Detección de anomalías enfocado a networking para redes inteligentes.

Trabajo Final de Máster

Integrantes:

José Roberto García Guevara
Joaquim Carlos René Martin
Gabriel Fernández Mora
Nibaldo Rojas Mancilla

2022, Abril

RESUMEN.

Este trabajo, de fin de proyecto de Máster, se centra en el desarrollo de un modelo de detección de anomalías en redes de computadores, utilizando algoritmos de machine learning. Para lograr lo anterior, se presenta, desde una perspectiva conceptual, los diferentes tipos de ataques, que sufren las redes de computadores junto a los enfoques de detección de anomalías utilizados para contrarrestar dichos ataques. La perspectiva presentada permite alcanzar una mejor comprensión del ámbito práctico en el cual el producto logrado con el proyecto puede ser de utilidad.

Posteriormente se da un marco de referencia de trabajo, presentando las técnicas disponibles en el ámbito de la inteligencia artificial y de los diferentes algoritmos que se tienen disponibles para desarrollar el modelo de detección.

Avanzando a una perspectiva más práctica, se describe el conjunto de datos proporcionado por la empresa junto a sus correspondientes atributos. En esta parte se hace una descripción del análisis de los datos y el proceso de diseño de características (features) que se llevó a cabo para lograr validar si se dispone de un set de datos que permita desarrollar el modelo de predicción, junto a las conclusiones obtenidas del preprocesamiento de los datos.

A partir de la información obtenida en la etapa anterior se focalizará el trabajo en determinar el modelo más adecuado, para ello se consideran diferentes modelos de aprendizaje automático y se aplican métricas de desempeño a cada modelo. El modelo que mejor se ajuste a las métricas esperadas se seleccionará como el recomendado.

Finalmente, en la conclusión se presentarán los resultados finales obtenidos y se dejarán abiertas otras propuestas que permitan complementar el modelo desarrollado.

TABLA DE CONTENIDO

1. INTRODUCCIÓN.

1.1 CONTEXTO

Los datos que generan uno o más procesos de un sistema, en general, pueden describir la actividad total del sistema o bien, especificar el comportamiento de partes específicas del funcionamiento del sistema. Cuando en el sistema se produce algún comportamiento inusual en algunos de sus procesos, los valores de los datos que se generan a partir de esos procesos reflejarán tal situación. Entonces, se puede definir que un valor atípico contiene información útil sobre eventos inusuales del sistema que afecta el proceso de generación de datos. La detección de valores atípicos puede proporcionar información muy útil para el reconocimiento de eventos inusuales y la búsqueda de soluciones, si es el caso. Su aplicabilidad se da en un gran número de dominios. Por ejemplo, en el dominio de los datos de salud, las técnicas de detección de valores atípicos se aplican ampliamente en los registros médicos para detectar patrones inusuales, los cuales podrían ser síntomas de nuevas enfermedades. En el contexto de las transacciones de tarjetas bancarias, detectar valores atípicos en los datos que representan transacciones bancarias, podrían indicar un robo o un mal uso de las tarjetas de crédito o débitos de los clientes. La aparición de una lectura diferente en el tablero de una aeronave podría significar una falla en algunas de sus piezas. En el ámbito de las redes de computadores, detectar un patrón de tráfico inusual podría significar que una computadora está intervenida y está enviando datos confidenciales a un destino no localizado, dando lugar a la detección de intrusiones.

1.2 OBJETIVOS

Este trabajo tiene como objetivo principal brindar una solución a la empresa T para el análisis de anomalías en su red:

1. Selección de algoritmos candidatos de machine learning enfocándonos a un número limitado de ataques y crear una prueba de concepto
2. Identificar si el set de datos de la empresa cumple con las características que permita desarrollar el modelo

3. En caso de no, buscar un complemento de datos para sacar provecho máximo de los algoritmos que vamos a utilizar
4. Llevar a cabo la configuración e implementación de los algoritmos seleccionados
5. Evaluar los resultados obtenidos para elegir el modelo más óptimo
6. Presentar recomendaciones para el despliegue de la solución ofrecida a la empresa

2. ESTADO DEL ARTE.

2.1.- ATAQUES A REDES DE COMPUTADORES

A medida que nos volvemos dependientes de las redes de computadoras y de internet para realizar una proporción cada vez mayor de nuestro trabajo, entretenimiento y vida social, el impacto de afectar estos sistemas aumenta proporcionalmente, atrayendo a un grupo cada vez mayor de atacantes que esperan ganar dinero o simplemente demostrar sus habilidades computacionales al efectuar ataques a la red. Un ataque a una red de ordenadores consiste en ejecutar un conjunto de actividades maliciosas para comprometer la integridad, confidencialidad o disponibilidad de los datos y servicios que otorga la red. Actualmente, las empresas de toda índole e instituciones de gobierno tienen una gran dependencia de sus redes de computadores y de la información que por ella fluye, para realizar sus actividades del día a día, por lo mismo es de gran importancia la capacidad que tienen de proteger sus sistemas de ataques de los que puedan ser víctima. Existen numerosos tipos de ataques que enfrenta una red, el conocer el tipo de comportamiento del ataque permite identificar el ataque y disponer de métodos para dar una respuesta apropiada y evitar las consecuencias de un ataque exitoso. Las consecuencias de un ataque exitoso se pueden traducir en la paralización o degradación del funcionamiento de la red y/o, la utilización o modificación no autorizada de datos que por ella fluyen, haciendo que los usuarios cuestionan no solo la solidez de la red informática, sino que también la confiabilidad de la empresa o institución pública que utiliza la red. Por lo mismo, es importante tener una buena comprensión de las diferentes formas que pueden tomar las amenazas para afectar la confiabilidad del sistema informático.

Tomando como referencia el trabajo de Hansman y Hunt [1], es posible presentar un conjunto de tipos de ataques conocidos, esta lista en ningún caso presenta todos los posibles tipos de ataques existentes.

Algunos de los ataques más conocidos son: Virus, Gusano, Troyano, Spyware, Adware, Ransomware, Rootkit, Backdoor, Bot, Botnet, Exploit, Probing, IPSweep,

PortSweep, Scanning, Sniffing, Keylogger, Spam, Login attack, Account takeover ATO, Phishing, Spear phishing, ingeniería social, Discurso incendiario, Denegación de servicio (DoS) y denegación de servicio distribuida (DDoS), Amenazas persistentes avanzadas (APT), Vulnerabilidad de día cero, Buffer Overflow, Amenazas internas, Amenazas polimórficas, R2L y U2R. En la parte de Glosario se describen con un mayor nivel de detalle cada uno de los ataques antes mencionados.

2.2.- DETECCIÓN DE ANOMALÍAS

En el contexto de la seguridad de la red de computadores y sus datos, una herramienta importante para intentar controlar los ataques a la red es la detección de intrusiones. Una intrusión se define como uno más intentos de comprometer la confidencialidad, integridad, disponibilidad de los datos o de eludir los mecanismos de seguridad de una computadora o de una red de computadoras y de sus datos. Las intrusiones pueden ser causadas por atacantes que acceden a los sistemas desde Internet, por usuarios autorizados de los sistemas, que intentan obtener privilegios adicionales para los que no están autorizados y usuarios autorizados que hacen un mal uso de los privilegios que se les han otorgado. Un sistema de detección de intrusiones permite a las organizaciones proteger sus sistemas de ordenadores e informáticos de las amenazas que conlleva el aumento de la conectividad de red y la dependencia de sus sistemas de información.

Los sistemas de detección de anomalías buscan encontrar algo inusual, en un comportamiento o en alguna característica de una entidad. Un comportamiento se considera inusual si se ha desviado significativamente de un umbral establecido, en función de su historia pasada. Una característica de una entidad es inusual si es significativamente diferente de la misma característica de los otros miembros de su población. La detección de anomalías no se limita sólo al contexto de la seguridad. En un contexto más general, es cualquier método para encontrar eventos que no se ajustan a una expectativa. En las situaciones en las que la confiabilidad de un sistema es de importancia crítica, hacer uso de la detección de anomalías puede permitir identificar oportunamente señales de falla del sistema, generando alertas que se traducen en acciones de investigaciones o en la aplicación de medidas preventivas o correctivas. Un sistema de detección de anomalías de redes de computadores busca

identificar la aparición de comportamientos o características inusuales en el tráfico de la red, en los computadores de la red o en las aplicaciones soportadas por la plataforma tecnológica. Funciona bajo la hipótesis que los ataques son diferentes de la actividad "normal" o “legítima” de la red y, por lo tanto, pueden ser detectados por los sistemas que identifican estas diferencias. Los sistemas detectores de anomalías construyen perfiles que representan el comportamiento normal de los usuarios, hosts o conexiones de red. Estos perfiles se construyen a partir de fuentes de datos históricas, recopiladas durante el período de funcionamiento. Los detectores recopilan datos de eventos y utilizan una variedad de medidas para determinar cuándo la actividad monitoreada se desvía de la norma. Los enfoques utilizados para implementar la detección de anomalías incluyen:

Detección de umbrales

En la que ciertos atributos del comportamiento del usuario y del sistema se expresan en términos de recuentos, con algún nivel establecido como permisible. Tales atributos de comportamiento pueden incluir el número de archivos a los que accede un usuario en un período de tiempo determinado, el número de intentos fallidos de iniciar sesión en el sistema, la cantidad de CPU utilizada por un proceso, etc. Este nivel puede ser estático o heurístico (es decir, diseñado para cambiar con los valores reales observados a lo largo del tiempo)

Medidas estadísticas

Se usan medidas estadísticas tanto paramétricas, donde se supone que la distribución de los atributos perfilados se ajusta a un patrón particular, como no paramétrica, donde la distribución de los atributos perfilados se "aprende" a partir de un conjunto de valores históricos, observados a lo largo del tiempo.

Uso de técnicas de Machine Learning

Se puede usar técnicas de machine learning como redes neuronales, algoritmos genéticos y modelos del sistema inmunitario

Es importante también tener presente que una de las debilidades de los detectores de anomalías es que tienden a producir una gran cantidad de falsas alarmas, ya que los patrones normales de comportamiento del usuario y del sistema pueden variar enormemente.

Tipos de anomalías

Es posible analizar la ocurrencia de una anomalía bajo 3 miradas, no necesariamente independiente. Según Ahmed, Mahmood, and Hu (2016) [2], la anomalía es una muestra que no tiene propiedades bien definidas respecto de una de una muestra normal. Para que la anomalía sea entendida, las reglas que componen el concepto normal deben ser específicas y válidas.

Anomalía puntual: Si una muestra de datos en particular se ve diferente de todo el conjunto de datos con las propiedades que lleva, se denomina anomalía puntual. Por ejemplo, es una anomalía puntual que una persona que hace una baja cantidad de compras todos los días con una tarjeta de crédito haga una cantidad muy alta de compras en un día aleatorio.

Anomalía contextual: El comportamiento fuera de patrón de una muestra de datos depende de ciertas condiciones u ocurre bajo ciertas condiciones. Por ejemplo, es una anomalía contextual que una persona que hace una baja cantidad de compras todos los días con una tarjeta de crédito hace una cantidad muy alta de compras en los días festivos.

Anomalía colectiva: Si un montón de datos que consiste en datos similares tiene propiedades anormales de acuerdo con los datos normales, esto se denomina anomalía colectiva. Por ejemplo, es una anomalía colectiva el aumento en el gasto de las tarjetas de crédito en el día de los enamorados

Sistema de detección de intrusión (IDS)

Un intento de hacer un mal uso de un sistema se llama "intrusión". Una intrusión normalmente explota una vulnerabilidad y debe detectarse lo antes posible.

En el contexto de este proyecto un sistema de detección de anomalías es un sistema de detección de intrusión que se ocupa de la detección de tráfico malicioso en una red de computadoras. Los IDS se están convirtiendo en un medida estándar de seguridad en las redes de computadores según Scarfone y Mell (2007) [3].

A diferencia de un Firewall (FW), un IDS generalmente se encuentra dentro de la red para monitorear el tráfico interior, ver figura 2.2.1 .

figura 2.2.1

Uno puede considerar el uso de Firewall e IDS para proteger la red de manera más eficiente.

Tipos de IDS

Podemos dividir los IDS en función de su ubicación en la red o del método de detección utilizado en la red.

a) **Basados en la ubicación del IDS en la red**, podríamos distinguir IDS basados en red e IDS basados en host.

   a1) IDS basado en red (NIDS). Como se muestra en la figura, el IDS se ubica dentro de la red, donde todo el flujo puede ser monitoreado. Este IDS comprueba si hay actividades maliciosas al inspeccionar todos los paquetes que se mueven a través de la red como en la figura 2.2.2.

