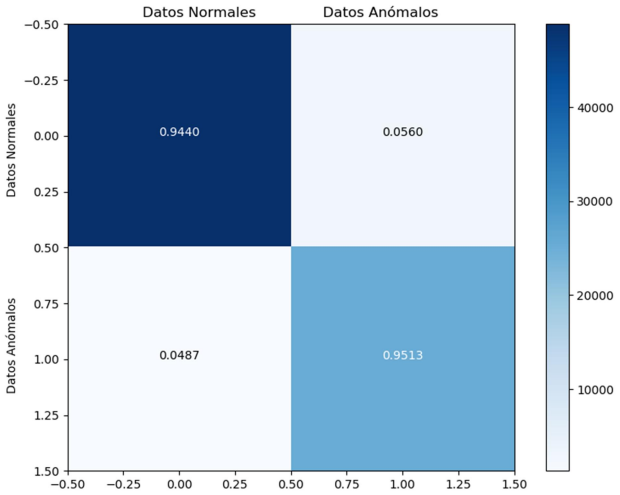


Ilustración 38. Matriz de Confusión para el ataque Http-Flood con Isolation Forest.

De la ilustración anterior se deducen los siguientes resultados:

- Datos normales detectados como legítimos (verdaderos negativos [TN]) → **94.44%**
- Datos normales detectados como maliciosos (falso positivo [FP]) → **5.56%**
- Datos maliciosos detectados como ilegítimos (verdaderos positivos [TP]) → **95.13%**
- Datos maliciosos detectados como legítimos (falso negativo [FN]) → **4.87%**

Accuracy = (TP + TN) / (TP + FP + FN + TN) = (95.13 + 94.44) / (95.13 + 5.56 + 94.44 + 4.87) = **94.78%**
Precision = TP / (TP + FP) = 95.13/ (95.13 +5.56) = **94.47%**
Recall = TP / (TP + FN) = 95.13 / (95.13 + 4.87) = **96.13%**
F1 = (2 * Precision * Recall) / (Precision + Recall) = (2 * 94.47 * 95.13) / (94.47 + 95.13) = **95.79%**

Se puede comprobar gracias a los datos de falsos positivos y falsos negativos y al F1-score que la clasificación de los datos es bastante correcta, por lo que el análisis sobre el ataque Http-Flood por parte del algoritmo Isolation Forest ha resultado ser muy positivo.

Por lo tanto los resultados obtenidos son eficaces y fiables gracias a la matriz de confusión, que nos permite visualizar los falsos positivos y los falsos negativos, datos que muestran si los resultados son correctos o no. Si se diera la situación de que gracias a la matriz de confusión se determina que los resultados obtenidos son erróneos habría que seleccionar el segundo mejor resultado que en este caso sería el TCP-SYN o incluso volver a capturar tráfico ya que este podría estar desequilibrado.

9.3 COMPARATIVA FINAL

Finalizadas las anteriores comparaciones ya se dispone de los resultados finales de los dos algoritmos con cada uno de los tres ataques y la selección del mejor resultado para cada uno de ellos.

Estos datos equivalen a un entorno virtualizado controlado (ataque de Fuerza Bruta) y al conjunto de datos KDD (ataque Http-Flood y ataque TCP-SYN), resultados muy positivos que verifican que ambos algoritmos han funcionado de forma muy correcta.

Sin embargo, estos resultados no poseen ningún carácter verídico sino se comparan con resultados obtenidos a través de conjuntos reales, minuciosamente creados para este tipo de análisis. Por lo que en esta sección se realiza una pequeña comparativa final entre los resultados obtenidos de nuestros recursos con datos fiables.

CONJUNTO DE DATOS VIRTUALIZADO Y KDD'99

Tipo de Ataque	Fuerza bruta	Http-Flood(KDD)	TCP-SYN(KDD)
Gaussiana(F1)/Curva AUC	0.890756/0.911	0.802504/0.768	0.842678/0.816
Isolation Forest (F1)	0.875005	0.956200	0.884008

Tabla 22. Mejores resultados del entorno virtualizado y KDD'99

CONJUNTO DE DATOS REAL (NSL-KDD)

Tipo de Ataque	Fuerza Bruta	Http-Flood (KDD)	TCP-SYN (KDD)
Gaussiana(F1)/Curva AUC	0.969822/0.970	0.928453/0.901	0.938888/0.912
Isolation Forest (F1)	0.913988	0.991221	0.940099

Tabla 23. Resultados obtenidos al analizar el conjunto NSL-KDD.

COMPARATIVA MATRIZ DE CONFUSIÓN

	D. Gaussiana	Isolation Forest	D. Gauss(NSL-KDD)	I.Forest(NSL-KDD)
M. Confusión	88.77%/87.09%	94.44%/95.13%	96.45%/95.98%	98.43%/99.12%

Tabla 24. Mejores resultados obtenidos de las matrices de confusión.

Como se puede visualizar en las tablas anteriores los resultados son muy similares, obteniéndose las siguientes afirmaciones:

- En ambos casos el ataque **Http-Flood** es el más difícil de detectar para el algoritmo de **Distribución Gaussiana** con los porcentajes **0.8025** para el conjunto KDD y **0.9284** para el conjunto NSL-KDD.
- En ambos casos la curva AUC generada por el algoritmo de **Distribución Gaussiana** contra el ataque **Http-Flood** es la que peor resultado ha obtenido, con los porcentajes **0.768** para el conjunto KDD y **0.901** para el conjunto NSL-KDD.
- En ambos casos el ataque de **Fuerza Bruta** es el más fácil de detectar para el algoritmo de **Distribución Gaussiana** con los porcentajes **0.8907** para el entorno virtualizado y **0.9698** para el conjunto NSL-KDD.
- En ambos casos la curva AUC generada por el algoritmo de **Distribución Gaussiana** contra el ataque de **Fuerza Bruta** es el mejor resultado obtenido, con los porcentajes **0.911** y **0.970** para el conjunto NSL-KDD.

- En ambos casos el ataque de **Fuerza Bruta** es el más difícil de detectar para el algoritmo de **Isolation Forest** con un F1-Score de **0.8750** para el entorno virtualizado y **0.9139** para el conjunto NSL-KDD.
- En ambos casos el ataque **Http-Flood** es el más fácil de detectar para el algoritmo **Isolation Forest** con un F1-Score de **0.9562** para el conjunto KDD y **0.9912** para el conjunto NSL-KDD.
- En ambos casos el algoritmo de **TCP-SYN** se encuentra en segunda posición, aunque posee resultados muy positivos igualmente.
- En todas las situaciones los resultados obtenidos en un conjunto de datos real posee mejoría sobre el resultado obtenido en un entorno virtualizado o perteneciente al conjunto KDD'99.
- El mejor resultado óptimo para cada uno de los algoritmos posee una matriz de confusión similar al porcentaje de eficacia, por lo que son resultados muy fiables.
- De esta forma se puede deducir que gracias a los resultados obtenidos en las diferentes evaluaciones realizadas a lo largo del proyecto los algoritmos funcionan de forma muy satisfactoria en los tres entornos propuestos: entorno virtualizado y controlado, conjunto de datos pertenecientes al KDD'99 y conjunto de datos pertenecientes al NSL-KDD.

Permitiendo observar una evaluación en los mismos, algo que en un futuro facilitará el uso de estos algoritmos en entornos más reales y complicados donde la detección de anomalías sea una tarea que requiera mucho más entrenamiento que el requerido por los algoritmos en este trabajo.

10. CONCLUSIONES

Podemos concluir con la consecución de los siguientes objetivos:

- Explicar la utilidad del Machine Learning, centrándose en su aplicación en la seguridad informática.
- ✓ *Conseguido: Se ha hecho un estudio detallado del aprendizaje automático y su aplicación a la seguridad.*
- Estudiar herramientas de aprendizaje automático.
- ✓ *Conseguido: Se ha aprendido a usar diversas opciones de Scikit-Learn y Pandas.*
- Estudiar algoritmos basados en aprendizaje automático y conocer su funcionamiento.
- ✓ *Conseguido: Se ha aprendido el funcionamiento de los algoritmos de distribución Gaussiana e Isolation Forest, destacando su usabilidad en la seguridad informática.*
- Estudiar y conocer diferentes ataques de red.
- ✓ *Conseguido: Se ha realizado un ataque de red a través de un entorno virtualizado y se han estudiado el funcionamiento de otros dos ataques más.*
- Realizar un análisis exhaustivo de los datos capturados.
- ✓ *Conseguido: Se han realizado distintos análisis con cada uno de los algoritmos seleccionados, pudiendo comprobar que ataque es más difícil capturar y que algoritmo funciona de una forma óptima.*

Tener la posibilidad de realizar un proyecto final de carrera donde la seguridad informática es la columna vertebral del mismo ha sido una de las experiencias académicas más importantes vividas hasta la actualidad.

Realizar un trabajo con tal volumen de datos no es una tarea fácil, esto hace que realizar la experimentación se convierta a veces en un verdadero reto. Por otro lado el haber podido

trabajar y contrastar los resultados obtenidos con datos verídicos aportados en el conjunto de datos NSL-KDD hace que la satisfacción final al haber conseguido buenos resultados sea única.

En particular, se ha profundizado en el uso de algoritmos de Machine Learning para la detección de anomalías sobre conjuntos de datos simulando entornos reales. Valorando todos los algoritmos de manera individual junto con la efectividad de los mismos con cada uno de los ataques propuestos, se ha podido comprobar la eficiencia del algoritmo de Distribución Gaussiana para detectar ataques de Fuerza Bruta y el algoritmo de Isolation Forest para detectar ataques de tipo de negación de servicio.

Sin embargo, se ha visto que en algunas ocasiones los resultados obtenidos pueden tener poca fiabilidad debido a algunas suposiciones iniciales, por lo tanto es necesario asegurar dicha fiabilidad con diferentes métodos como por ejemplo las *curvas ROC/AUC o la matriz de confusión*. También hay que tener en cuenta que es necesario invertir tiempo en entrenar al sistema para poder aumentar la eficacia de los mismos ya que en un futuro podrían ser la llave para una defensa efectiva contra los ataques informáticos.

11. ESTUDIOS FUTUROS Y AUTOEVALUACIÓN

En un principio resultó bastante complicado adquirir los conocimientos necesarios para realizar el proyecto ya que es una disciplina completamente nueva para mí y todos los procesos y herramientas necesarias para llevarlo a cabo tenían dificultad. El proceso más complicado fue conseguir adaptar los datos capturados en los diferentes entornos a los algoritmos basados en Machine Learning, ya que cada conjunto de datos poseía unas características diferentes por lo que había que realizar una optimización de cada conjunto.

Sin embargo, una vez comprendidos los conceptos básicos y finalizada la adaptación de los conjuntos el resto del proyecto era realizar tareas similares por lo que la dificultad disminuyó. Considero que se han obtenido muy buenos resultados, mucho mejores de los esperados inicialmente, que junto al conocimiento adquirido me permite tener una buena base para seguir formándome en esta disciplina.

El campo de investigación en el cual se centra este Trabajo de Fin de Grado es tan extenso, que existen numerosas vías abiertas para poder continuar estudiando los posibles beneficios que pueden aportarnos este tipo de prácticas.

Aún existe una gran necesidad de estudiar los algoritmos planteados de una forma más concisa y profunda para perfeccionar su actuación en entornos reales. También existen otros esquemas de aprendizaje que no están expuestas en este trabajo por falta de tiempo y/o información sobre los mismos, a su vez en este trabajo se exponen una serie de herramientas o ataques para la realización del mismo, sin embargo existen otras opciones cuyo resultado podría ser mejor que el obtenido actualmente por lo que sería necesarios realizar pruebas con diferentes alternativas.

En último lugar, sería una opción interesante la posibilidad de poner en práctica todos los escenarios planteados anteriormente en este trabajo para poder contrastar los resultados obtenidos en el mismo. De esta forma se podría comprobar que los resultados obtenidos son aplicables a entornos reales y que los algoritmos seleccionados actúan de una manera muy eficaz tanto en entornos virtualizados como en un entorno real.

12. BIBLIOGRAFÍA

[1] Explicación simple de Machine Learning. 8 de Julio de 2018, [Internet] [Consultado el 29 de junio de 2019] Disponible en: https://medium.com/datos-y-ciencia/la-explicaci%C3%B3n-m%C3%A1s-simple-de-machine-learning-que-jam%C3%A1s-habr%C3%A1s-leido-4fa7ac6243ba

[2] Tipos de algoritmos de Machine Learning. 4 de abril de 2019, [Internet] [Consultado el 28 de Junio de 2019] Disponible en: https://www.apd.es/algoritmos-del-machine-learning/

[3] Machine Learning en ciberseguridad. 11 de mayo de 2019, [Internet] [Consultado el 29 de Junio de 2019] Disponible en: https://www.apd.es/machine-learning-en-ciberseguridad/

[4] La distribución normal. 26 de diciembre de 2013, [Internet] [Consultado el 27 de Junio de 2019] Disponible en: https://es.slideshare.net/dj_jdo/la-distribucin-normal-29512010

[5] Distribución normal. 5 de enero de 2015, [Internet] [Consultado el 27 de Junio de 2019] Disponible en: https://www.uv.es/ceaces/pdf/normal.pdf

[6] Outlier Detection with Isolation Forest. 2 de julio de 2018, [Internet] [Consultado el 30 de Junio de 2019] Disponible en: https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e

[7] ¿Qué es y cómo funciona un ataque de fuerza bruta? 11 de febrero de 2016 [Internet] [Consultado el 2 de Julio de 2019] Disponible en: https://redinfocol.org/atacando-por-fuerza-bruta-bruteforce-1/

[8] Fuerza Bruta contra aplicaciones Web. 17 de marzo de 2017, [Internet] [Consultado el 2 de Julio de 2019] Disponible en: http://highsec.es/2014/03/fuerza-bruta-contra-aplicaciones-web-usando-hydra/

[9] THC-Hydra: Obtener credenciales de usuario por fuerza bruta. 4 de febrero de 2013, [Internet] [Consultado el 3 de Julio de 2019] Disponible en https://www.securityartwork.es/2013/02/04/thc-hydra-obtener-credenciales-de-usuario-por-fuerza-bruta/

[10]Tstat. 2008, [Internet] [Consultado el 3 de Julio de 2019] Disponible en: http://Tstat.polito.it/HOWTO.shtml

[11] Nprobe. 2007, [Internet] [Consultado el 3 de Julio de 2019] Disponible en: https://www.ntop.org/products/netflow/Nprobe/

[12] Puerta de enlace. [Internet] [Consultado el 4 de Julio de 2019] Disponible en: http://ayuda.svigo.es/index.php/otros-2/93-configuracion-red-en-virtualbox

[13] Aplicación de técnicas de Machine Learning a la detección de ataques. 4 de junio de 2018, [Internet] [Consultado el 10 de Julio de 2019] Disponible en: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/81126/11/jmrodriguez85TFM0618memoria.pdf

[14] Detección de sucesos raros con Machine Learning. Julio de 2017, [Internet] [Consultado el 10 de Julio de 2019] Disponible en: http://oa.upm.es/47931/1/TFM_ANDER_CARRENO_LOPEZ.pdf

[15] Estudio de algoritmos de detección de anomalías y propuesta de recomendaciones para su aplicación a entornos de ciberseguridad. 2016, [Internet] [Consultado el 11 de Julio de 2019] Disponible en: http://oa.upm.es/40490/1/PFC_RAQUEL_NOBLEJAS_SAMPEDRO_2016.pdf

[16] Cleverpy. 2012, [Internet] [Consultado el 11 de Julio de 2019] Disponible en: https://cleverpy.com/deteccion-de-anomalias/

[17] Netskope. 2010, [Internet] [Consultado el 11 de Julio de 2019] Disponible en: https://www.netskope.com/es/platform/machine-learning-anomaly-detection

[18] Aprendizaje no Supervisado y Detección de Anomalías: ¿Qué es una Anomalía? 3 de abril de 2018, [Internet] [Consultado el 14 de Julio de 2019] Disponible en: https://elbauldelprogramador.com/aprendizaje-nosupervisado-anomalias/

[19] Cyber Akuma. (2017, Febrero 21). Fuerza bruta con Hydra a página de login - Kali Linux. [Archivo de video]. Recuperado en https://www.youtube.com/watch?v=3hjHdkZsCCo&t=3s

[20] HTTP flood attack detection in application layer using machine learning metrics and bio inspired bat algorithm. Enero de 2019, [Internet] [Consultado el 12 de Julio de 2019] Disponible en: https://www.sciencedirect.com/science/article/pii/S2210832717301655

[21] Ataque de denegación de servicio. 31 de marzo de 2017, [Internet] [Consultado el 13 de julio de 2019] Disponible en: https://es.wikipedia.org/wiki/Ataque_de_denegaci%C3%B3n_de_servicio

[22] What is a SYN flood attack? 2019, [Internet] [Consultado el 13 de Julio de 2019] Disponible en: https://www.imperva.com/learn/application-security/syn-flood/

[23] A detailed analysis of the KDD CUP 99 Data set. 2009, [Internet] [Consultado el 14 de Julio de 2019] Disponible en: https://www.ee.ryerson.ca/~bagheri/papers/cisda.pdf

[24] A Detailed Analysis on NSL-KDD Dataset Using Various Machine Learning Techniques for Intrusion Detection. Diciembre de 2013, [Internet] [Consultado el 16 de Julio de 2019] Disponible en: https://www.ijert.org/research/a-detailed-analysis-on-nsl-kdd-dataset-using-various-machine-learning-techniques-for-intrusion-detection-IJERTV2IS120804.pdf

[25] NSL-KDD dataset. 2018, [Internet] [Consultado el 16 de Julio de 2019] Disponible en: https://www.unb.ca/cic/datasets/nsl.html

[26] ¿Qué es una puerta de enlace (Gateway)? 23 de junio de 2014, [Internet] [Consultado el 7 de Julio de 2019] Disponible en: https://www.puertadeenlace.com/faq/general/46-que-es-una-puerta-de-enlace-gateway

[27] Conoce los Ataques de Fuerza Bruta y las mejores formas de evitarlos. 4 de octubre de 2018, [Internet] [Consultado el 15 de Julio de 2019] Disponible en: https://www.gb-advisors.com/es/conoce-los-ataques-de-fuerza-bruta/

[28] Medidas de protección frente ataques de denegación de servicio (DoS). 21 de enero de 2018, [Internet] [Consultado el 15 de Julio de 2019] Disponible en: https://www.incibe-cert.es/blog/medidas-proteccion-frente-ataques-denegacion-servicio-dos

[29] Instituto Nacional de estadística. Noviembre de 2018, [Internet] [Consultado el 15 de Junio de 2019] Disponible en:

https://www.ine.es/ss/Satellite?L=es_ES&c=INESeccion_C&cid=1259925528782&p=1254735110672&pagename=ProductosYServicios%2FPYSLayout