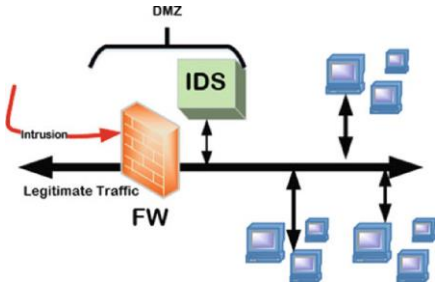


figura 2.2.2

a2) IDS Basados En Host (HIDS). Coloca el módulo IDS en cada cliente de la red. El IDS examina todo el tráfico que entra y sale del host cliente, hace un monitoreo detallado de cada cliente en particular (ver figura 2.2.3).

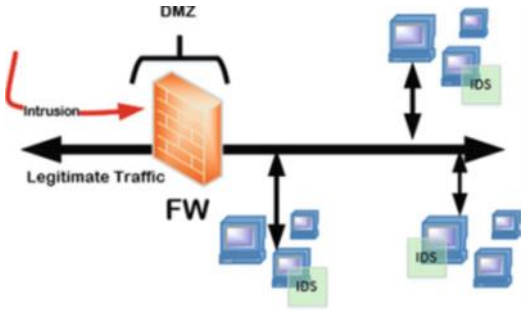


figura 2.2.3

b) **Basado en el método detección**, los IDS se pueden dividir en IDS basados en el uso indebido, en anomalía y en especificaciones.

b1) IDS basado en el uso indebido, conocido también como IDS IDS basado en firma [4], busca cualquier actividad maliciosa haciendo coincidir patrones de ataques conocidos, o firmas, con los tráficos monitoreados. Este IDS se adapta bien a detección de ataques conocidos; sin embargo, ataques nuevos o desconocidos (también llamados zeroday) son difíciles de detectar por él.

b2) IDS basado en anomalías, detecta un ataque mediante al obtener un perfil del comportamiento normal del tráfico y, a continuación, activa una alarma si hay alguna desviación de eso. La fortaleza de este IDS es su capacidad para la detección de ataques desconocidos. IDS basado en firmas generalmente

logra un mayor rendimiento de detección de ataques conocidos que IDS basado en anomalías.

b3) **IDS basado en especificaciones**, define manualmente un conjunto de reglas y restricciones para expresar las operaciones normales. Cualquier desviación de las reglas y las restricciones durante la ejecución se marcan como maliciosas.

El tipo de IDS que se considerará en este trabajo corresponderá a un IDS basados en anomalías, que se adapta muy bien al utilizar un enfoque de implementación de aprendizaje automático [4].

Medidas de rendimiento

Evaluar el desempeño de un IDS es un punto relevante, tanto para efecto de comparación con soluciones similares, como para justificar la selección del mismo. La evaluación del IDS se puede realizar utilizando un modelo común de medidas de rendimiento [5]**,** que considera cuatro criterios, a saber, Accuracy, Precision, F-measure y Recall. Todos estos criterios toman un valor entre 0 y 1. Cuando se acerca a 1, el rendimiento aumenta, mientras que cuando se acerca a 0, disminuye.

Accuracy: La relación entre los datos categorizados con éxito y los datos totales [6].

$$\text{Accuracy} = \frac{TN+TP}{FP+TN+TP+FN}$$

Recall (Sensibilidad): La relación entre los datos clasificados como un ataque y todos los datos de ataque [6].

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

Precisión: La relación entre los datos clasificados exitosos como el ataque y todos los datos clasificados como el ataque [6].

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

F-measure (F-score/F1-score): La media armónica de sensibilidad y precisión. Este concepto se utiliza para expresar el éxito general. por lo que, en este estudio, a la hora de analizar los resultados, se centrará, especialmente en el F1 score.

$$\text{F- measure} = \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$$

Para calcular estos cuatro elementos, se utilizan los cuatro valores que se resumen a continuación :

TP: True Positive: La data indica que la conexión es ataque y es ataque.
FP: False Positive: La data indica que la conexión es ataque y no es ataque.
FN: False Negative: La data indica que la conexión es normal y corresponde a un ataque.
TN: True Negative : La data indica que la conexión es normal y no hay ataque.
Estos valores se presentan visualizando la matriz de confusión.

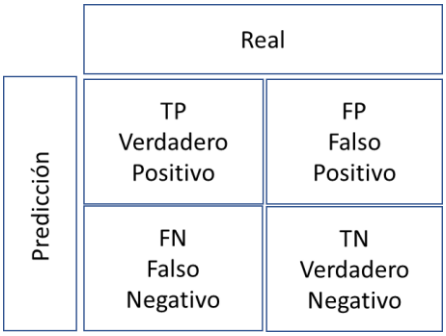


Figura XX. Matriz de confusión

Además de estas 4 medidas fundamentales, otras medidas a considerar en la selección del algoritmo corresponden al tiempo de procesamiento, los recursos de hardware y software que se requieren. Estos factores son importantes de tener en cuenta al momento de la selección de los algoritmos.

2.3.- DEFINICIÓN DE LA INTELIGENCIA ARTIFICIAL.

La Inteligencia Artificial la podemos definir como la capacidad de las máquinas de hacer tareas que simulan la inteligencia humana. Estas pueden ser funciones cognitivas como percibir, razonar, aprender, mejorar basado en la experiencia, interactuar, etc. Los objetivos principales de la IA incluyen la deducción y el razonamiento, la representación del conocimiento, la planificación, el procesamiento del lenguaje natural (NLP), el aprendizaje, la percepción y la capacidad de manipular y mover objetos. Los objetivos a largo plazo incluyen el logro de la Creatividad, la Inteligencia Social y la Inteligencia General (a nivel Humano). Dentro de la Inteligencia Artificial podemos encontrar diferentes ramas, como lo son el Machine Learning, Aprendizaje Supervisado, Aprendizaje no supervisado, aprendizaje por refuerzo, Deep Learning, entre otras. Las técnicas de la Inteligencia Artificial pueden ser aplicadas en una gran variedad de industrias y situaciones, como lo pueden ser la medicina, los autos autónomos, Bancos, Agricultura, Educación, entre otros.

Actualmente también ha surgido un debate ético alrededor de la Inteligencia Artificial. Algunos de los pensadores más importantes del planeta han establecido su preocupación sobre el progreso de la IA. Entre los problemas que puede traer aparejado el desarrollo de la Inteligencia Artificial, podemos encontrar los siguientes:

- Las personas podrían perder sus trabajos por la automatización.
- Las personas podrían tener demasiado (o muy poco) tiempo de ocio.
- Las personas podrían perder el sentido de ser únicos.
- Las personas podrían perder algunos de sus derechos privados.
- La utilización de los sistemas de IA podría llevar a la pérdida de responsabilidad.
- El éxito de la IA podría significar el fin de la raza humana.

Los principales lenguajes de programación que se utilizan en el desarrollo de la inteligencia artificial son Python y R. También se cuenta con una gran variedad de frameworks open source que nos facilitan el trabajar con los diferentes modelos, como lo pueden ser TensorFlow y PyTorch.

En el campo de la detección de anomalías podemos emplear la Inteligencia Artificial para identificar patrones anormales dentro de un conjunto de datos. Los sistemas definen la detección de anomalías como "un método utilizado para identificar patrones irregulares o inusuales en un entorno complejo". En otras palabras, la detección de anomalías detecta patrones de una manera que un humano no puede.

2.4.- DEFINICIÓN DEL MACHINE LEARNING APLICADO A LA DETECCIÓN DE ANOMALÍAS EN LA RED.

Con la detección de valores atípicos se busca descubrir datos que escapan al comportamiento de la mayoría de los datos, en un cierto dominio. La Estadística, a través de sus técnicas y herramientas es la rama que primero se preocupó en tratar el problema de la detección de valores atípicos y hacerlos visibles. Actualmente, el Machine Learning (ML), utiliza técnicas que permiten que la computadora “aprenda”, lo que ha facilitado, el tratar problemas de detección de valores que presentan anomalías en un contexto determinado y hacerlo visible. Las técnicas más utilizadas en ML son el aprendizaje supervisado y el aprendizaje no supervisado y la mezcla entre estas dos técnicas, el aprendizaje semi supervisado. En el aprendizaje supervisado, se tiene un conjunto de datos etiquetados, que representan observaciones pasadas sobre algún evento u objeto y lo que se busca es establecer alguna función, utilizando los datos etiquetados para entrenarla, que permita predecir un valor de salida para cualquier objeto nuevo de entrada. Cuando se tiene un set de datos que incluye datos de anomalías y todos los datos están con etiquetas, la detección de valores atípicos es supervisada y es un caso especial de un problema de clasificación. En el caso donde el set de datos carece de etiquetas y, por lo tanto, el modelo se entrena con datos sin etiquetar, da como resultado, una situación de aprendizaje no supervisado.

Entre los algoritmos que más se suelen utilizar en los problemas de Machine Learning particularmente a la hora de detectar anomalías en la red son los siguientes: PCA, Regresión Lineal, Regresión Logística, Árboles de Decisión, Random Forest, SVM o Máquinas de vectores de soporte, KNN o K vecinos más cercanos, K-means

2.5.- DEFINICIÓN DEL DEEP LEARNING APLICADO A LA DETECCIÓN DE ANOMALÍAS EN LA RED.

El Deep Learning o aprendizaje profundo es un subcampo dentro del Machine Learning, el cuál utiliza distintas estructuras de redes neuronales para lograr el aprendizaje de sucesivas capas de representaciones cada vez más significativas de los datos. El profundo o deep en Deep Learning hace referencia a la cantidad de capas de representaciones que se utilizan en el modelo; en general se suelen utilizar decenas o incluso cientos de capas de representación. las cuales aprenden automáticamente a medida que el modelo es entrenado con los datos.

Funcionamiento

El Deep Learning realiza un mapeo de entrada-a-objetivo por medio de una red neuronal artificial que está compuesta de un número grande de capas dispuestas en forma de jerarquía. La red aprende algo simple en la capa inicial de la jerarquía y luego envía esta información a la siguiente capa. La siguiente capa toma esta información simple, lo combina en algo que es un poco más complejo, y lo pasa a la tercera capa. Este proceso continúa de forma tal que cada capa de la jerarquía construye algo más complejo de la entrada que recibió de la capa anterior. De esta forma, la red irá aprendiendo por medio de la exposición a los datos de ejemplo.

La especificación de lo que cada capa hace a la entrada que recibe es almacenada en los pesos de la capa, que, en esencia, no son más que números. Utilizando terminología más técnica podemos decir que la transformación de datos que se produce en la capa es parametrizada por sus pesos. Para que la red aprenda debemos encontrar los pesos de todas las capas de forma tal que la red realice un mapeo perfecto entre los ejemplos de entrada con sus respectivas salidas objetivo. Pero el problema reside en que una red de Deep Learning puede tener millones de parámetros, por lo que encontrar el valor correcto de todos ellos puede ser una tarea realmente muy difícil, especialmente si la modificación del valor de uno de ellos afecta a todos los demás.

Para poder controlar algo, en primer lugar debemos poder observar. En este sentido, para controlar la salida de la red neuronal, deberíamos poder medir cuán lejos está la salida que obtuvimos de la que se esperaba obtener. Este es el trabajo de la función de pérdida de la red. Esta función toma las predicciones que realiza el modelo y los valores objetivos (lo que realmente esperamos que la red produzca), y calcula cuán lejos estamos de ese valor, de esta manera, podemos capturar que tan bien está funcionando el modelo para el ejemplo especificado. El truco fundamental del Deep Learning es utilizar el valor que nos devuelve esta función de pérdida para retroalimentar la red y ajustar los pesos en la dirección que vayan reduciendo la pérdida del modelo para cada ejemplo. Este ajuste, es el trabajo del optimizador, el cuál implementa la propagación hacia atrás.

Resumiendo, el funcionamiento sería el siguiente: inicialmente, los pesos de cada capa son asignados en forma aleatoria, por lo que la red simplemente implementa una serie de transformaciones aleatorias. En este primer paso, obviamente la salida del modelo dista bastante del ideal que deseamos obtener, por lo que el valor de la función de pérdida va a ser bastante alto. Pero a medida que la red va procesando nuevos casos, los pesos se van ajustando de forma tal de ir reduciendo cada vez más el valor de la función de pérdida. Este proceso es el que se conoce como entrenamiento de la red, el cual, repetido una suficiente cantidad de veces, generalmente 10 iteraciones de miles de ejemplos, logra que los pesos se ajusten a los que minimizan la función de pérdida. Una red que ha minimizado la pérdida es la que logra los resultados que mejor se ajustan a las salidas objetivo, es decir, que el modelo se encuentra entrenado.