[30] Ciberataques en España: un tercio de usuarios fue víctima en 2018. Enero 2019, [Internet] [Consultado el 15 de Junio de 2019] Disponible en: https://www.tuyu.es/ciberataques-mas-comunes-en-espana-2018/

[31] Anomaly Detection using Gaussian Distribution. 2017, [Internet] [Consultado el 10 de Junio de 2019] Disponible en: https://www.kaggle.com/shelars1985/anomaly-detection-using-gaussian-distribution

[32] Conjuntos de entrenamiento y prueba: Separación de datos. 24 de septiembre de 2017, [Internet] [Consultado el 13 de Julio de 2019] Disponible en: https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data?hl=es-419

[33] Python. 2044, [Internet] [Consultado el 10 de Julio de 2019] Disponible en: https://es.wikipedia.org/wiki/Python

[34] Introducción a la librería Scikit-Learn de Python. 2 de noviembre de 2018, [Internet] [Consultado el 17 de Junio de 2019] Disponible en: http://ligdigonzalez.com/libreria-scikit-learn-de-python/

[35] Regla 68-95-99.7. 22 de abril de 2017, [Internet] [Consultado el 17 de Junio de 2019] Disponible en: https://es.wikipedia.org/wiki/Regla_68-95-99.7

[36] MIT Lincoln Labs, 1998 DARPA Intrusion Detection Evaluation. Junio de 2000, [Internet] [Consultado el 17 de Junio de 2019] Disponible en: http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/index.html

[37] Ataque de diccionario. [Internet], 2010 [Consultado el 17 de Julio de 2019] Disponible en: https://es.wikipedia.org/wiki/Ataque_de_diccionario

[38] Machine Learning a tu alcance: La matriz de confusión. 23 de enero de 2018, [Internet] [Consultado el 28 de Julio de 2019] Disponible en: https://empresas.blogthinkbig.com/ml-a-tu-alcance-matriz-confusion/

[39] Machine Learning en ciberseguridad: ¿Cómo y para qué puede funcionar? Febrero de 2014, [Internet] [Consultado el 28 de Agosto de 2019] Disponible en: https://www.apd.es/machine-learning-en-ciberseguridad/

[40] Uso de curvas ROC en investigación clínica. Aspectos teórico-prácticos. Abril de 2012, [Internet] [Consultado el 25 de Agosto de 2019] Disponible en: https://scielo.conicyt.cl/scielo.php?script=sci_arttext&pid=S0716-10182012000200003

[41] Clasificación: ROC y AUC. Septiembre de 2018, [Internet] [Consultado el 22 de Agosto de 2019] Disponible en: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=es-419

[42] Nuevas formas de phishing, malware avanzado y los errores humanos, a la orden del día durante 2018. Febrero de 2019, [Internet] [Consultado el 19 de Agosto de 2019] Disponible en: https://www.itdigitalsecurity.es/actualidad/2017/12/nuevas-formas-de-phishing-malware-avanzado-y-los-errores-humanos-a-la-orden-del-dia-durante-2018

[43] CREATE YOUR OWN WORDLIST WITH CRUNCH. 28 de noviembre de 2018, [Internet] [Consultado el 10 de Agosto de 2019] Disponible en: https://www.securitynewspaper.com/2018/11/28/create-your-own-wordlist-with-crunch/

[44] Nmap. 2008, [Internet] [Consultado el 15 de Agosto de 2019] Disponible en: https://nmap.org/

[45] Portswigger Web Security. 2014, [Internet] [Consultado el 19 de Agosto de 2019] Disponible en: https://portswigger.net/burp

[46] Packet Sender. 2010, [Internet] [Consultado el 17 de Agosto de 2019] Disponible en: https://packetsender.com/

[47] Sklearn IsolationForest. 2007, [Internet] [Consultado el 21 de Agosto de 2019] Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[48] Curva ROC. Septiembre 2007, [Internet] [Consultado el 12 de Agosto de 2019] Disponible en: https://es.wikipedia.org/wiki/Curva_ROC

[49] Machine Learning con H2O y R. 2 de octubre de 2017, [Internet] [Consultado el 15 de Agosto de 2019] Disponible en: https://rpubs.com/Joaquin_AR/406480

[50] Cross Validation Explained: Evaluating estimator performance. Noviembre de 2018, [Internet] [Consultado el 30 de Agosto de 2019] Disponible en: https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85

13. ÍNDICE DE ILUSTRACIONES

14. ÍNDICE DE TABLAS

Anexos

ÍNDICE DE ANEXOS

ANEXO A.1 (METASPLOITABLE Y KALI LINUX)

En este anexo está la explicación detallada del proceso de instalación de Metasploitable y Kali Linux en la computadora.

El primer paso fue realizar la instalación de "Kali-Linux", en este caso a través de VirtualBox, instalándolo gracias a un disco ISO el cual hay que descargar previamente de la página oficial de Linux, una vez finalizada la descarga (proceso largo ya que el ISO ocupa bastante) hay que seguir los pasos correspondientes indicados en el instalador para realizarlo de forma correcta como por ejemplo seleccionar el tipo de S.O. que se va usará(Debian-64 Bits) o el tamaño que se quiere dar al disco. Existen diferentes versiones, en este caso se ha utilizado la versión Kali-Linux-2018.

El segundo paso es descargar el disco ISO perteneciente a Metasploitable2, es posible descargarlo desde diferentes fuentes y el proceso es el mismo seguido que en disco anterior seleccionando una serie de configuraciones.

Una vez instaladas las dos máquinas virtuales (Kali-Linux como atacante y Metasploitable como víctima) es necesario realizar una serie de configuraciones en la conexión de red antes de realizar los ataques. Kali-Linux debe estar configurada en modo "NAT" (acceso a internet) y metasploitable2 configurada como "Host-Only", de esta forma el atacante podrá visualizar a la víctima y realizar los ataques correspondientes.

En definitiva se posee dos máquinas virtuales, cada una con su IP, y gracias a las configuraciones otorgadas anteriormente es posible visualizar desde la V.M. atacante a la V.M. víctima y poder realizar los ataques sobre ella. Una forma rápida de comprobar que ambas máquinas están conectadas es realizar un ping entre ellas, si este resulta satisfactorio significa que la configuración se ha realizado de forma correcta, en caso contrario sería necesaria una reconfiguración de las redes de virtual box porque posiblemente algo esté bloqueando la conexión como por ejemplo IPtables, herramienta que en función de la configuración que posea puede bloquear paquetes de red.

ANEXO A.2 (COMANDOS DE LINUX)

En este Anexo aparece el comando entero utilizado por el software Hydra para realizar el ataque de Fuerza Bruta, con una posterior explicación de cada sección de dicho comando.

Hydra –l admin –P rockyou.txt 192.168.168.10 http-post-form"mutillidae/index.php?page=login.php:username=^USER^&password=^PASS^&Login=Login: Login failed" –V

- **Hydra** → comando que usado para utilizar dicho software.

- **-l admin** → palabra que hace referencia al usuario que pueda ser el correcto para poder realizar el login.
- **-L** → en este caso se selecciona una "userlist" generada previamente en la cual se incluye todos los usuarios que tienen indicios de ser los adecuados para acceder a la víctima, si no se posee una idea clara de cuáles pueden ser los más comunes son "admin"o "root".

- **-p** → palabra que hacer referencia a la contraseña con la cual se intentará hacer el login junto con los usuarios previamente seleccionados.

- **-P rockyou.txt** → en este caso es seleccionada una Wordlist, las wordlist se pueden encontrar en internet y cuanto más grande sea más posibilidades hay de encontrar la combinación necesario ya que comprueba cada uno de los usuarios con cada una de las contraseñas, una muy famosa es la llamada "rockyou", la cual es muy usada para este tipo de ataques.

- **192.168.56.101** → es la dirección IP que corresponde a la víctima.

- **http-post-form** → variación del ataque, en este caso es usada debido al estado del puerto 80 a través del cual realizamos el ataque.

- **/mutillidae/index.php?page=login.php** → dirección "web" que se ha generado al ejecutar el login fallido y que es posible capturar gracias a la herramienta "burp suite", esta dirección hay que escribirla tal y como nos aparece en la imagen, en caso contrario el ataque será fallido.

- **":username=^USER^&password=^PASS^&Login=Login:"** → parte del comando que requiere la variación de Hydra que se ha utilizado, al igual que con la URL tiene que estar escrito de forma correcta sino el ataque no funcionará.

- **"Authentication error: Bad username or password"** → mensaje de error que aparece cuando es realizado el primer intento de login, al igual que con las demás secciones del comando hay que escribirlo de forma idéntica a como aparece en la página.

- **-V** → permite ver como output el intento de cada ataque de hydra, con su respectivo usuario y la contraseña usada en cada caso [9].

- **N net.private** → fichero en el cual esta guardada la configuración del rango de IP que se quiere atacar, se ha centrado en la IP específica para capturar el tráfico de una forma más clara y concisa.

ANEXO A.3 (COMANDOS DE TSTAT)

Comandos pertenecientes a la herramienta Tstat necesarios para realizar la captura de los datos:

Tstat –i eht0 –l–N net.private

- **Tstat** → comando usado para ejecutar dicha herramienta.

- **i eth0** → interfaz en la cual se quiere capturar el tráfico. En este caso la interfaz usada por defecto en las máquinas virtuales para conectarse entre ellas es eth0, pero en función de la configuración de red que se haya realizado previamente esta puede cambiar.
- **L** → comando específico a través del cual es capturado el tráfico en "vivo", es decir, el cual es generado desde el momento en que se ejecuta el comando hasta que se detiene, dentro de este tráfico se encuentra tanto tráfico no anómalo como datos generados por el ataque.