Algunas de las arquitecturas más populares en las redes neuronales son:

- Redes neuronales prealimentadas
- Redes neuronales convolucionales
- Redes neuronales recurrentes

2.6.- ALGORITMOS USADOS PARA EL MODELO

Hemos seleccionado distintos algoritmos para construir el modelo. La razón detrás de esta selección es incorporar técnicas de aprendizaje de machine learning y deep learning. Los diferentes experimentos se llevan a cabo para medir y validar el rendimiento de estos algoritmos utilizando el conjunto de datos NSL-KDD [7]

La meta es hacer una detección binaria: ataque o no ataque. Para este tipo de trabajo, los modelos que se adaptan pueden ser regression logística. En NSL-KDD, el volumen de datos es muy grande así que debemos elegir modelos que funcionan bien con grandes volúmenes de datos como Random Forest, SVM y red neuronal.

Los algoritmos elegidos corresponden a:

Regresión Logística

Es un algoritmo básico de aprendizaje automático supervisado para todo tipo de problemas de clasificación. Encuentra la asociación entre variables dependientes (o objetivo) y conjuntos de variables (o características) independientes. Es un tipo de regresión lineal simple donde la variable dependiente o objetivo es categórica. Utiliza la función sigmoide en el resultado de predicción de la regresión lineal. [8] Se usa comúnmente para estimar la probabilidad de que una instancia pertenezca a una clase en particular (por ejemplo, ¿cuál es la probabilidad de que un correo electrónico sea spam?). Si la probabilidad estimada es mayor que el 50%, entonces el modelo predice que la instancia pertenece a esa clase, de lo contrario predice que no. Esto lo convierte en un clasificador binario.

También podemos utilizar el algoritmo de regresión logística para múltiples clases objetivo. Para problemas de clases múltiples, se denomina regresión logística multinomial. La regresión logística multinomial es una modificación de la regresión logística; utiliza la función softmax en lugar de la función de activación sigmoide. [8]

Árboles de decisión

El árbol de decisión es una de las técnicas de clasificación más conocidas. Se puede emplear para ambos tipos de problemas de aprendizaje supervisado: problemas de

clasificación y regresión. Es una estructura de árbol e imita el pensamiento a nivel humano, lo que hace que sea más fácil de entender e interpretar. Cada árbol de decisión consta de nodos (nodo raíz y subnodos), ramas y hojas. Dentro de cada nodo, hay una decisión a tomar. Según el resultado de esta decisión, el algoritmo elige una de las dos ramas en el paso siguiente (el número de ramas puede ser mayor que dos). La rama seleccionada lleva el algoritmo al siguiente nodo. Este proceso termina con el último elemento, la hoja. [9] Hay muchos algoritmos de árbol de decisión disponibles, por ejemplo, CART, ID3, C4.5 y CHAID.

Random Forest

Es un algoritmo de aprendizaje automático muy popular que utiliza árboles de decisión. En este método, se crea un "bosque" ensamblando un gran número de estructuras de árboles de decisión diferentes que se forman de diferentes maneras.(10) Se generan N árboles de decisión a partir de los datos del conjunto de entrenamiento. Durante este proceso, se vuelve a muestrear aleatoriamente el conjunto de entrenamiento para cada árbol. Luego, N árboles de decisión son obtenidos, cada uno de los cuales es diferente del otro. Finalmente, la clasificación se realiza votando por las estimaciones realizadas por los N árboles. El valor con la calificación más alta se determina como el valor final [10]

Ventajas de un Random Forest [11]

- Este algoritmo funciona bien con conjuntos de datos muy grandes y complicados.
- El problema de sobreajuste que se encuentra con frecuencia en los árboles de decisión es muy raro en este algoritmo.
- Se puede aplicar a muchos tipos de problemas de aprendizaje automático.
- Funciona bastante bien cuando existen valores faltantes en el conjunto de datos. Reemplaza estos valores perdidos con sus propios valores creados.
- Calcula y utiliza el nivel de importancia de las variables a la hora de realizar la clasificación.

También se utiliza para la selección de características en el aprendizaje automático. Una de las principales ventajas de un Random Forest es la velocidad, la clasificación ocurre muy rápidamente [12]

Máquinas Vector de Soporte

Una *máquina vector de soporte* (SVM) es un modelo de aprendizaje automático, capaz de realizar clasificación lineal o no lineal, regresión e incluso detección de valores atípicos. Es uno de los modelos más populares en Machine Learning. Los SVM son especialmente adecuados para la clasificación de conjuntos de datos complejos de tamaño pequeño o mediano.

El objetivo de la clasificación de vectores de apoyo es desarrollar una forma computacionalmente eficiente de aprender hiperplanos de separación "buenos" en un espacio de características de alta dimensión, donde por hiperplanos "buenos" se entienden aquellos que optimizan los límites de separación de los vectores de apoyos y por "computacionalmente eficientes" a algoritmos capaces de tratar con tamaños de muestra del orden de 100.000 instancias. La visión popular de SVM es que encuentra un hiperplano “óptimo” como la solución al problema de clasificación. [13]. La formulación más simple de SVM es la lineal [14].

Red Neuronal Artificial

La Red Neuronal Artificial (ANN) es un enfoque de proceso de información que trata de imitar el cerebro humano. Consiste en neuronas interconectadas que colaboran para realizar tareas. En general, un sistema de este tipo aprende a resolver tareas considerando ejemplos (datos de entrenamiento) sin usar ninguna regla específica de la tarea. La red neuronal profunda (DNN) es una red neuronal artificial con múltiples capas entre la capa de entrada y salida, que puede modelar relaciones no lineales complejas.

En la detección de intrusiones basadas en anomalías, la red neuronal profunda aprende a reconocer ataques (como ataque DDoS), analizando un conjunto lo suficientemente grande de ejemplos etiquetados, luego el modelo se utiliza para

identificar nuevas instancias de ataque como DDoS. [15] En este trabajo se utiliza una ANN con ReLU y optimizador Adam.

2.7.- EL CONJUNTO DE DATOS A UTILIZAR

El primer objeto de estudio para el buen desarrollo de este trabajo es el conjunto de datos, tener un buen conjunto de datos es un elemento fundamental para cumplir el objetivo de realizar el sistema de detección de intrusión basado en anomalías (SDA). El conjunto de datos que se requiere debe contener información en el contexto de la seguridad de las conexiones de una red de computadores, debe contener conexiones normales y conexiones anómalas, de tal manera que sea posible a través de un análisis identificar algún tipo de ataque a un servidor o a la integridad de una red de computadores, por ejemplo, un ataque de denegación de servicio en el contexto de las conexiones que afectan a la red de computadores. El gran problema de este tipo de conjuntos de datos es su escasez, rara vez se publican, ya que involucran información confidencial de la empresa, como la arquitectura de red, los mecanismos de seguridad, etc. Además, también está la dificultad del esfuerzo que conlleva etiquetar tráfico real de la red.

En nuestro caso, el conjunto de datos para trabajar en el diseño del SDA es entregado por la empresa que solicita el desarrollo del SDA. Respecto de esto, la empresa hizo una primera entrega de un conjunto de datos, con el con cual se dio inicio al trabajo de análisis exploratorio (etapa EDA), desde este análisis se obtuvieron las características de la data, no obstante, presentaba un problema de volumen, los datos eran muy pocos, un par de horas de registros. Se solicito a la empresa la entrega de un mayor volumen de datos, pero no hubo más entregas. Esta situación generó un inconveniente, toda vez que ya se había invertido tiempo en el análisis de la data entregada y se enfrentaba la situación que no se disponía de un conjunto de datos que permitiera continuar con el proyecto. La situación antes descrita obliga a buscar con premura una solución alternativa, con el fin de cumplir con el objetivo de desarrollar el SDA. El objetivo es encontrar un conjunto de datos que contenga anomalías en el contexto de la ciberseguridad en redes de computadores y que esté disponible públicamente, se logra encontrar inicialmente el conjunto de datos

KDDCup99 [16] y luego se encuentra otra versión, que deriva de KDDCup99 , el conjunto de datos denominado NSL-KDD.[17]

Conjunto de datos KDDCup99

El conjunto de datos KDDCup99 es el resultado del programa de evaluación de detección de intrusiones apoyado por DARPA, y administrado por MIT Lincoln Labs, contiene nueve semanas de datos obtenidos desde una red de área local (LAN) que simula una LAN típica de la Fuerza Aérea de los Estados Unidos. Esta LAN fue operada como si fuera parte del ambiente de red de la Fuerza Aérea, y fue sometida a múltiples ataques. Los datos de entrenamiento obtenidos alcanzaron aproximadamente cuatro gigabytes de datos TCP binarios en las primeras siete semanas de tráfico de red, generando cerca cinco millones de registros de conexión de 100 bytes cada uno. Los datos de prueba se construyeron a partir del tráfico de red en las últimas dos semanas, lo que arrojó alrededor de dos millones de registros de conexión.

Una de las deficiencias más importantes en el conjunto de datos KDD es la gran cantidad de registros redundantes, lo que hace que los algoritmos de aprendizaje estén sesgados hacia los registros frecuentes, y así evitar que aprendan registros poco frecuentes que suelen ser más dañinos para redes como los ataques U2R y R2L. Además, la existencia de estos registros repetidos en el conjunto de pruebas hace que los resultados de la evaluación estén sesgados por los métodos que tienen mejores tasas de detección en los registros frecuentes. [7]

Conjunto de datos NSL-KDD

NSL-KDD es un conjunto de datos de detección de intrusiones que contiene datos que describen tipos de conexiones normales o de ataques a una red. Este conjunto de datos es una versión revisada del conjunto de datos de detección de intrusiones KDDCup99.

Para el desarrollo del proyecto se opta por utilizar el conjunto de datos NSL-KDD, considerando que presenta mejoras respecto del conjunto de datos KDDCUP99. [18]

- No se incluye registros redundantes en el conjunto de entrenamientos, por lo que los clasificadores no estarán sesgados hacia los registros más frecuentes.
- No hay registros duplicados en los conjuntos de prueba propuestos; por lo tanto, el desempeño del modelo de aprendizaje 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 las diferentes técnicas de aprendizaje.
- El número de registros en el conjunto de entrenamiento y de prueba son razonables, lo que hace que sea asequible ejecutar los experimentos en el conjunto completo sin la necesidad de seleccionar aleatoriamente una pequeña porción. En consecuencia, los resultados de la evaluación de diferentes modelos serán consistentes y comparables.

Estructura del conjunto de datos NSL-KDD

En el conjunto de datos NSL-KDD cada registro representa una conexión de red TCP/IP con un total de 41 características cuantitativas y cualitativas, además cada registro tiene asociada una etiqueta, que lo identifica como un registro normal o un registro de ataque, este último identifica el tipo de ataque específico. [19]

Cada registro tiene 32 características numéricas, 3 características nominales y 6 categorías binarias. Algunas de las características se generan utilizando una ventana de tiempo de dos segundos.

Tipo	Característica *
Nominal (3)	protocol_type, service, flag.
Binario (6)	land, logged_in, root_shell, su_attempted, is_host_login, is_guest_login.
Numerico (32)	duration, src_bytes, dst_bytes, wrong_fragment, urgent, hot, num_failed_logins, num_compromised, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, count, srv_count,serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count,| dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, st_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate.

*Ver Glosario para una descripción de las características

Los conjuntos de entrenamiento y prueba contienen 22 y 39 ataques respectivamente. Es importante mencionar que el conjunto de datos de prueba incluye tipos de ataque que no están en el conjunto de datos de entrenamiento, lo que hace que la tarea de clasificación de los ataques sea más realista.

Los numerosos tipos de ataque en el conjunto de datos NSL-KDD se agrupan en cuatro clases de ataque, esto con el fin de mejorar la tasa de detección.

Clases de Ataques. - Los ataques se dividen en cuatro categorías:

- **Ataque de denegación de servicio (DoS):** el atacante envía múltiples requerimientos a la red, haciendo que los recursos y servicios estén ocupados o saturados para manejar las solicitudes legítimas, denegando o bloqueando a los usuarios legítimos las solicitudes de acceso a los recursos de la red. **Tipos de Ataques:** apache2, back, land, mailbomb, netpune, pod, processtable, smurf, teardrop, worm, udpstorm.

- **Ataque de usuario a root (U2R):** el atacante comienza con el acceso a una cuenta de usuario válida en el sistema (como: sniffing passwords), luego explota vulnerabilidades para obtener acceso de usuario root del sistema. **Tipos de Ataques:** buffer_overflow, loadmodule, perl, ps, rootkit, sqlattack, xterm.

- **Ataque remoto a local (R2L):** el atacante a través de una maquina remota envía paquetes a través de la red, explotando vulnerabilidades para obtener acceso a la maquina local como un usuario más, buscando alcanzar privilegios de usuario root para realizar actividades no autorizadas. **Tipos de Ataques:** ftp-write, httprunnel, guess_password, imap, multihop, named, phf, sendmail, snmpgetattack, snmpguess, spy, warezclient, warezmaster

- **Ataque Probe (sondeo) :** Es un ataque donde se analiza la red para recopilar información o encontrar vulnerabilidades conocidas con el fin de eludir sus controles de seguridad y lanzar un ataque a la red. **Tipos de Ataques:** xlock, xsnoop, satanipsweep, nmap, portsweep, mscan, saint.