ANEXO A.4 (COMANDOS DE NPROBE)

Comandos pertenecientes a la herramienta Nprobe necesarios para realizar la captura de los datos:

Nprobe -i eth0

- **Nprobe** → comando usado para ejecutar el software.
- **-i eth0** →interfaz sobre la cual va a capturar los datos.

ANEXO A.5 INSTALACIÓN WINDOWS 8.1 EN VIRTUALBOX

Al igual que se ha realizado para las máquinas virtuales Kali-Linux y Metasploitable2, para la instalación de Windows 8.1 hay que seguir un proceso semejante.

Primeramente hay que descargar un disco virtual de Windows 8.1, se puede encontrar en diferentes fuentes de internet pero es recomendable descargarlo a través de la página oficial de Windows. Existen otras versiones de Windows que también pueden ser una opción para este trabajo, como por ejemplo Windows 10 o Windows XP, pero se encontraron algunas dificultades con los mismos así como el gran tamaño de la instalación como ocurría con Windows 10 o la incompabilidad que tiene Windows XP con algunas versiones de Virtual Box.

Una vez descargado, hay que seguir los pasos correspondientes indicados en el instalador para realizarlo de forma correcta como por ejemplo seleccionar el tipo de S.O. que se va usará (Windows 8.1) o el tamaño que queremos darle al disco.

Antes de encender la máquina virtual es necesario realizar las configuraciones específicas de red, las cuales están explicadas en el **Anexo A.6**.

ANEXO A.6 PUERTA DE ENLACE (CONFIGURACIÓN E INSTALACIÓN)

Lo primero que hay que hacer es configurar la red de las V.M. que están involucradas en el ataque ya sea como víctima (Windows 8.1) o atacante (Kali-Linux) [12].

Kali-Linux poseerá dos interfaces, la primera conectada a la WAN (la cual nos proporcionará acceso a internet) y la segunda conectada a una LAN (red interna), Windows 8.1 solo poseerá una interfaz en este caso de tipo LAN (debe ser la misma a la cual está conectada Kali-Linux, sino los datos no pasarán a través de ella y nunca podrán ser capturados, que a su vez proporcionará internet para poder generar todo el tráfico que es necesario capturar).

El siguiente paso es realizar una serie de configuraciones internas dentro de las V.M.

- Kali-Linux(SERVIDOR): estos son los siguientes ficheros a modificar y comandos a escribir:

  echo 1 > /proc/sys/net/ipv4/ip_fo

  - Reenviar conexiones de una tarjeta de red a la otra.

    sudo nano /etc/network/interfaces
    auto eth0 → primera interfaz
    iface eth0 inet dhcp

 # Configure the LAN port
auto eth1 → segunda interfaz
iface eth1 inet static
address 192.168.10.1 → IP asignada a la víctima
netmask 255.255.255.0

- Asignar las direcciones IP a cada una de las interfaces, configurando la eth0(WAN) como dhcp y la eth1 como dinámica(LAN), el siguiente paso es reiniciar los servicios de red de Kali-Linux con el siguiente comando:

  sudo /./etc/init.d/networking restart

- Windows 8.1(CLIENTE):

  - En el cliente se asigna una dirección IP, es necesario que se encuentre en el mismo rango que las escritas anteriormente en el servidor.

Una vez finalizados los pasos anteriores desde la máquina cliente debería poder hacer ping tanto a la misma máquina, al atacante y al router, este último se puede fallar en algunas situaciones debido a una configuración de red y el IPtables, para solucionarlo, hay que escribir una serie de comandos en la terminal Linux:

iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -m state –state RELATED,ESTABLISHED -j ACCEPT

ANEXO A.7 ALGORITMO DISTINCIÓN DE CARACTERÍSTICAS

A continuación se muestra el código completo del algoritmo de distinción de características:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

df = pd.read_csv("C:/Users/Alvaro/Desktop/Uni
segundo/Cuarto/TFG/Anomaly Detection using Gaussian
Distribution/Capturas/Tstat/BlasOtero/Mix/mixAtaquehttpblood/grafica
HttpBlood.csv", sep =",")

features = [f for f in list(df)
            if f not in ["Class"]]
plt.figure(figsize=(30,67*2))
gs = gridspec.GridSpec(60, 15)
gs.update(hspace=0.9)
gs.update(bottom=-10.95)
gs.update(top=0.95)

for i, f in enumerate(features):
    ax = plt.subplot(gs[i])
    sns.distplot(df[f][df.Class == 1])
    sns.distplot(df[f][df.Class == 0])
    ax.set_xlabel('')
    ax.set_title(' ' + str(f))
```

```
plt.show()
```

Este tipo de algoritmo posee la misma estructura que los otros explicados durante el proyecto, pero en este caso es mucho más sencillo.

Después de seleccionar una serie de librerías necesarias para su funcionamiento, hay que cargar el fichero con todas las transacciones (tanto legítimas como maliciosas) en una misma variable.

Posteriormente gracias a la feature añadida llamada "Class", se puede distinguir entre las transacciones anómalas y legítimas para así poder imprimirlas por pantalla con las gráficas que genera este algoritmo como se podrá comprobar en los siguientes Anexos.

ANEXO A.8 ALGORITMO DE ELIMINACIÓN DE CARACTERÍSTICAS INNECESARIAS

Es un algoritmo predictivo que usa la técnica de "**bagging**" para combinar diferentes árboles, siendo cada una de estas generadas gracias diferentes variables y observaciones.

Su código es el siguiente:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

df = pd.read_csv("C:/Users/Alvaro/Desktop/Uni
segundo/Cuarto/TFG/Anomaly Detection using Gaussian
Distribution/Capturas/Tstat/BlasOtero/Mix/mixDefinitivoMio.csv", sep
=",")

features = [f for f in list(df)
            if f not in ["Class"]]

rnd_clf = RandomForestClassifier(n_estimators = 100,
                                 criterion = 'entropy',
                                 random_state = 0)

rnd_clf.fit(df.iloc[:,1:121],df.iloc[:,122])

x, y = (list(x) for x in zip(*sorted(zip(rnd_clf.feature_importances_,
df.iloc[:,1:121].columns), reverse = True)))

for xe, ye in zip(x, y):
    print(ye, "-", xe)
```

Una vez ejecutado el algoritmo a través de la terminal de Windows 10 se puede observar un resultado como este:

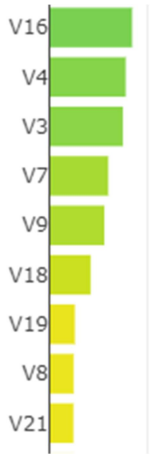


Ilustración 39. Importancia de características.

Gracias a esta gráfica se puede comprobar que las características (de un ejemplo en concreto) V21, V8 y V19 poseen muy poca importancia y son irrelevantes, por lo que hay que eliminarlas.

La función es la que aparece a continuación, siendo un meta estimulador que se ajusta a varios clasificadores de árbol de decisión en varias sub-muestras del conjunto de datos y utiliza el promedio para mejorar la precisión predictiva y controlar el sobreajuste.

RandomForestClassifier (n_estimators = 100, criterion = 'entropy', random_state = 0)

Este tipo de algoritmos están basados en una técnica llamada **Bagging,** que consiste en crear diferentes modelos usando muestras aleatorias con reemplazo y luego combinar o ensamblar los resultados, siguiendo estos pasos:

- Divide el set de Entrenamiento en distintos sub set de datos, obteniendo como resultado diferentes muestras aleatorias con las siguientes características: muestra uniforme y muestras con reemplazo.
- Posteriormente se crea un modelo predictivo con cada set, obteniendo modelos diferentes.
- Finalmente se construye o ensambla un único modelo predictivo, que es el promedio de todos los modelos.
- Selecciona individuos al azar, con el método muestreo con reemplazo, para crear diferentes sets de datos.

- Con cada set de datos que poseemos creamos un árbol de decisión, gracias a esto obtenemos una gran cantidad de árboles ya que con cada set es posible encontrar diferentes individuos y a su vez diferentes variables en cada nodo.
- Al crear los árboles se eligen variables al azar en cada nodo del árbol, dejando crecer el árbol en profundidad (es decir, sin podar).
- Predice los nuevos datos usando el "voto mayoritario", donde clasificará como "positivo" si la mayoría de los arboles predicen la observación como positiva.

ANEXO A.9 EXPLICACIÓN Y OPTIMIZACIÓN DE CARÁCTERÍSTICAS (FUERZA BRUTA)

Aquí tenemos el resultado obtenido al ejecutar el algoritmo explicado en el Anexo A.7 sobre el conjunto de datos generado en fase de **entorno virtualizado** perteneciente al ataque de **Fuerza Bruta**:

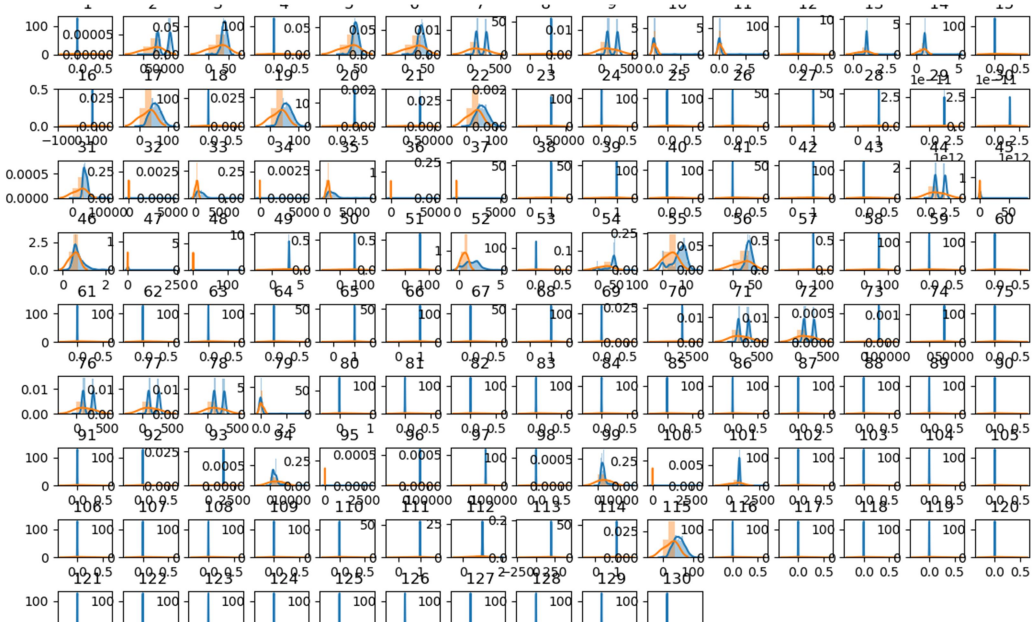


Ilustración 40. Características Fuerza Bruta (Sin optimizar)

Las características cuyas gráficas (azul y naranja) sigan un patrón parecido o idénticas serán eliminadas gracias a una función llamada *drop*, gracias a esto no se tendrán en cuenta esas características. El código de la función es el siguiente:

```
train_df.drop(labels = ["Time","Amount"], axis = 1, inplace = True)
```

Ilustración 41. Función que elimina las características seleccionadas.

El objetivo de esta distinción es aumentar el porcentaje de probabilidad de detección que obtendrá nuestro algoritmo. El resultado una vez eliminadas las "features" innecesarias sería el siguiente:

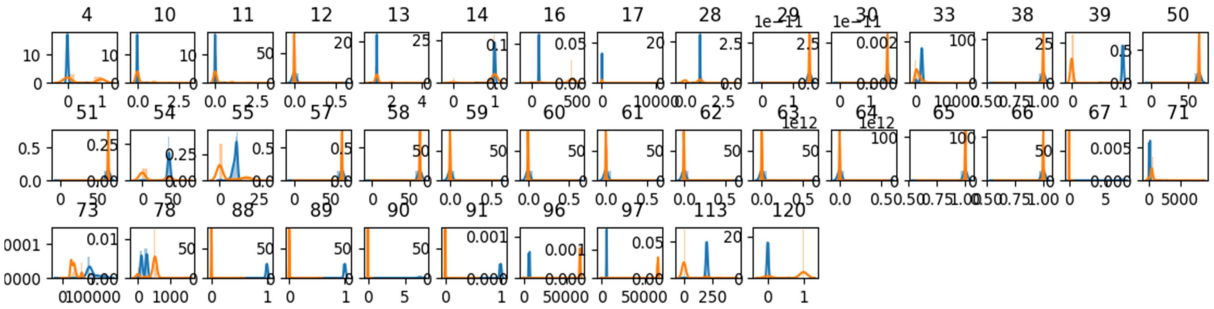


Ilustración 42. Características de Fuerza Bruta (optimizado)

Aquí están expuestas algunas de las características más importantes de los conjuntos con una breve explicación de las mismas:

- ***Número 44(http_t:44)*** → las conexiones http establecidas.

- ***Número 53-54-55(s_rtt_min:53,s_rtt_max:54,s_rtt_std:55)***→ RTT(Round-Trip delay Time), tiempo que tarda un paquete de datos envidado desde un emisor hasta que vuelve al mismo(sabiendo que ha llegado al destino(receptor)).De esta definición se puede deducir que rtt_min(tiempo mínimo), rtt_max(tiempo máximo) y rtt_std(desviación estándar).

- ***Número 71(c_mss_max:71)*** → El Tamaño Máximo de Segmento (Maximum Segment Size - MSS) es el tamaño más grande de datos, especificado en bytes, que un dispositivo de comunicaciones puede recibir en un único trozo, sin fragmentar.

- ***Número 88(s_f1323_opt:88)*** → incremento tamaño de la ventana de recepción permitido por el protocolo de control de transmisión por encima de su valor máximo anterior.

- ***Número 113(http_res:113)*** → Los códigos de estado de respuesta HTTP indican si se ha completado satisfactoriamente una solicitud HTTP específica. Las respuestas se agrupan en cinco clases: respuestas informativas, respuestas satisfactorias, redirecciones, errores de los clientes y errores de los servidores.

- ***Número 120(c_tls_sesid:120)*** → Es la seguridad de la capa de transporte (Transport Layer Security), un protocolo criptográfico que proporcionan comunicaciones seguras por internet, en este caso cifrando la ID de la sesión.

- ***Número 10(c_pkts_retx:10)*** → Número de paquetes de TCP que han sido retrasmitidos, excluyendo paquetes lanzados.

- ***Número 11(c_bytes_retx:11)*** → Número de bytes de TCP que han sido retrasmitidos, excluyendo paquetes lanzados.
- ***Numero 12(c_pkts_ooo:12)*** → (Out of delivery) número de paquetes que se han entregado en un orden diferente al que se enviaron.

- ***Número 57,58(s_ttl_min:57,s_ttl_max:58)*** →(Time To Live), concepto usado en redes de computadores para indicar por cuántos nodos puede pasar un paquete antes de ser descartado por la red o devuelto a su origen(ttl_max=número máximo de nodos) y (ttl_min= número mínimo de nodos).

- ***Número60/64(ed2k_data:60,ed2k_sig:61,ed2k_c2s:62,ed2k_c2c:63,ed2k_chat:64)*** → son hiperenlaces usados para localizar archivos dentro de redes P2P,y suelen hacer referencia a un archivo en concreto.

ANEXO A.10 EXPLICACIÓN Y OPTIMIZACIÓN DE CARÁCTERÍSTICAS (HTTP-FLOOD)

Aquí tenemos el resultado obtenido al ejecutar el algoritmo explicado en el Anexo A.7 sobre el conjunto de datos generados gracias al **conjunto KDD'99** perteneciente al ataque de **Http-Flood**:

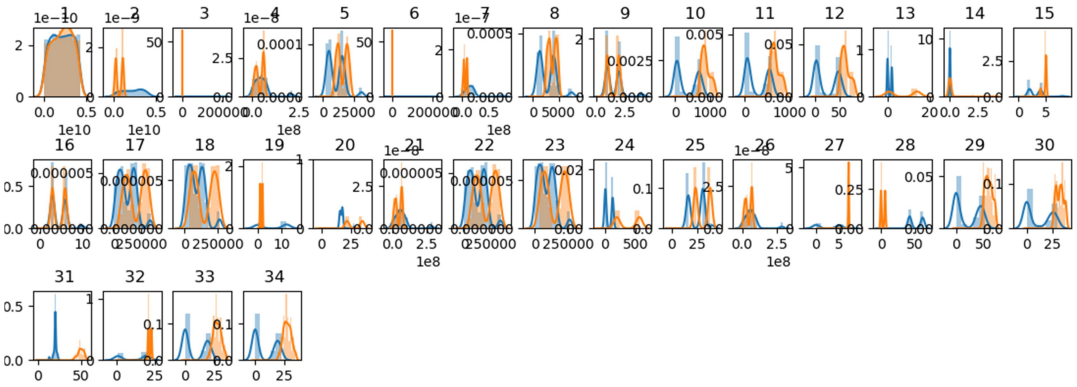


Ilustración 43. Características Http-Flood del KDD'99 (Sin optimizar).

El objetivo de esta distinción es aumentar el porcentaje de probabilidad de detección que obtendrá el algoritmo. El resultado una vez eliminadas las "features" innecesarias sería el siguiente:

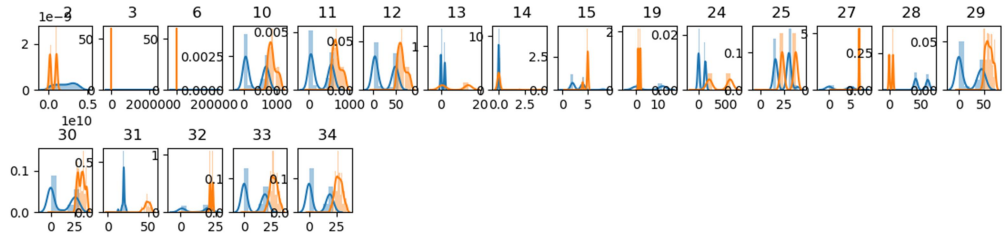


Ilustración 44. Características Http-Flood del KDD'99 (Optimizadas).

Aquí están expuestas algunas de las características más importantes de los conjuntos con una breve explicación de las mismas:

- **Duration** → Número de segundos de la conexión.
- **Protocol_type**→ Tipo de protocolo, TCP, UDP…
- **Src_Bytes** → Número de bytes de datos del origen al destino.
- **dst_bytes** → Número de bytes de datos del destino al origen.
- **flag** → Estado de la conexión.
- **num_failed_logins** → Número de intentos fallidos para realizar login.
- **root_shell** → 1 si root se ha obtenido, 0 si sucede lo contrario.
- **num_root** → Número de accesos de root.
- **num_file_creations** → Número de operaciones de creación de ficheros.
- **num_access_files** → Número de operaciones para acceder a control de ficheros.
- **Count** → Número de conexiones en los últimos dos segundos.
- **serror_rate** → % de conexiones que tiene error de tipo "SYN".