Clases de registros. - Las características de los registros NSL-KDD se pueden clasificar en las siguientes clases:

- **Características básicas:** Se refiere a todos los atributos que se pueden extraer de los encabezados de los paquetes en la conexión TCP/IP. Tales características causan cierto retraso en el proceso de detección de ataques.

- **Características de tráfico basada en el tiempo:** Se refiere a las características que dependen del intervalo de la ventana de tiempo de 2 segundos. Un caso sería tener una extensa variedad de asociaciones con exactamente el mismo host durante el intervalo de 2 segundos. Se clasifican en dos tipos:

- **Características de tráfico al mismo host**: Conexiones en los pasados 2 segundos que pertenecen al mismo host de destino que la conexión actual. Calcula estadísticas que pertenecen a servicios, comportamientos de protocolo, etc.

- **Características de tráfico al mismo servicio**: Conexiones en los pasados 2 segundos que pertenecen al mismo servicio que la conexión actual.

- **Nota:** Existen ataques de probing que son lentos, utilizan tiempo mayores a 2 segundos en explorar un host o un puerto, por ejemplo, cada 30 segundos. Estos ataques no producen patrones de intrusión en una ventana de tiempo de 2 segundos. Para solucionar esto, la característica de tráfico al mismo host y la característica de tráfico al mismo servicio son recalculadas, pero basada en una ventana de conexión de 100 conexiones.

- **Características de contenido:** se refiere a las características utilizadas para buscar comportamientos sospechosos en la parte de dato del paquete TCP (como una serie de intentos de inicio de sesión fallidos), se utiliza la información del dominio y esto involucra aspectos que incorporan la amplia variedad de esfuerzos de inicio de sesión fallidos. Se puede usar específicamente en la detección de ataques R2L y U2R, donde estos ataques no tienen patrones secuenciales de intrusión, están incrustados en los datos de los paquetes y normalmente involucra una conexión individual.

3. ANÁLISIS DEL DATASET PROPORCIONADO POR LA EMPRESA.

3.1.- ANÁLISIS EXPLORATORIO

Con el fin de poder respaldar nuestras conclusiones y determinar la factibilidad de nuestro proyecto, vamos a proceder con la exploración de los datos suministrados por parte de la empresa, con el fin de analizar y determinar si con los datos suministrados por parte de la empresa es posible detectar anomalía o se requiere de una muestra más grande, con una mayor cantidad de variabilidad en los datos.

La empresa ha entregado un dataset que consta de datos en un intervalo de tiempo de dos horas consecutivas para un mismo día con datos no etiquetados y sin variable de tiempo.

3.2.- AMBIENTE Y COMPONENTES

Para nuestro análisis exploratorio inicial, vamos a hacer uso de Jupyter notebook, el cual nos va a facilitar la documentación, exploración y aplicación de técnicas basadas en lenguaje python. En este caso de estudio, el análisis, creación, pruebas de los modelos, se aplicarán sobre un contenedor docker en una máquina con las siguientes capacidades de hardware:

- Procesador AMD Ryzen 7 5800X 8 CPU con 16 núcleos a una velocidad de reloj de 3.8GHz hasta un máximo de 4.7 GHz.
- 40 GB de memoria ram
- Tarjeta gráfica Nvidia GEFORCE GTX 1650 de 4GB
- Ubuntu 20.04.3 LTS
- imagen Docker: tensorflow/tensorflow “latest-gpu-jupyter” optimizada para proceso en GPU.

Además, vamos a hacer uso de librerías o componentes de Python:

- Pandas: para la manipulación a nivel de filas y columnas y carga de los datos.
- Numpy: para el tratamiento matricial de los datos a nivel matemático.

- Matplotlib: para la visualización de los datos, gráficos.
- Sklearn: para aplicar técnicas de clusterización, escalado de datos y análisis de componentes principales.

3.3.- EXPLORACIÓN Y LIMPIEZA DE LOS DATOS

Para iniciar con el proceso se procede con la carga de los datos suministrado por la empresa en nuestro dataframe de pandas, para ello vamos a hacer uso de dos archivos .csv suministrados:

```
data_13h = pd.read_csv('data/2021-12-13-13H.csv')
data_14h = pd.read_csv('data/2021-12-13-14H.csv')
data = pd.concat([data_13h, data_14h])
```

Figura 3.3.1

Hacemos una exploración rápida de los primeros 5 registros cargados.

```
data.head(5)
```

	_time	level	application_id	application_name	direction	engine_id	host	host_category	wan_ip_country_code	lan_ip_country_code	...	wan_ip_as_name	wan_H_port	serial_number	saas_name	saas_family	host_l2	referer_l2	events	sum_bytes	sum_pkts
0	1639390860000	23f24570-d2c9-4e6e-a833-3d1734e110c8.abcc9097-...	6:3	PRIVATE ADDRESSING	upstream	6	teams.events.data.microsoft.com	cleaning	US	CN	...	ATT-INTERNET4	771	848/06464	NaN	NaN	microsoft.com	NaN	1	229	1
1	1639390860000	23f24570-d2c9-4e6e-a833-3d1734e110c8.abcc9097-...	6:3	PRIVATE ADDRESSING	upstream	6	ssl.gstatic.com	liste_blanche	US	US	...	AKAMAI-AS	54606	848/06464	Google portal and shared services	IT & Communication	gstatic.com	NaN	1	509	1
2	1639390860000	23f24570-d2c9-4e6e-a833-3d1734e110c8.abcc9097-...	6:3	PRIVATE ADDRESSING	upstream	6	r2.sn-h5qzen76.googlevideo.com	movies	US	US	...	EDGECAST	60733	848/06464	Youtube	Streaming	googlevideo.com	NaN	1	524	1
3	1639390860000	23f24570-d2c9-4e6e-a833-3d1734e110c8.abcc9097-...	6:3	PRIVATE ADDRESSING	upstream	6	presence.teams.microsoft.com	cleaning	US	US	...	ATGS-MMD-AS	58244	848/06464	NaN	NaN	microsoft.com	NaN	1	540	1
4	1639390860000	23f24570-d2c9-4e6e-a833-3d1734e110c8.abcc9097-...	6:3	PRIVATE ADDRESSING	upstream	6	play.google.com	shopping	US	CN	...	GOOGLE	65504	848/06464	Google Play	Storage	google.com	NaN	1	509	1

5 rows × 36 columns

Figura 3.3.2

Haciendo una exploración de nuestro set de datos podemos ver rápidamente que tenemos inicialmente 36 columnas y 717370 filas, de las cuales, tenemos 10 columnas de tipo numéricas y 25 de tipo alfanumérico, además algunas columnas con datos NULL, los cuales vamos a tener que explorar con el fin de determinar qué tanto podrían aportar a un posible modelo de detección de anomalías.

A continuación, se observan las columnas con valores null, la cantidad de valores y el porcentaje que representan.

```
host : 386303 -> 53.85%
host_category : 512495 -> 71.44%
referer : 708736 -> 98.8%
saas_name : 452411 -> 63.07%
saas_family : 452411 -> 63.07%
host_l2 : 386303 -> 53.85%
referer_l2 : 708736 -> 98.8%
```

Figura 3.3.3

Esto permite tomar la decisión de eliminar aquellas columnas que por su alto nivel de datos null, no aportan a nuestro análisis, reduciendo de esta forma las dimensiones de nuestro set de datos; Para ello vamos a tomar la decisión de eliminar aquellas columnas con más de un 90% de datos null, como lo son: *referer y referer_l2.*

En el caso de las columnas restantes con datos null, vamos a reemplazar este valor con el valor *'No existente'.*

```
data = data.apply(lambda x: x.fillna('No existente'))
```

figura 3.3.4

Continuando con nuestro análisis, rápidamente podemos observar la columna *_time*, la cual se procede a explorar con el fin de observar que tanta historia tenemos en los datos y si nos pueden aportar en la detección de anomalías que es nuestro fin primordial.

Nos vamos a concentrar en la columna _time inicialmente con el fin de ver si nos aporta algo en nuestro análisis inicial o si nos puede generar ruido.

Convertimos nuestra columna y podemos notar en la siguiente imagen que solo tenemos datos para un día y para solo dos horas, por lo que, podemos determinar que no es suficiente historia como para considerar la columna *_time* en nuestro análisis inicial, vamos a proceder a eliminar dicha columna.

```
data['__time_convert'] = [datetime.fromtimestamp(int(ele[:-3])) for ele in data['__time'].astype(str)]
```

```
print("Cantidad de días: {0}".format(len(data['__time_convert'].dt.date.unique())))
print("Cantidad de horas: {0}".format(len(data['__time_convert'].dt.hour.unique())))
```

```
Cantidad de días: 1
Cantidad de horas: 2
```

figura 3.3.5

A continuación, se explora la cantidad de características que tenemos por columna, con el fin de ver si podemos descartar alguna otra columna que podrían no aportarnos y de esta forma, seguir disminuyendo las dimensiones de nuestro set de datos.

```
{'level': 1,
 'application_id': 35,
 'application_name': 35,
 'direction': 2,
 'engine_id': 5,
 'host': 4926,
 'host_category': 43,
 'wan_ip_country_code': 71,
 'lan_ip_country_code': 35,
 'l4_proto': 4,
 'lan_interface_name': 2,
 'lan_ip': 178,
 'lan_ip_as_name': 143,
 'lan_l4_port': 50903,
 'flow_exporter': 1,
 'device_ip': 1,
 'device_id': 1,
 'observation_id': 1,
 'selector_name': 5,
 'tcp_flags': 30,
 'tos': 12,
 'type': 1,
 'wan_interface_name': 3,
 'wan_ip': 3910,
 'wan_ip_as_name': 556,
 'wan_l4_port': 26336,
 'serial_number': 1,
 'saas_name': 131,
 'saas_family': 16,
 'host_l2': 1223,
 'events': 3,
 'sum_bytes': 27512,
 'sum_pkts': 3073}
```

figura 3.3.6

Sacando la cantidad de datos únicos por columna, podemos notar que tenemos columnas como: level, *flow_exporter, device_ip, device_id, observation_id, type, serial_number*, type, que no aportan valor en nuestro análisis, ya que solo contiene una característica, por lo que vamos a eliminar todas aquellas columnas con una unica característica.

```
eliminar = []
for x in cantidadDeCaracteristicasPorColumna:
    if cantidadDeCaracteristicasPorColumna[x] == 1:
        eliminar.append(x)
```

```
data = data.drop(eliminar, axis=1)
```

figura 3.3.7

Con este análisis inicial, hemos pasado de 36 columnas a reducir nuestro set de datos a 26 columnas.

```
print("-Cantidad de columnas {0}".format(data.shape[1]))
```

```
-Cantidad de columnas 26
```

figura 3.3.8

Además, como parte del proceso, se hizo un análisis manual de cada columna, determinando que tenemos dos columnas que representan lo mismo a nivel de datos, por lo que procedimos a eliminar una de ella, reduciendo nuestras columnas a 25.

```
data.groupby(['application_id', 'application_name']).size().reset_index().rename(columns={0:'count'})
```

	application_id	application_name	count
0	0:0	0:0	5720
1	129:1037	129:1037	2
2	129:1047	129:1047	32
3	129:1116	129:1116	32
4	129:34	129:34	29
5	1:1	1:1	159
6	3:123	3:123	762
7	3:137	3:137	43
8	3:1433	3:1433	7

figura 3.3.9

Podemos notar que ambas columnas representan lo mismo por lo que vamos a eliminar una de ellas con el fin de minimizar el ruido en nuestro modelo.

```
data = data.drop(['application_name'], axis=1)
print("-Cantidad de columnas {0}".format(data.shape[1]))
```

```
-Cantidad de columnas 25
```

figura 3.3.10

3.4.- APLICANDO TÉCNICA DE TRANSFORMACIÓN DEL SET DE DATOS

Como parte del proceso previo de aplicación de técnicas de agrupación y análisis de datos, los algoritmos requieren hacer una transformación y/o representación numérica de estos, con el fin de aplicar técnicas matemáticas que nos permita explorar gráficamente y saber su distribución, con el fin de, determinar en un análisis previo, si los datos tienen alguna tendencia a sufrir ruido que demuestre que se puede detectar comportamientos anómalos en estos.

Para ello, primero se obtienen las características únicas por columna, con el fin de explorar los datos y luego proceder a convertir cada uno a un valor numérico que sea soportado por los algoritmos que se utilizaran.