- **srv_count** → Número de conexiones al mismo servicio.
- **srv_diff_host_rate** → % de conexiones a diferentes hosts.

ANEXO A.11 EXPLICACIÓN Y OPTIMIZACIÓN DE CARÁCTERÍSTICAS (TCP-SYN)

Aquí tenemos el resultado obtenido al ejecutar el algoritmo explicado en el Anexo A.7 sobre el conjunto de datos generados gracias al **conjunto KDD'99** perteneciente al ataque de **TCP-SYN**:

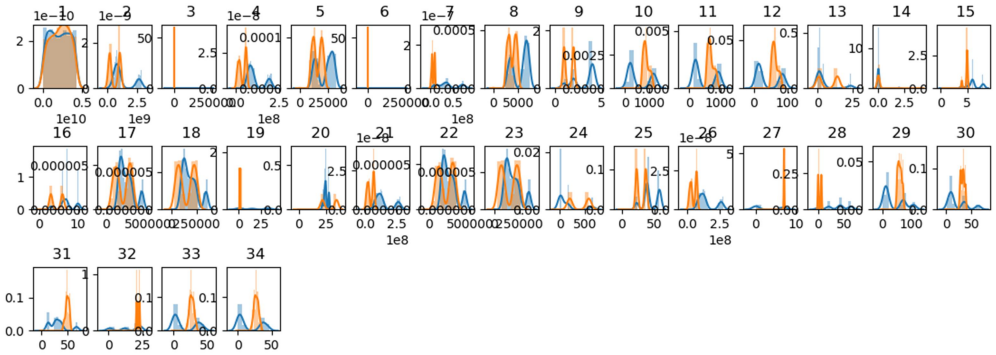


Ilustración 45. Características TCP-SYN del KDD'99 (Sin optimizar).

El resultado una vez eliminadas las "features" innecesarias es el siguiente:

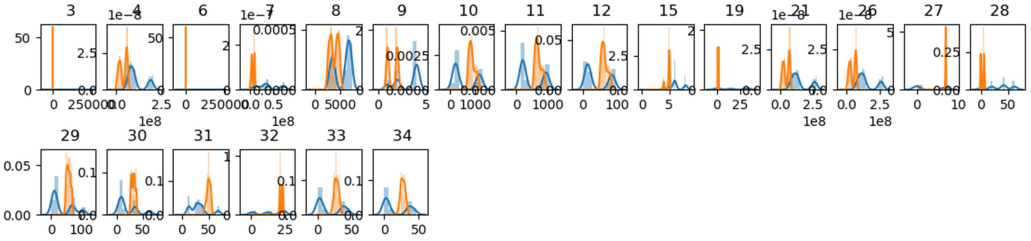


Ilustración 46. Características TCP-SYN del KDD'99 (Optimizadas).

Las características son las mismas explicadas y enumeradas en el Anexo A.10 correspondiente al ataque Http-Flood ya que ambos conjuntos pertenecen al KDD'99.

ANEXO A.12 EXPLICACIÓN Y OPTIMIZACIÓN DE CARACTERÍSTICAS KDD-NSL

Aquí están expuestas algunas de las características más importantes de los conjuntos con una breve explicación de las mismas, al ser un conjunto mejorado del KDD'99 posee muchas características en común, por lo que se expondrán aquellas que no se encuentren en el KDD'99:

- **Dst_host_count** → Número de conexiones que tienen la misma IP como destino.
- **Dst_host_srv_count** → Número de conexiones que tienen el mismo puerto.
- **Dst_host_same _srv_rate** → **%** de conexiones que estaban en el mismo servicio.

- **Dst_host_diff_ srv_rate → %** de conexiones que estaban en diferentes servicios.
- **Dst_host_same _src_port_rate →** % de conexiones que poseen el mismo Puerto origen.
- **Dst_host_srv_ diff_host_rate →** % de direcciones que poseen una máquina destino diferente.

FUERZA BRUTA

Aquí tenemos el resultado obtenido al ejecutar el algoritmo explicado en el Anexo A.7 sobre el conjunto de datos generados gracias al **conjunto NLS-KDD** perteneciente al ataque de **Fuerza Bruta**:

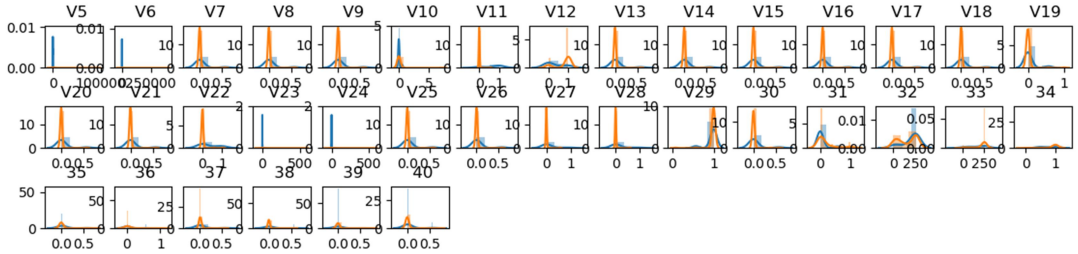


Ilustración 47. Características Fuerza Bruta del NSL-KDD (Sin optimizar).

El resultado una vez eliminadas las "features" innecesarias es el siguiente:

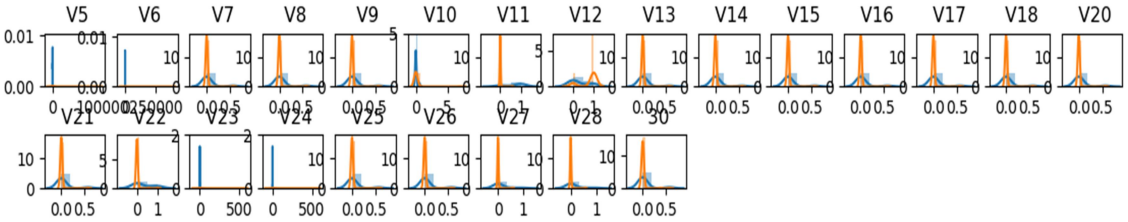


Ilustración 48. Características Fuerza Bruta del NSL-KDD (Optimizadas).

HTTP-FLOOD

Aquí tenemos el resultado obtenido al ejecutar el algoritmo explicado en el Anexo A.7 sobre el conjunto de datos generados gracias al **conjunto NLS-KDD** perteneciente al ataque de **Http-Flood**:

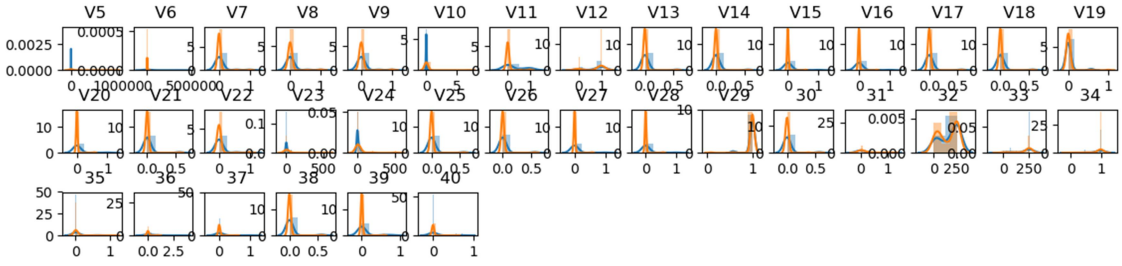


Ilustración 49. Características Fuerza Bruta del Http-Flood NSL-KDD (Sin optimizar).

El resultado una vez eliminadas las “features” innecesarias es el siguiente:

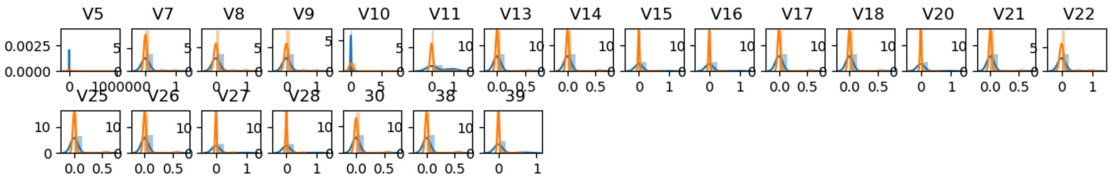


Ilustración 50. Características Http-Flood del NSL-KDD (Optimizadas).

TCP-SYN

Aquí tenemos el resultado obtenido al ejecutar el algoritmo explicado en el Anexo A.7 sobre el conjunto de datos generados gracias al **conjunto NLS-KDD** perteneciente al ataque de **TCP-SYN**:

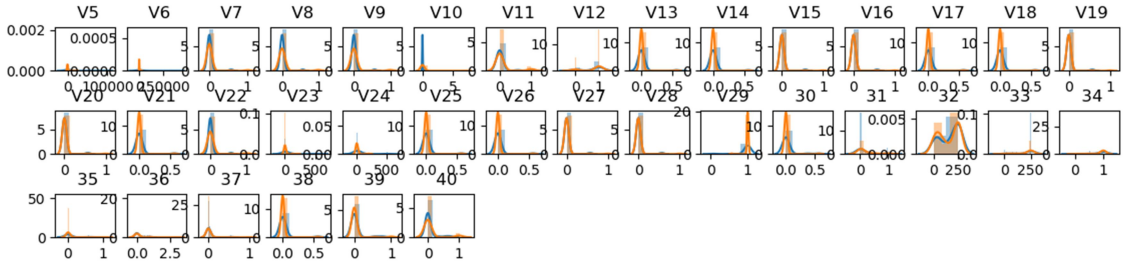


Ilustración 51. Características TCP-SYN del NSL-KDD (Sin optimizar).