Con el siguiente código Python, se obtienen las características únicas por columna y se agrupan en un diccionario, que luego se usará para la transformación de cada valor alfanumérico a numérico, mediante el uso de su respectivo índice:

```
listaConCaracterizaciones = {}
for c in data.columns:
    listaConCaracterizaciones[c] = data[c].unique()
listaConCaracterizaciones
```

```
{'application_id': array(['6:3', '6:2', '3:80', '3:53', '3:443', '3:179', '1:1', '0:0',
       '3:60000', '3:22', '3:123', '6:1', '6:0', '3:514', '3:445',
       '3:137', '3:5432', '3:1433', '3:161', '129:1047', '3:15000',
       '129:1116', '129:34', '3:19903', '3:7996', '3:19902', '6:7',
       '3:19092', '3:52490', '3:51319', '3:61415', '129:1037', '3:60171',
       '3:3389', '3:51322'], dtype=object),
 'direction': array(['upstream', 'downstream'], dtype=object),
 'engine_id': array([  6,   3,   1,   0, 129]),
 'host': array(['teams.events.data.microsoft.com', 'ssl.gstatic.com',
       'r2.sn-h5qzen76.googlevideo.com', ..., 'pdns3.ultradns.org',
       'b-tr-teasc-euwe-05.westeurope.cloudapp.azure.com',
       'bid.dr.socdm.com'], dtype=object),
 'host_category': array(['cleaning', 'liste_blanche', 'movies', 'shopping', 'No existente',
       'music', 'chat', 'news', 'tracker', 'searchengines', 'liste_bu',
       'hobby/games-misc', 'socialnet', 'webmail', 'doh', 'imagehosting',
       'adv', 'publicite', 'remotecontrol', 'webphone', 'downloads',
       'updatesites', 'social_networks', 'audio-video', 'games',
       'recreation/humor', 'translation', 'bank', 'government', 'forum',
       'isp', 'filehosting', 'finance/banking', 'recreation/sports',
       'education/schools', 'urlshortener', 'recreation/travel',
       'library', 'gamble', 'press', 'radiotv', 'update',
       'hobby/games-online'], dtype=object),
 'wan_ip_country_code': array(['US', 'NL', 'SG', 'JP', 'IT', 'IE', 'GR', 'ES', 'DE', 'CN', 'TW',
       'CA', 'GB', 'AT', 'FR', 'IS', 'CH', 'AU', 'SE', 'NG', 'RU', 'MX',
       'BR', 'AZ', 'BY', 'ZA', 'LV', 'DK', 'PL', 'UA', 'UY', 'CO', 'SK',
       'ID', 'IN', 'LU', 'DZ', 'BE', 'PT', 'EE', 'CZ', 'HU', 'IL', 'LB',
       'NC', 'BA', 'GI', 'NO', 'VN', 'MD', 'SI', 'AR', 'RS', 'TR', 'MY',
       'CL', 'HR', 'LI', 'HK', 'FI', 'RO', 'IQ', 'TH', 'KR', 'BG', 'IR',
       'PK', 'LT', 'NZ', 'SC', 'KG'], dtype=object),
```

figura 3.4.1

Con el siguiente código se procede a transformar los datos de tipo alfanumérico a numérico, siendo estos representados por el valor del índice de los valores del diccionario generado anteriormente.

```
columnas = data.columns.values
data_player = data.copy()
for col in columnas:
    if data_player.dtypes[col] == "object":
        data_player["{0}_new".format(col)] = [ listaConCaracterizaciones[col].tolist().index(i) for i in data_player[col]]
```

figura 3.4.2

Lo que nos permite generar un set de datos nuevo, pero con todos los datos numéricos, conservando la misma cantidad de columnas y filas y sus valores ahora representados por un valor numérico que se puede identificar mediante el índice en nuestro diccionario anterior.

```
data_new.head(2)
```

	engine_id	l4_proto	lan_l4_port	tos	wan_l4_port	events	sum_bytes	sum_pkts	application_id_new	direction_new	...	lan_ip_new	lan_ip_as_name_new	selector_name_new	tcp_flags_new	wan_interface_name_new	wan_ip_new	wan_ip_as_name_new	saas_name_new	saas_family_new	host_l2_new
0	6	1	0	192	771	1	229	1	0	0	...	0	0	0	0	0	0	0	0	0	0
1	6	17	53	0	54606	1	509	1	0	0	...	1	1	0	0	0	1	1	1	1	1

2 rows × 25 columns

```
print("-Cantidad de columnas en nuestro nuevo data_new {0}".format(data_new.shape[1]))
```

```
-Cantidad de columnas en nuestro nuevo data_new 25
```

figura 3.4.3

3.5.- APLICANDO TÉCNICAS DE AGRUPACIÓN Y ANÁLISIS

Una vez los datos están transformados, podemos aplicar técnicas de agrupación y análisis; para ello vamos a usar KMeans para agrupar o crear los clúster que mejor representan la distribución de los datos y una reducción de nuestros componentes, aplicando la técnica de PCA, estos nos permitirá observar la distribución de nuestros datos y determinar si tenemos datos anómalos.

Agrupando datos con KMeans

Como ya tenemos nuestros datos limpios y transformados, podemos usar Kmeans que se encarga de aplicar fórmulas matemáticas, con el fin de determinar la cercanía de los datos y de esta forma determinar la agrupación que mejor explica las etiquetas de cada una de nuestros registros.

Para ello, primero se procederá a escalar los datos, esto con el fin de minimizar las distancias de nuestros datos .

Entrenamos y luego escalamos nuestros datos

```
scaler.fit(data_km)
data_km=scaler.transform(data_km)
```

```
print('Validamos el número de dimensiones: ', len(data_km))
```

```
Validamos el número de dimensiones:  717370
```

figura 3.5.1.1

Una vez escalado nuestros datos, con el fin de reducir la dimensionalidad en estos, procedemos a encontrar la cantidad de clúster que mejor representan nuestros datos, para ello, vamos a entrenar nuestros algoritmos por 50 iteraciones y de esta forma,

determinar según la recta de nuestra gráfica, que valor explica mejor la distribución de nuestros datos.

Entrenamos y obtenemos los datos necesarios para generar la gráfica que nos indicará la cantidad de clúster que mejor se adapta a nuestros datos.

```
Nc = range(1, 50)
kmeans = [KMeans(n_clusters=i, random_state=0, max_iter = 100) for i in Nc]
score = [kmeans[i].fit(data_km).inertia_ for i in range(len(kmeans))]
```

figura 3.5.1.2

Graficamos los datos obtenidos y determinamos que 15 es un buen número de clúster para nuestro set de datos, esto a pesar de que sabemos que nuestro distribucion final que andamos buscando, deberá de ser representada por 2 grupos que serían los que representen si el registro que se está analizando es o no anomalia.

```
plt.rcParams['figure.figsize'] = (14, 9)
plt.style.use('ggplot')
plt.plot(Nc,score)
plt.xlabel('Número de Clusteres')
plt.ylabel('Puntaje')
plt.title('Nuestra curva de clusters')
plt.show()
```

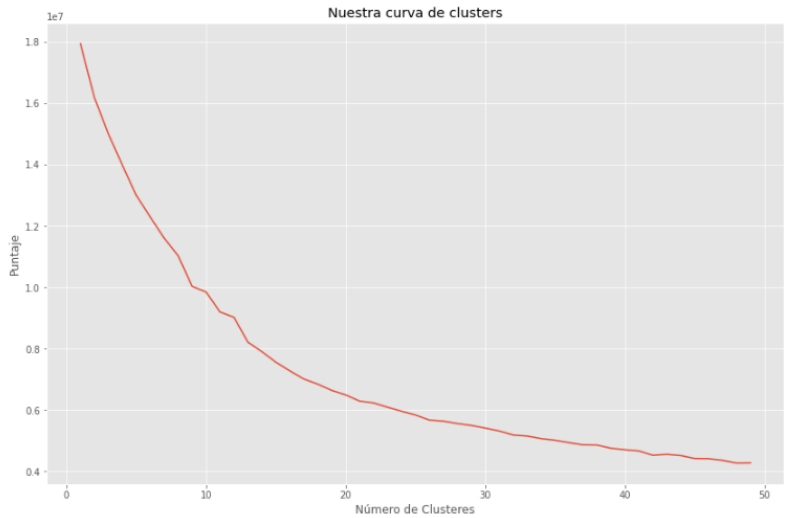


figura 3.5.1.3

Ahora se podrá:

Entrenar nuevamente nuestro modelo, con el fin de clasificar nuestros datos.
Obtener las etiquetas que representa cada clúster para cada registro de nuestro set de datos.
Asignamos cada etiqueta a nuestro set de datos y vemos como queda distribuido.

```
kmeans = KMeans(n_clusters=cantidadDeCluster, random_state=0, max_iter = 300).fit(data_km)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
```

Creamos un nuevo dataset con los datos escalado y el nombre de la respectiva columna

```
df_escalado = pd.DataFrame(data_km, columns=data_new.columns.values)
```

```
df_escalado['labels'] = labels
df_escalado[['sum_bytes','labels']]#.head(10)
```

	sum_bytes	labels
0	-0.013753	1
1	-0.013671	10
2	-0.013667	10
3	-0.013662	10
4	-0.013671	10
...	...	...
717365	-0.012189	8
717366	-0.013736	8
717367	-0.013761	8
717368	-0.013072	8
717369	-0.013719	8

717370 rows × 2 columns

figura 3.5.1.4

Se puede observar que ahora cada registro tiene una etiqueta o label, lo que nos permite saber cómo se agrupan estos, por lo que ahora podemos aplicar un análisis de componentes principales o PCA, para luego graficar nuestro set de datos y ver la distribución de este en un plano de 2 y 3 dimensiones.

Aplicando análisis de componentes principales “PCA”

Ejecutamos nuestro PCA, con el fin de buscar cual es el valor de dimensiones que mejor explican la distribución de nuestros datos, con el fin de, poder generar nuestro modelo de predicción, para ello vamos a dar la instrucción a nuestro algoritmo para que genere un nuevo set de datos, que explique el 90% de nuestros datos; recordemos que el objetivo del PCA es reducir las dimensiones, con el fin de facilitar la manipulación de estos, por lo que se busca reducir las 25 columnas actuales.

```
pca=PCA(0.9)
reduced_data = pca.fit_transform(data_km)
```

Simplificamos la lectura -> podemos ver que variables generadas por nuestro PCA explican mas nuestros datos

```
explained_variance = [(i,np.round(pca.explained_variance_ratio_[i],4)) for i in range(len(pca.explained_variance_ratio_))]
explained_variance
```

```
[(0, 0.1275),
 (1, 0.0967),
 (2, 0.0779),
 (3, 0.0765),
 (4, 0.0734),
 (5, 0.0579),
 (6, 0.0508),
 (7, 0.0458),
 (8, 0.0429),
 (9, 0.04),
 (10, 0.0377),
 (11, 0.0358),
 (12, 0.0339),
 (13, 0.0305),
 (14, 0.0291),
 (15, 0.0274),
 (16, 0.0216)]
```

El PCA inicial nos genera un dataset con la misma dimensionalidad a nivel de filas 717370 pero nos reduce las columnas de 25 a 17 las cuales explican nuestros datos en un 90%.

figura 3.5.1.5

Reconstruimos nuestro dataset con las componentes principales que explican mejor nuestro modelo, recordemos que nuestros datos están escalados.

```
PCAs = [ 'PCA{0}'.format(i)  for i in range(1, (cantiRecomendadaDePCA + 1))]
dfpca_15 = pd.DataFrame(data = reduced_data15,
                        columns=PCAs)
dfpca_15_anali = pd.concat([dfpca_15, df_escalado[['labels']]], axis=1)
dfpca_15_anali.head(4)
```

	PCA1	PCA2	PCA3	PCA4	PCA5	PCA6	PCA7	PCA8	PCA9	PCA10	PCA11	PCA12	PCA13	PCA14	PCA15	PCA16	PCA17	labels
0	-2.487928	0.600554	-0.439212	-0.231249	0.931107	1.740307	1.257741	0.077209	-1.461624	0.874957	0.458779	0.417743	0.322081	-0.726185	-1.328509	-1.933540	2.689489	1
1	-3.887235	0.955276	1.375445	-0.582252	0.975339	-0.322870	-1.072478	-1.765681	-0.219106	0.285956	0.384423	0.578729	0.589800	0.236022	-0.249374	-0.849233	0.439882	10
2	-3.760745	1.046337	1.501553	-0.649705	1.152473	-0.307974	-1.214331	-2.027513	-0.067158	0.304303	0.316556	0.673952	0.554954	0.135392	-0.318908	-0.819868	0.579955	10
3	-4.132853	0.836070	1.496345	-0.590867	0.937345	-0.464733	-0.895768	-1.704683	-0.294818	0.339886	0.409505	0.585050	0.695587	0.285558	-0.111276	-0.954034	0.517728	10

figura 3.5.1.6

Graficar un modelo de 17 dimensiones para ver cómo se distribuyen nuestros datos es casi imposible, por lo que, nos vamos a aprovechar de la técnica de PCA para forzar nuestro algoritmo a generar dos sets de datos, uno con 2 dimensiones y el otro con 3 dimensiones, esto con el fin de graficar y poder observar su distribución.