El resultado una vez eliminadas las “features” innecesarias es el siguiente:

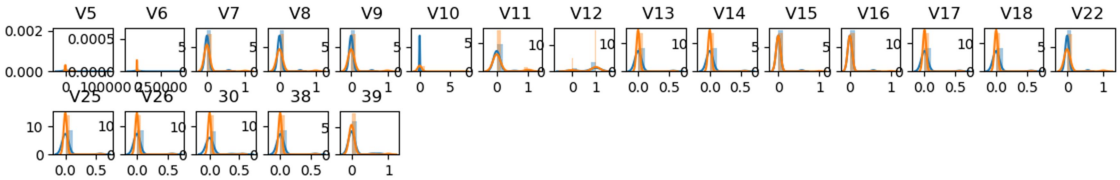


Ilustración 52. Características Fuerza Bruta del TCP-SYN (Optimizadas).

ANEXO A.13 CÓDIGO COMPLETO ALGORITMO DISTRIBUCIÓN GAUSSIANA

En este anexo se encuentra el código completo del algoritmo de **Distribución Gaussiana**:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import numpy as np
import random as rnd
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.gridspec as gridspec
from sklearn.preprocessing import StandardScaler
from numpy import genfromtxt
```

```
from scipy.stats import multivariate_normal
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score , average_precision_score
from sklearn.metrics import precision_score, precision_recall_curve
#%matplotlib inline
import plotly.graph_objs as go
#from 'react-plotly.js' import Plot
from plotly.offline import
download_plotlyjs,init_notebook_mode,plot,iplot
#init_notebook_mode(connected=True)
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier,
VotingClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from inspect import signature
from matplotlib  import  pyplot
from sklearn.metrics  import  auc
from  sklearn.metrics  import  roc_curve
from  sklearn.metrics  import  roc_auc_score
from sklearn.model_selection import GridSearchCV, cross_val_score,
StratifiedKFold, learning_curve
from sklearn.metrics import confusion_matrix


train_df = pd.read_csv("C:/Users/Alvaro/Desktop/Uni
segundo/Cuarto/TFG/Anomaly Detection using Gaussian
Distribution/Capturas/Tstat/BlasOtero/Mix/mixDefinitivoMioAdaptado.
csv", sep=",")
print(train_df.columns.values)
train_df.head(5)

def estimateGaussian(dataset):
    mu = np.mean(dataset, axis=0)
    sigma = np.cov(dataset.T)
    return mu, sigma

def multivariateGaussian(dataset,mu,sigma):
    p = multivariate_normal(mean=mu, cov=sigma)
    return p.pdf(dataset)

def selectThresholdByCV(probs,gt):
    best_epsilon = 0
```

```
    best_f1 = 0
    f = 0
    farray = []
    Recallarray = []
    Precisionarray = []
    #epsilon = (0.0000e+00, 1.0527717316e-72, 1.0527717316e-76,
1.0527717316e-15)
    epsilon = ( 1.0527717316e-72)
    #epsilon = np.asarray(epsilon)l
    predictions = p_cv < epsilon
    f = f1_score(train_cv_y, predictions, average = "binary")
    Recall = recall_score(train_cv_y, predictions, average = "binary")
    Precision = precision_score(train_cv_y, predictions, average =
"binary")
    farray.append(f)
    Recallarray.append(Recall)
    Precisionarray.append(Precision)
    print ('For below Epsilon')
    print(epsilon)
    print ('F1 score , Recall and Precision are as below')
    print ('Best F1 Score %f' %f)
    print ('Best Recall Score %f' %Recall)
    print ('Best Precision Score %f' %Precision)

    print ('-'*40)
    if f > best_f1:
        best_f1 = f
        best_recall = Recall
        best_precision = Precision
        best_epsilon = epsilon
    fig = plt.figure()
    ax = fig.add_axes([0.1, 0.5, 0.7, 0.3])
    #plt.subplot(3,1,1)
    plt.plot(farray ,"ro")
    plt.plot(farray)
    ax.set_xticks(range(5))
    #ax.set_xticklabels(epsilon,rotation = 60 ,fontsize = 'medium' )
    ax.set_ylim((0,1.0))
    ax.set_title('F1 score vs Epsilon value')
    ax.annotate('Best F1 Score', xy=(best_epsilon,best_f1),
xytext=(best_epsilon,best_f1))
    plt.xlabel("Epsilon value")
    plt.ylabel("F1 Score")
    plt.show()
    fig = plt.figure()
    ax = fig.add_axes([0.1, 0.5, 0.9, 0.3])
    #plt.subplot(3,1,2)
    plt.plot(Recallarray ,"ro")
```

```
    plt.plot(Recallarray)
    ax.set_xticks(range(5))
    #ax.set_xticklabels(epsilon,rotation = 60 ,fontsize = 'medium' )
    ax.set_ylim((0,1.0))
    ax.set_title('Recall vs Epsilon value')
    ax.annotate('Best Recall Score', xy=(best_epsilon,best_recall),
xytext=(best_epsilon,best_recall))
    plt.xlabel("Epsilon value")
    plt.ylabel("Recall Score")
    plt.show()

    fig = plt.figure()
    ax = fig.add_axes([0.1, 0.5, 0.9, 0.3])
    #plt.subplot(3,1,3)
    plt.plot(Precisionarray ,"ro")
    plt.plot(Precisionarray)
    ax.set_xticks(range(5))
    #ax.set_xticklabels(epsilon,rotation = 60 ,fontsize = 'medium' )
    ax.set_ylim((0,1.0))
    ax.set_title('Precision vs Epsilon value')
    ax.annotate('Best Precision Score', xy=(best_epsilon,best_precision),
xytext=(best_epsilon,best_precision))
    plt.xlabel("Epsilon value")
    plt.ylabel("Precision Score")
    plt.show()

    def plot_confusion_matrix(cm,
                              target_names,
                              title='Confusion matrix',
                              cmap=None,
                              normalize=True):

        import matplotlib.pyplot as plt
        import numpy as np
        import itertools

        if cmap is None:
            cmap = plt.get_cmap('Blues')

        plt.figure(figsize=(8, 6))
        plt.imshow(cm, interpolation='nearest', cmap=cmap)
        plt.title(title)
        plt.colorbar()

        #if target_names is not None:
            #tick_marks = np.arange(len(target_names))
            #plt.xticks(tick_marks, target_names, rotation=45)
            #plt.yticks(tick_marks, target_names)
```

```
        if normalize:
            cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

        thresh = cm.max() / 1.5 if normalize else cm.max() / 2
        for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
            if normalize:
                plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                         horizontalalignment="center",
                         color="white" if cm[i, j] > thresh else "black")
            else:
                plt.text(j, i, "{:,}".format(cm[i, j]),
                         horizontalalignment="center",
                         color="white" if cm[i, j] > thresh else "black")

        plt.tight_layout()
        plt.ylabel('Datos Anómalos                    Datos Normales')
        plt.show()

    plot_confusion_matrix(cm           = np.array(),
                          normalize    = True,
                          target_names = ['high', 'medium', 'low'],
                          title        = "Datos Normales            Datos Anómalos")

    Precision, Recall, thresholds = precision_recall_curve(train_cv_y,
predictions)
    average_precision = average_precision_score(train_cv_y,
predictions)
    auc_PR = auc(Recall ,Precision)
    step_kwargs = ({'step': 'post'}
    if 'step' in signature(plt.fill_between).parameters
       else {})
    plt.step(Recall, Precision, color='b', alpha=0.8,
         where='post')
    plt.fill_between(Recall, Precision, alpha=0.8, color='b',
**step_kwargs)
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('2-class Precision-Recall curve:
AP={0:0.2f}'.format(average_precision))
    plt.xlabel('2-class AUC precision-recall:
AUC_PR={0:0.2f}'.format(auc_PR))
    print ( 'auc=%.3f ap=%.3f'  %  (auc_PR , average_precision ) )
    # plot no skill
    pyplot . plot ( [ 0 ,   1 ] ,   [ 0.5 ,   0.5 ] ,   linestyle = '--' )
    # plot the precision-recall curve for the model
    pyplot . plot ( Recall ,   Precision ,   marker = '.' )
```

```
    # show the plot
    pyplot . show ( )


    auc_ROC_score  =  roc_auc_score ( train_cv_y, predictions)
    print ( 'AUC_ROC_Score: %.3f'  %  auc_ROC_score )
    # calculate roc curve
    fpr ,  tpr ,  thresholds  =  roc_curve ( train_cv_y, predictions)
    auc_ROC  =  auc ( fpr, tpr)
    print ( 'AUC_ROC: %.3f'  %  auc_ROC )
    # plot no skill
    pyplot.title('Receiver Operating Characteristic')
    pyplot . plot ( [ 0 ,  1 ] ,  [ 0 ,  1 ] ,  linestyle = '--' )
    # plot the roc curve for the model
    pyplot . plot ( fpr ,  tpr ,  marker = '.' )
    # show the plot
    pyplot.ylabel('True Positive Rate')
    pyplot.xlabel('False Positive Rate')
    pyplot . show ( )
    return best_f1, best_epsilon

train_df.drop(['V19','V21','V2','V14','V7','V9','V3','V6','V5','V27','V2
6','V25','V24','V23','V22','V20','V15','V8','V16','V17','V18',], axis =1,
inplace = True)
train_df.drop(labels = ["Time","Amount"], axis = 1, inplace = True)

train_strip_v1 = train_df[train_df["Class"] == 1]
train_strip_v0 = train_df[train_df["Class"] == 0]