Generamos y exploramos nuestro set de datos de 2 dimensiones:

```
pca = PCA(n_components=2)
pca.fit(data_km)
reduced_data2 = pca.transform(data_km)

pca_graficar = pd.DataFrame(data = reduced_data2,
                            columns=['PCA1','PCA2'])

pca_graficar = pd.concat([pca_graficar, df_escalado[['labels']]], axis=1)
```

Mostramos los 4 primeros registros de nuestro nuevo dataset reducido a 2 dimensiones

```
pca_graficar.head(4)
```

	PCA1	PCA2	labels
0	-2.487941	0.600363	1
1	-3.887242	0.955237	10
2	-3.760750	1.046311	10
3	-4.132863	0.836023	10

figura 3.5.1.7

Lo graficamos y vemos la distribución de los datos.

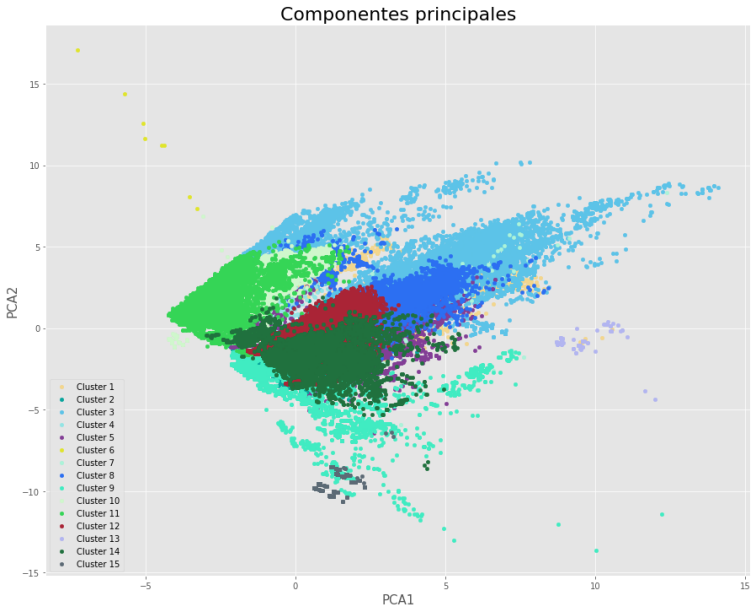


figura 3.5.1.8

Generamos y exploramos nuestro set de datos de 3 dimensiones:

```
pca = PCA(n_components=3)
pca.fit(data_km) # obtener los componentes principales
reduced_data3 = pca.transform(data_km) # convertimos nuestros datos con las nuevas dimensiones de PCA

pca_graficar3 = pd.DataFrame(data = reduced_data3,
                             columns=['PCA1','PCA2','PCA3'])

pca_graficar3 = pd.concat([pca_graficar3, df_escalado[['labels']]], axis=1)
```

figura 3.5.1.9

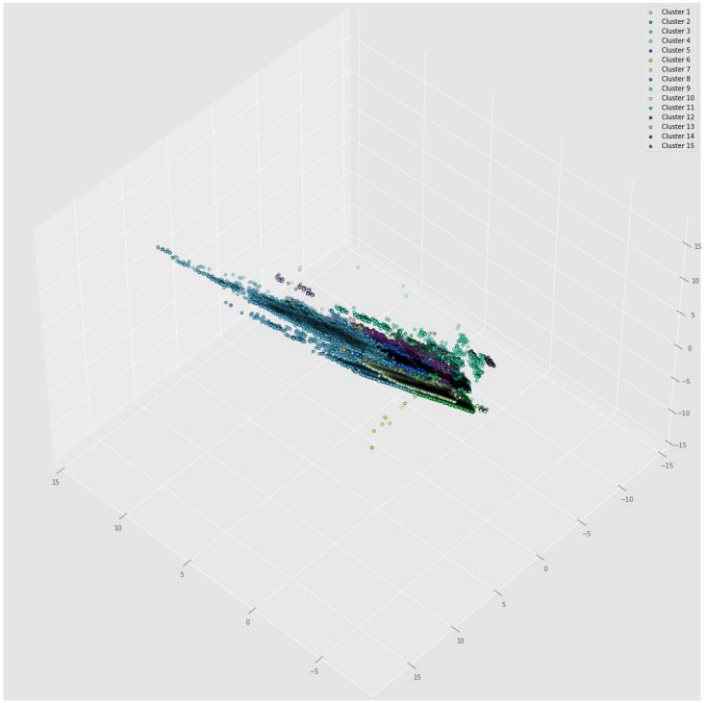


figura 3.5.1.10

Cada clúster queda representado por un color.

Conclusión del análisis inicial de los datos suministrados por la empresa

Después del análisis aplicado a la data recibida por parte de la empresa, con la cual se trabajado en este este capítulo, podemos concluir, basado en un juicio no experto, que los datos analizados corresponden a una situación “normal” de NO ciberataque o NO anomalía; y que los datos no son muy adecuados para realizar el entrenamiento del modelo de detección de anomalías que se pretende desarrollar en las siguientes etapas, esto apoyado en las gráficas suministradas anteriormente, donde se ve una distribución normal de los datos, además de que la variabilidad parece ser muy elevada entre las columnas analizadas.

Como parte del proceso de análisis, se han tenido inconvenientes con los datos, ya que, el set de datos contiene pocos registros que permitan continuar con el fortalecimiento y explicación de los datos y se ha dificultado el coordinar con la empresa y experto para poder acceder a más datos que permitan explicar mejor los datos y/o crear un modelo para predecir si existen o no anomalías.

No obstante, lo anterior, a pesar de los déficits que presentaba el set de datos, fue posible identificar atributos como IP de origen y destino, tipo de aplicación, host, cantidad de paquetes y cantidad de bytes transferidos, puertos y tipo de transferencia “subida o descarga”, etc. como candidatos para poder detectar posibles anomalías.

3.6.- CONCLUSIÓN DEL TRABAJO REALIZADO

Podemos señalar que, después de realizar una exhaustiva evaluación de dataset que se tenía disponible, nos encontramos que este presentaba fuertes debilidades, en término del volumen de la data que contiene y de la aparición del tipo de evento que se busca modelar, se requiere que la data contenga casos de anomalías y pareciera a juicio de no experto en el área de seguridad y basados en la distribución de los datos, que no existen anomalía en los datos.

La situación antes descrita afecta fuertemente los siguientes pasos que se deben seguir en el diseño e implementación del modelo de detección de anomalías, el cual debe entrenarse con un dataset adecuado, para que tenga la capacidad de clasificar y predecir un tipo de ataque a una red de ordenadores, además de que, para poder fortalecer y etiquetar cada clúster de nuestro set de datos, requerimos del experto en el área, suministrado por la empresa T, el cual en esta etapa se ha dificultado el acceso a este.

Teniendo presente que la probabilidad de obtener más data de parte de la empresa es baja, y que además el tiempo que se requiere para el análisis y verificación de la data, en término de si contiene o no eventos de ataques, no es la idónea, debido a que se requiere del juicio experto de la persona en seguridad, suministrado por la empresa, del cual se dificulto poder contar con su apoyo, y que ya se avanzó en reconocer que hay ciertos atributos que son adecuados para detectar algunos tipos de ataque, consideramos que es relevante continuar con el el trabajo desarrollado

hasta ahora, con la salvedad que, en la siguiente etapa, se buscará y seleccionará un dataset público, que disponga del volumen de datos y de algunos de los atributos antes identificados, que nos permita realizar, de manera adecuada, el entrenamiento del modelo y así otorgarle la capacidad de clasificar y reconocer un cierto tipo de ataque.

4. ANÁLISIS DEL DATASET NSL-KDD

Según la conclusión del análisis de los datos proporcionados por la empresa y dado que la empresa no va a proporcionar más datos en este momento, es necesario trabajar con un nuevo dataset similar al dataset de la empresa pero más completo, idealmente etiquetado y con un volumen y periodo de tiempo significativo para nuestro estudio. Continuando con el proceso de experimentación, análisis y exploración de datos relacionados con tráfico de red, con el fin de poder crear un modelo capaz de predecir en etapas tempranas los ataques de red, y dado los problemas relacionados con el set de datos analizados anteriormente, vamos a proceder con un nuevo análisis de datos, usando de referencia el conjunto de datos de NSL-KDD el cual se basa en el conjunto de datos KDD'99; Se tomará como referencia este set de datos, debido a que contiene datos etiquetados en diferentes tipos de ataques o anomalías, que podemos categorizar en 4 grupos: DoS, Probe, U2R y R2L, además de contener datos considerados normales o no anómalos.

En este set de datos, contamos con columnas tipo etiquetas definidas por expertos en el campo, los cuales han logrado etiquetar el nivel de ataque y el tipo de ataque, como parte del proceso vamos a usar solo una de estas etiquetas debido a que lo que buscamos es poder generar un modelo capaz de predecir si es o no anomalia el registro analizar; para ello nos vamos a basar en la columna attack y eliminaremos la columna level; es importante mencionar antes de iniciar con el análisis, que se procederá a unificar la columna attack en dos grandes categorías: registros no anómalos representados por un 0 y datos anómalos representados por un 1; debido a que, la columna attack para los datos de tipo anómalos se distribuyen de la siguiente forma:

- 11 tipos diferentes de ataques **DoS**
  apache2, back, land, neptune, mailbomb, pod, processtable, smurf, teardrop, udpstorm, worm
- 6 tipos diferentes de ataques **Probe**
  ipsweep, mscan, nmap, portsweep, saint, satan
- 7 tipos diferentes de ataques **U2R**
  buffer overflow, loadmodule, perl, ps, rootkit, sqlattack, xterm
- 15 tipos diferentes de ataques **R2L**
  ftpwrite, guesspasswd, httptunnel, imap, multihop, named, phf, sendmail, Snmpgetattack, spy, snmpguess, warezclient, warezmaster, xlock, xsnoop

Otro dato a tomar en cuenta es la columna duración, esta columna fue categorizada, de tal forma de que se agruparon los datos a nivel de segundos, por lo que, aquellas transacciones que no superan el 1 segundo o que van de 0 a menos 1 segundo, se agruparon en la categoría 0, los que van de 1 a menos 2 segundo, se agruparon en la categoría 1, los que van de 2 a 3 segundo, se agruparon en la categoría 2 y de esta forma se construyeron los demás grupos.

4.1.- CARGA Y AGRUPACIÓN DEL SET DE DATOS PARA EL ANÁLISIS

Nuestro set de datos está separado actualmente en dos grandes grupos, pruebas y entrenamiento, con sus respectivas etiquetas de anomalía y no anomalía, como parte de nuestro proceso, vamos a unir estos sed de datos y quitar registros duplicados, esto con el fin de luego construir y distribuir nuestros datos de pruebas y entrenamiento a conveniencia del análisis aplicado, además de incluir el etiquetado de las respectivas columnas.

Cargamos los datos NSL-KDD

```
columnas = ['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urgent'
,'hot','num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','num_root','num_file_creations'
,'num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_login','count','srv_count','serror_rate'
,'srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate','diff_srv_rate','srv_diff_host_rate','dst_host_count'
,'dst_host_srv_count','dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_host_rate'
,'dst_host_serror_rate','dst_host_srv_serror_rate','dst_host_rerror_rate','dst_host_srv_rerror_rate','attack','level']


testskdd = pd.read_csv('datav2/NSL-KDD/KDDTest+.txt',names = columnas)
testskdd21 = pd.read_csv('datav2/NSL-KDD/KDDTest-21.txt',names = columnas)
traindd = pd.read_csv('datav2/NSL-KDD/KDDTrain+.txt',names = columnas)
traindd20 = pd.read_csv('datav2/NSL-KDD/KDDTrain+_20Percent.txt',names = columnas)

data = pd.concat([testskdd, testskdd21,traindd,traindd20])
```

Cantidad de filas antes de quitar duplicados 185,559.00 y después de quitar duplicados 147,907.00

figura 4.1.1

4.2.- EXPLORANDO EL SET DE DATOS.

Como parte del proceso exploratorio, podemos observar que, de los ***147,907.00*** registros del set de datos quitando registros duplicados; ***76,967.00*** son registros no anómalos y ***70,940.00*** son registros clasificados como anómalos.

```
duracionDeTiposAtaque = data[data['attack'] != 'normal']['duration']

somelist_df = pd.DataFrame(duracionDeTiposAtaque)

print("Cantidad de registros {0} - Cantidad de no ataques {1} - Cantidad de ataques {2}".format(filasDespuesDeQuitarDuplicados, (filasDespuesDeQuitarDuplicados - somelist_df.shape[0]),somelist_df.shape[0] ))
```

```
Cantidad de registros 147907 - Cantidad de no ataques 76967 - Cantidad de ataques 70940
```

figura 4.2.1

En la siguiente imagen, podemos observar la cantidad de columnas y la distribución de nuestras columnas de acuerdo al tipo de dato.

```
print("Cantidad de columnas: {0}".format(data.shape[1]))
pd.value_counts(data.dtypes)
```

```
Cantidad de columnas: 43
int64      24
float64    15
object      4
dtype: int64
```

figura 4.2.2

Tenemos 4 columnas de tipo alfanumérico: *protocol_type, service, flag, attack*, las cuales en punto del proceso se va a requerir convertir en numericas; continuando con el análisis, podemos ver cómo se distribuyen estas columnas y cual es el valor que más se repite, con el fin de darnos una idea de cómo están distribuidos nuestros datos alfanuméricos.

Exploramos nuestras columnas alfanumérico

```
columnasTipoObject = [ i for i in data.columns.values if (data.dtypes[i] == "object")  ]

print("Columnas de tipo alfanumérico: {0} - Total: {1}".format(columnasTipoObject, len(columnasTipoObject)))
```

```
Columnas de tipo alfanumérico: ['protocol_type', 'service', 'flag', 'attack'] - Total: 4
```

Validamos la distribución de los datos no numéricos

```
data.describe(exclude="number")
```

	protocol_type	service	flag	attack
count	147907	147907	147907	147907
unique	3	70	11	40
top	tcp	http	SF	normal
freq	121361	48168	89394	76967

figura 4.2.3

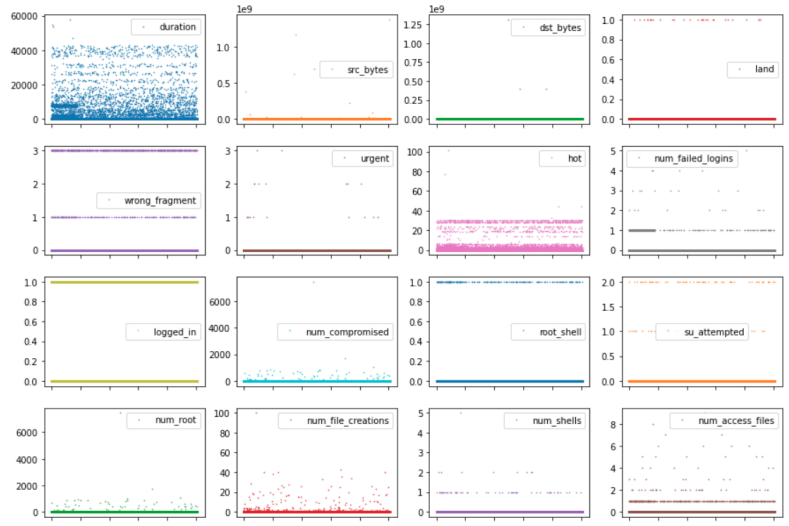
Se puede notar de la imagen anterior, que el protocolo que más se repite es el tcp, los servicios http, en la bandera SF y tenemos mayor cantidad de datos normales que anómalos, como se demostró anteriormente.

Con respecto a los datos null, este dataset, al ser un dataset usado académicamente, se encuentra limpio y no contiene datos null.

A continuación, se puede observar gráficamente la distribución de los datos por columna, inicialmente solo de las columnas de tipo numéricas. Esto nos ayuda a entender cómo se distribuye la data de cada columna.

Exploramos la distribución de los datos por columna

```
data.plot(lw=0,
          marker=".",
          subplots=True,
          layout=(-1, 4),
          figsize=(15, 30),
          markersize=1);
```



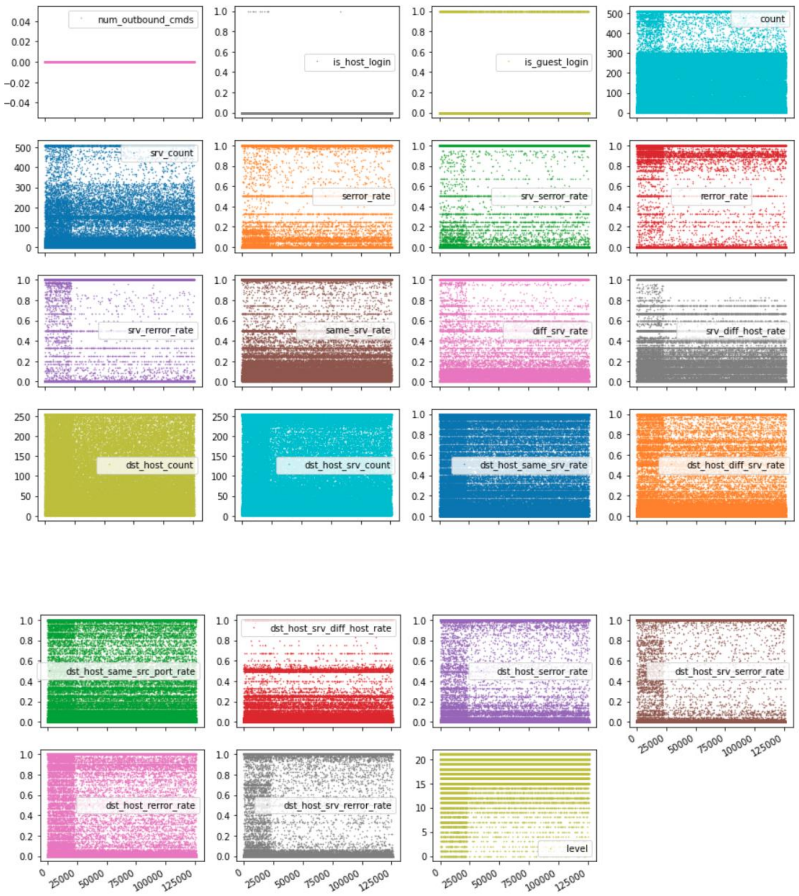


figura 4.2.4

Se puede ver gráficamente la distribución de los datos para las columnas de tipo alfanuméricas.

```
fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(12, 8))
df_non_numerical = data.select_dtypes(exclude=["number"])

for col, ax in zip(df_non_numerical.columns, axes.ravel()):
    df_non_numerical[col].value_counts().plot(
        logy=True, title=col, lw=0, marker=".", ax=ax)

plt.tight_layout();
```

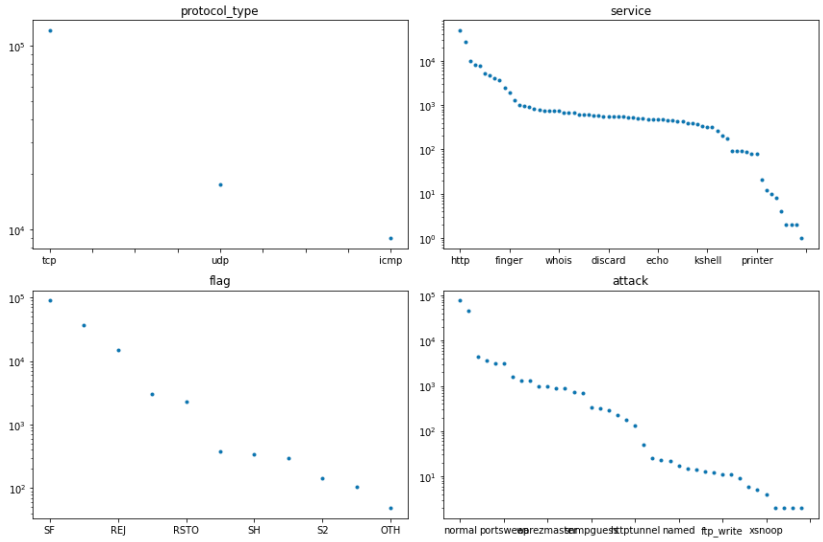


figura 4.2.5

Se observa la cantidad de características por columna, esto nos permitirá observar posibles columnas que no aporten a nuestro futuro modelo, porque solo contienen una característica.

```
cantidadDeCaracteristicasPorColumna = {}

for c in columnas:
    if c in data.columns.values:
        cantidadDeCaracteristicasPorColumna[c] = len(data[c].unique())

cantidadDeCaracteristicasPorColumna
```

```
{'duration': 3424,
 'protocol_type': 3,
 'service': 70,
 'flag': 11,
 'src_bytes': 3601,
 'dst_bytes': 10401,
 'land': 2,
 'wrong_fragment': 3,
 'urgent': 4,
 'hot': 29,
 'num_failed_logins': 6,
 'logged_in': 2,
 'num_compromised': 96,
 'root_shell': 2,
 'su_attempted': 3,
 'num_root': 91,
 'num_file_creations': 36,
 'num_shells': 4,
 'num_access_files': 10,
 'num_outbound_cmds': 1,
 'is_host_login': 2,
 'is_guest_login': 2,
 'count': 512,
 'srv_count': 512,
 'serror_rate': 99,
 'srv_serror_rate': 94,
 'rerror_rate': 98,
 'srv_rerror_rate': 95,
 'same_srv_rate': 101,
 'diff_srv_rate': 101,
 'srv_diff_host_rate': 87,
 'dst_host_count': 256,
 'dst_host_srv_count': 256,
 'dst_host_same_srv_rate': 101,
 'dst_host_diff_srv_rate': 101,
 'dst_host_same_src_port_rate': 101,
 'dst_host_srv_diff_host_rate': 75,
 'dst_host_serror_rate': 101,
 'dst_host_srv_serror_rate': 101,
 'dst_host_rerror_rate': 101,
 'dst_host_srv_rerror_rate': 101,
 'attack': 40,
 'level': 22}
```

figura 4.2.6

Como se mencionó anteriormente, se observa que la columna num_outbound_cmds, únicamente posee una característica, por lo que se puede proceder con la eliminación de dicha columna, ya que no aporta valor a nuestro modelo.

```
eliminar = []

for x in cantidadDeCaracteristicasPorColumna:
    if cantidadDeCaracteristicasPorColumna[x] == 1:
        eliminar.append(x)

if eliminar in data.columns.values:
    data = data.drop(eliminar, axis=1)

print("Columnas eliminadas porque solo tienen una característica {0} y cantidad de columnas eliminadas {1}"
      .format(eliminar, len(eliminar)))
```

```
Columnas eliminadas porque solo tienen una característica ['num_outbound_cmds'] y cantidad de columnas eliminadas 1
```

figura 4.2.7

4.3.- LIMPIEZA NUESTRO SET DE DATOS.

Antes de aplicar la transformación de nuestros datos, se va a eliminar y respaldar aquellas columnas mencionadas al inicio de este capítulo, las cuales son generadas por juicio experto con el fin de explicar los datos, procederemos a respaldar y transformar los valores de la columna attack, el cual nos indica si el registro es un registro normal o anómalo, para ello, se representan los datos normales con el valor 0 y los anómalos con valor 1.

0 - normal y 1- Ataque

```
data_attack_flag = data.attack.map(lambda a: 0 if a == 'normal' else 1)
```

figura 4.3.1

Una vez respaldado los valores de ataque o no paquete, procedemos a eliminar las dos columnas, apoyado de la explicación dada al inicio de este capítulo.

```
columnaAtaque = ['attack','level']
if columnaAtaque[0] in data.columns.values:
    data = data.drop(columnaAtaque, axis=1)
```

figura 4.3.2

4.4.- TRANSFORMANDO NUESTRO SET DE DATOS.

Como se mencionó en puntos anteriores, con el fin de que nuestros algoritmos funcionen correctamente, se debe cambiar nuestros datos categóricos representados por valores alfanuméricos a numéricos, por lo que, se procederá

hacer el ajuste a las 4 columnas de nuestro set de datos mencionadas anteriormente con el fin de que cada uno de los valores sean representados numéricamente por el valor del índice en que se ubicará dentro del diccionario de datos que se construirá a partir de cada columna.