Normal_len = len (train_strip_v0)
Anomolous_len = len (train_strip_v1)

start_mid = Anomolous_len // 2
start_midway = start_mid + 1

train_cv_v1  = train_strip_v1 [: start_mid]
train_test_v1 = train_strip_v1 [start_midway:Anomolous_len]

start_mid = (Normal_len * 60) // 100
start_midway = start_mid + 1

cv_mid = (Normal_len * 80) // 100
cv_midway = cv_mid + 1

train_fraud = train_strip_v0 [:start_mid]
train_cv    = train_strip_v0 [start_midway:cv_mid]
train_test  = train_strip_v0 [cv_midway:Normal_len]
```

```
train_cv = pd.concat([train_cv,train_cv_v1],axis=0)
train_test = pd.concat([train_test,train_test_v1],axis=0)

print(train_fraud.columns.values)
print(train_cv.columns.values)
print(train_test.columns.values)

train_cv_y = train_cv["Class"]
train_test_y = train_test["Class"]

train_cv.drop(labels = ["Class"], axis = 1, inplace = True)
train_fraud.drop(labels = ["Class"], axis = 1, inplace = True)
train_test.drop(labels = ["Class"], axis = 1, inplace = True)

mu, sigma = estimateGaussian(train_fraud)
p = multivariateGaussian(train_fraud,mu,sigma)
p_cv = multivariateGaussian(train_cv,mu,sigma)
p_test = multivariateGaussian(train_test,mu,sigma)

fscore, ep= selectThresholdByCV(p_cv,train_cv_y)
```

ANEXO A.14 CÓDIGO COMPLETO ALGORITMO ISOLATION FOREST

En este anexo se encuentra el código completo del algoritmo de **Isolation Forest**:

```
# importing libaries ----
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import  LabelEncoder
from pylab import savefig
from sklearn import model_selection
from sklearn.ensemble import IsolationForest
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC

X = pd.read_csv("C:/Users/Alvaro/Desktop/Uni
segundo/Cuarto/TFG/Anomaly Detection using Gaussian
Distribution/Bueno/Pruebas/IsolationForest/isolationForestTestTraini
ng.csv", sep=",")
#X.drop(labels =
["Time","V15","116","117","127","128","Amount"], axis = 1,
inplace = True)
X.drop(labels =
["Time","V15","116","117","127","128","Amount"], axis = 1,
inplace = True)
#X = np.r_[X + 3, X]
#X = X*0.01
```

```
X = pd.DataFrame(X, columns =
["V2","V3","V4","V5","V6","V7","V8","V9","V10","V11","V12",
"V13","V14","V16","V17","V18","V19","V20","V21","V22","V23"
,"V24","V25","V26","V27","V28","V29","30","31","32","33","34"
,"35","36","37","38","39","40","41","42","43","44","45","46","4
7","48","49","50","51","52","53","54","55","56","57","58","59",
"60","61","62","63","64","65","66","67","68","69","70","71","72
","73","74","75","76","77","78","79","80","81","82","83","84","
85","86","87","88","89","90","91","92","93","94","95","96","97"
,"98","99","100","101","102","103","104","105","106","107","10
8","109","110","111","112","113","114","115","118","119","120"
,"121","122","123","124","125","126","129","Class"])

y = pd.read_csv("C:/Users/Alvaro/Desktop/Uni
segundo/Cuarto/TFG/Anomaly Detection using Gaussian
Distribution/Bueno/Pruebas/IsolationForest/isolationForestTest.csv",
sep=",")
y.drop(labels = ["Time","V15","116","117","127","128","Amount"],
axis = 1, inplace = True)
#y.drop(labels =
["Time","V2","V3","V4","V5","V6","V7","V8","V9","V10","V11"
,"V12","V13","V14","V15","V16","V17","V18","V19","V20","V21
","V22","V23","V24","V25","V26","V27","V28","V29","30","31",
"32","33","34","35","36","37","38","39","40","41","42","43","44
","45","46","47","48","49","50","51","52","53","54","55","56","
57","58","59","60","61","62","63","64","65","66","67","68","69"
,"70","71","72","73","74","75","76","77","78","79","80","81","8
2","83","84","85","86","87","88","89","90","91","92","93","94",
"95","96","97","98","99","100","101"], axis = 1, inplace = True)

#y= np.r_[y + 3, y]
y = pd.DataFrame(y, columns =
["V2","V3","V4","V5","V6","V7","V8","V9","V10","V11","V12",
"V13","V14","V16","V17","V18","V19","V20","V21","V22","V23"
,"V24","V25","V26","V27","V28","V29","30","31","32","33","34"
,"35","36","37","38","39","40","41","42","43","44","45","46","4
7","48","49","50","51","52","53","54","55","56","57","58","59",
"60","61","62","63","64","65","66","67","68","69","70","71","72
","73","74","75","76","77","78","79","80","81","82","83","84","
85","86","87","88","89","90","91","92","93","94","95","96","97"
,"98","99","100","101","102","103","104","105","106","107","10
8","109","110","111","112","113","114","115","118","119","120"
,"121","122","123","124","125","126","129","Class"])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7)

X_outliers = pd.read_csv("C:/Users/Alvaro/Desktop/Uni
segundo/Cuarto/TFG/Anomaly Detection using Gaussian
Distribution/Bueno/Pruebas/IsolationForest/isolationForestTestAnoma
lo.csv", sep=",")
X_outliers.drop(labels = ["Time","V15","116","117","127","128",
"Amount"], axis = 1, inplace = True)
X_outliers= np.r_[X_outliers + 3, X_outliers]
X_outliers = X_outliers*0.2
```

```
X_outliers = pd.DataFrame(X_outliers, columns =
["V2","V3","V4","V5","V6","V7","V8","V9","V10","V11","V12",
"V13","V14","V16","V17","V18","V19","V20","V21","V22","V23"
,"V24","V25","V26","V27","V28","V29","30","31","32","33","34"
,"35","36","37","38","39","40","41","42","43","44","45","46","4
7","48","49","50","51","52","53","54","55","56","57","58","59",
"60","61","62","63","64","65","66","67","68","69","70","71","72
","73","74","75","76","77","78","79","80","81","82","83","84","
85","86","87","88","89","90","91","92","93","94","95","96","97"
,"98","99","100","101","102","103","104","105","106","107","10
8","109","110","111","112","113","114","115","118","119","120"
,"121","122","123","124","125","126","129","Class"])

# Isolation Forest ----
# training the model
clf = IsolationForest(behaviour='old', bootstrap=False, n_jobs=None,
max_features=1.0)

param_grid = {'n_estimators': list(range(100, 200, 400, 800, 1600 )),
          'max_samples': list(range(100, 200, 400, 800, 1600)),
          'contamination': [0.001, 0.01, 0.1, 0.2, 0.5],}

f1sc = make_scorer(f1_score(average='micro'))

grid_dt_estimator = model_selection.GridSearchCV(clf,
                                    param_grid,
                                    scoring=f1sc,
                                    refit=True,
                                    cv=10,
                                    return_train_score=True)
grid_dt_estimator.fit(X_train, y_train)
#clf.fit(X_train)

print("Best parameters set found on development set:")
print()
print(grid_dt_estimator.best_params_)
print()
print('Best CV Score:')
print()
print(grid_dt_estimator.best_score_)
#clf.fit(X_train)

# predictions
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_outliers = clf.predict(X_outliers)

xx, yy = np.meshgrid(np.linspace(-5, 5, 50), np.linspace(-5, 5, 50))
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.title("IsolationForest")
plt.contourf(xx, yy, Z, cmap=plt.cm.Blues_r)

b1 = plt.scatter(X_train[:, 0], X_train[:, 1], c='white',
```

```
s=20, edgecolor='k')
b2 = plt.scatter(X_test[:, 0], X_test[:, 1], c='green',
s=20, edgecolor='k')
c = plt.scatter(X_outliers[:, 0], X_outliers[:, 1], c='red',
s=20, edgecolor='k')
plt.axis('tight')
plt.xlim((-5, 5))
plt.ylim((-5, 5))
plt.legend([b1, b2, c],
["training observations",
"new regular observations", "new abnormal observations"],
loc="upper left")
plt.show()

def plot_confusion_matrix(cm,
target_names,
title='Confusion matrix',
cmap=None,
normalize=True):

import matplotlib.pyplot as plt
import numpy as np
import itertools

if cmap is None:
cmap = plt.get_cmap('Blues')

plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
plt.colorbar()

#if target_names is not None:
#tick_marks = np.arange(len(target_names))
#plt.xticks(tick_marks, target_names, rotation=45)
#plt.yticks(tick_marks, target_names)

if normalize:
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

thresh = cm.max() / 1.5 if normalize else cm.max() / 2
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
if normalize:
plt.text(j, i, "{:0.4f}".format(cm[i, j]),
horizontalalignment="center",
color="white" if cm[i, j] > thresh else "black")
else:
plt.text(j, i, "{:,}".format(cm[i, j]),
horizontalalignment="center",
color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('Datos Anómalos                    Datos Normales')
plt.show()
```

```
plot_confusion_matrix(cm           = np.array([]),
normalize    = True,
target_names = ['high', 'medium', 'low'],
title        = "Datos Normales              Datos Anómalos")

print (“Recall:” , recall_score(y_train, y_predict_train, average =
"binary")
print (“Precision:” , precision_score(y_train, y_predict_train, average
= "binary")
print (“F1-Score:” , f1_score(y_train, y_predict_train, average =
"binary")
```