Tomar en cuenta que, como parte de las columnas alfanuméricas, estaba incluida la columna attack, la cual eliminamos en el punto anterior, por lo que vamos a crear nuestro diccionario y luego proceder a transformar, aquellas columnas restantes de tipo alfanuméricas.

```
listaConCaracterizaciones = {}
for c in data.columns:
    listaConCaracterizaciones[c] = data[c].unique()
listaConCaracterizaciones
```

```
{'duration': array([    0,     2,     1, ...,  5430, 11680,   679]),
 'protocol_type': array(['tcp', 'icmp', 'udp'], dtype=object),
 'service': array(['private', 'ftp_data', 'eco_i', 'telnet', 'http', 'smtp', 'ftp',
        'ldap', 'pop_3', 'courier', 'discard', 'ecr_i', 'imap4',
        'domain_u', 'mtp', 'systat', 'iso_tsap', 'other', 'csnet_ns',
        'finger', 'uucp', 'whois', 'netbios_ns', 'link', 'Z39_50',
        'sunrpc', 'auth', 'netbios_dgm', 'uucp_path', 'vmnet', 'domain',
        'name', 'pop_2', 'http_443', 'urp_i', 'login', 'gopher', 'exec',
        'time', 'remote_job', 'ssh', 'kshell', 'sql_net', 'shell',
        'hostnames', 'echo', 'daytime', 'pm_dump', 'IRC', 'netstat', 'ctf',
        'nntp', 'netbios_ssn', 'tim_i', 'supdup', 'bgp', 'nnsp', 'rje',
        'printer', 'efs', 'X11', 'ntp_u', 'klogin', 'tftp_u', 'red_i',
        'urh_i', 'http_8001', 'aol', 'http_2784', 'harvest'], dtype=object),
 'flag': array(['REJ', 'SF', 'RSTO', 'S0', 'RSTR', 'SH', 'S3', 'S2', 'S1',
        'RSTOS0', 'OTH'], dtype=object),
 'src_bytes': array([    0, 12983,    20, ...,  2816, 11466,  2358]),
 'dst_bytes': array([    0,    15, 14515, ...,  5424, 17655, 52866]),
 'land': array([0, 1]),
 'wrong_fragment': array([0, 1, 3]),
 'urgent': array([0, 1, 2, 3]),
 'hot': array([  0,   4,   2,   1,   7,   6,   5,   3,  22,  10,  11,  19, 101,
```

Construimos nuestro diccionario

figura 4.4.1

Reemplazamos los datos alfanuméricos por el valor del índice que le corresponde al dato en el diccionario construido anteriormente.

```
columnas = data.columns.values

data_player = data.copy() ## Copiamos el dataset que tenemos limpio, con el fin de poder reutilizar luego

for col in columnas:
    if data_player.dtypes[col] == "object":
        data_player["{0}_".format(col)] = [ listaConCaracterizaciones[col].tolist().index(i)
                                           for i in data_player[col]]
```

figura 4.4.2

4.5.- VISUALIZACIÓN DE NUESTRO SET DE DATOS PREPROCESADO.

Visualización de la matriz de correlación de los datos por columnas.

Podemos notar la relación existente entre nuestras columnas, se observa aquellas columnas que tiene una relación de más del 75%, representadas por una **S**, las que tiene una relación mayor al 50 y menor al 75% representadas por una **M**, las que superar el 25 % y menor al 50% por una **W** y por último, todo lo que queda, con una relación demasiado baja, menor al 25%**.**

```
datosCorrelacionados = dataDeTrabajo.corr(method="spearman")

labels = np.where(np.abs(datosCorrelacionados)>0.75, "S",
                  np.where(np.abs(datosCorrelacionados)>0.5, "M",
                           np.where(np.abs(datosCorrelacionados)>0.25, "W", "")))

plt.figure(figsize=(15, 15))
sns.heatmap(datosCorrelacionados, mask=np.eye(len(datosCorrelacionados)), square=True,
            center=0, annot=labels, fmt='', linewidths=.5,
            cmap="vlag", cbar_kws={"shrink": 0.8});
```

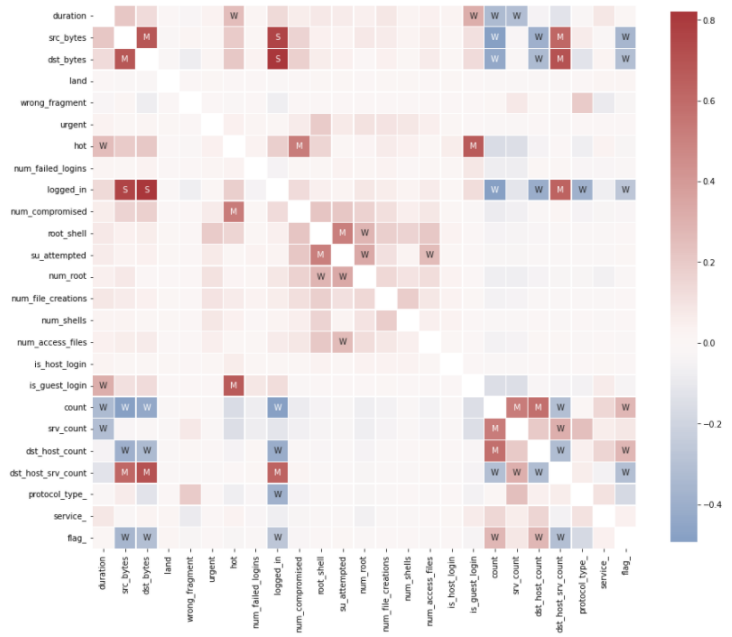


figura 4.5.1

Visualización del set de datos preprocesados.

El set de datos lo reducimos en este punto a 25 columnas y todos nuestros valores por columna de tipo numéricas.

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...	num_access_files	is_host_login	is_guest_login	count	srv_count	dst_host_count	dst_host_srv_count	protocol_type_	service_	flag_
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	229	10	255	10	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	136	1	255	1	0	0	0
2	2	12983	0	0	0	0	0	0	0	0	...	0	0	0	1	1	134	86	0	1	1
3	0	20	0	0	0	0	0	0	0	0	...	0	0	0	1	65	3	57	1	2	1
4	1	0	15	0	0	0	0	0	0	0	...	0	0	0	1	8	29	86	0	3	2

5 rows × 25 columns

figura 4.5.2.1

4.6.- ESCALANDO NUESTROS DATOS PARA MEJORAR LOS RESULTADOS DE LOS MODELOS.

Se escalan los datos, para ello entrenamos el algoritmo de escalado y luego se procede a escalar.

```
scaler=StandardScaler()
scaler.fit(data_km)
data_km=scaler.transform(data_km)
```

figura 4.6.1

Observamos los datos escalados.

```
df_escalado = pd.DataFrame(data_km, columns=dataDeTrabajo.columns.values)
df_escalado.head(6)
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...	num_access_files	is_host_login	is_guest_login	count	srv_count	dst_host_count	dst_host_srv_count	protocol_type_	service_	flag_
0	-0.112716	-0.007450	-0.004623	-0.014004	-0.085118	-0.010425	-0.09417	-0.059956	-0.823833	-0.011492	...	-0.042215	-0.009008	-0.111865	1.252482	-0.239937	0.722302	-0.983668	-0.446121	-0.725617	-1.316688
1	-0.112716	-0.007450	-0.004623	-0.014004	-0.085118	-0.010425	-0.09417	-0.059956	-0.823833	-0.011492	...	-0.042215	-0.009008	-0.111865	0.454382	-0.360316	0.722302	-1.064599	-0.446121	-0.725617	-1.316688
2	-0.111905	-0.005055	-0.004623	-0.014004	-0.085118	-0.010425	-0.09417	-0.059956	-0.823833	-0.011492	...	-0.042215	-0.009008	-0.111865	-0.704151	-0.360316	-0.506077	-0.300243	-0.446121	-0.647314	-0.454322
3	-0.112716	-0.007447	-0.004623	-0.014004	-0.085118	-0.010425	-0.09417	-0.059956	-0.823833	-0.011492	...	-0.042215	-0.009008	-0.111865	-0.704151	0.495715	-1.835974	-0.561023	1.050565	-0.569011	-0.454322
4	-0.112310	-0.007450	-0.004619	-0.014004	-0.085118	-0.010425	-0.09417	-0.059956	-0.823833	-0.011492	...	-0.042215	-0.009008	-0.111865	-0.704151	-0.266688	-1.572025	-0.300243	-0.446121	-0.490708	0.408045
5	-0.112716	-0.007401	-0.000712	-0.014004	-0.085118	-0.010425	-0.09417	-0.059956	1.213838	-0.011492	...	-0.042215	-0.009008	-0.111865	-0.678406	-0.320190	-0.292887	1.219478	-0.446121	-0.412405	-0.454322

6 rows × 25 columns

4.7.- REDUCCIÓN DE LA DIMENSIONALIDAD DE NUESTRO SET DE DATOS, MEDIANTE EL USO DE LA TÉCNICA DE ANÁLISIS DE COMPONENTES PRINCIPALES (PCA).

El objetivo inicial de aplicar la técnica de reducción de dimensiones es poder ver la distribución de nuestros datos y visualizar su distribución mediante una gráfica en 2 o 3 dimensiones, debido a que con nuestro set de datos actual de 25 dimensiones se nos hace imposible graficarlo en un plano de 2 o 3 dimensiones, se necesita reducir dicha cantidad de columnas.

Visualizando del set de datos en 2 dimensiones apoyado de la técnica de PCA.

Se observa la distribución de los datos anómalos y no anómalos, en una gráfica de dos dimensiones, para lo cual, vamos a tener que entrenar un modelo mediante la técnica de PCA y transformar nuestro set de datos en 2 columnas.

```
pca = PCA(n_components=2)
pca.fit(data_km)

reduced_data2 = pca.transform(data_km)

explained_variance = [(i,np.round(pca.explained_variance_ratio_[i],4)) for i in range(len(pca.explained_variance_ratio_))]

print( "Las dos dimensiones explican el {0}% de los datos".format(((explained_variance[0][1] + explained_variance[1][1])*100)) )
```

```
Las dos dimensiones explican el 22.57% de los datos
```

figura 4.7.1.1

Visualizamos nuestro set de datos reducido a 2 dimensiones

```
pca_graficar = pd.DataFrame(data = reduced_data2,
                            columns=['PCA1','PCA2'])

pca_graficar['labels'] = data_attack_flag.values

pca_graficar.head(5)
```

	PCA1	PCA2	labels
0	-0.875094	0.367159	1
1	-0.555471	0.219968	1
2	0.355690	-0.237213	0
3	0.353401	-0.286191	1
4	0.446640	-0.261400	1

figura 4.7.1.2

Visualizamos la distribución de datos en 2 dimensiones.

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('PCA1', fontsize = 25)
ax.set_ylabel('PCA2', fontsize = 25)
ax.set_title('Componentes principales', fontsize = 10)

ax.scatter(x = pca_graficar.PCA1, y = pca_graficar.PCA2,
           c=color_theme[pca_graficar.labels],
           marker='*', s=15)
plt.show()
```

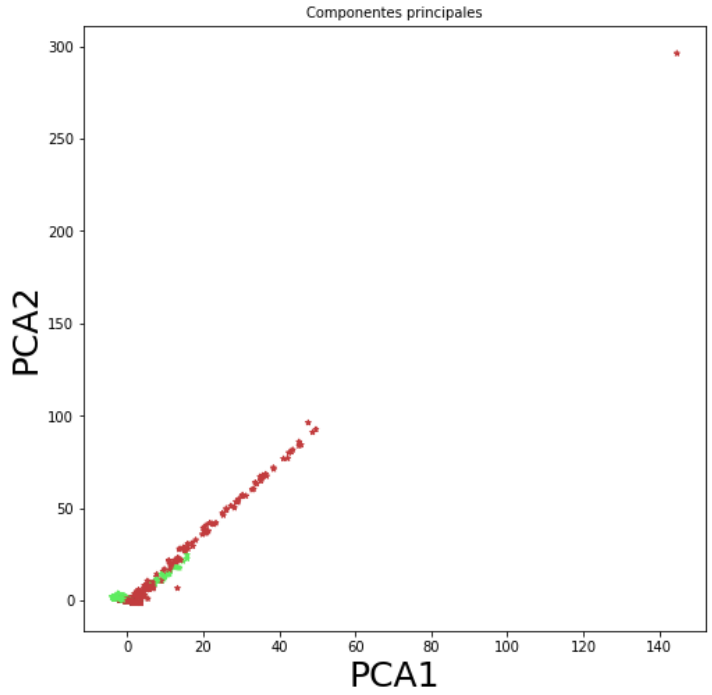


figura 4.7.1.1

Podemos notar que sigue una distribución aceptable y marcada de los datos, correspondiente datos anómalos y no anómalos, por lo que se podría esperar generar un modelo lo suficientemente bueno para poder detectar, aun así, vamos a ver nuestros datos en un tercera dimensión.

Visualizando el set de datos en 2 dimensiones apoyado de la técnica de PCA. Siguiendo los pasos del gráfico anterior, vamos a proceder con reducir las dimensiones a 3 y graficar los datos resultantes del proceso de reducción.

Entrenamos nuestro modelo de PCA y aplicamos la reducción de dimensiones

```
pca = PCA(n_components=3)
pca.fit(data_km) # obtener los componentes principales
reduced_data3 = pca.transform(data_km) # convertimos nuestros datos con las nuevas dimensiones de PCA

pca_graficar3 = pd.DataFrame(data = reduced_data3,
                             columns=['PCA1','PCA2','PCA3'])

pca_graficar3['labels'] = data_attack_flag.values

color_theme = generarColoresParaGraficar(cantidad = 3)

explained_variance = [(i,np.round(pca.explained_variance_ratio_[i],4)) for i in range(len(pca.explained_variance_ratio_))]

print( "Las dos dimensiones explican el {0}% de los datos".format(round((explained_variance[0][1] + explained_variance[1][1])*100,2)) )
```

```
Las dos dimensiones explican el 22.56% de los datos
```

figura 4.7.2.1

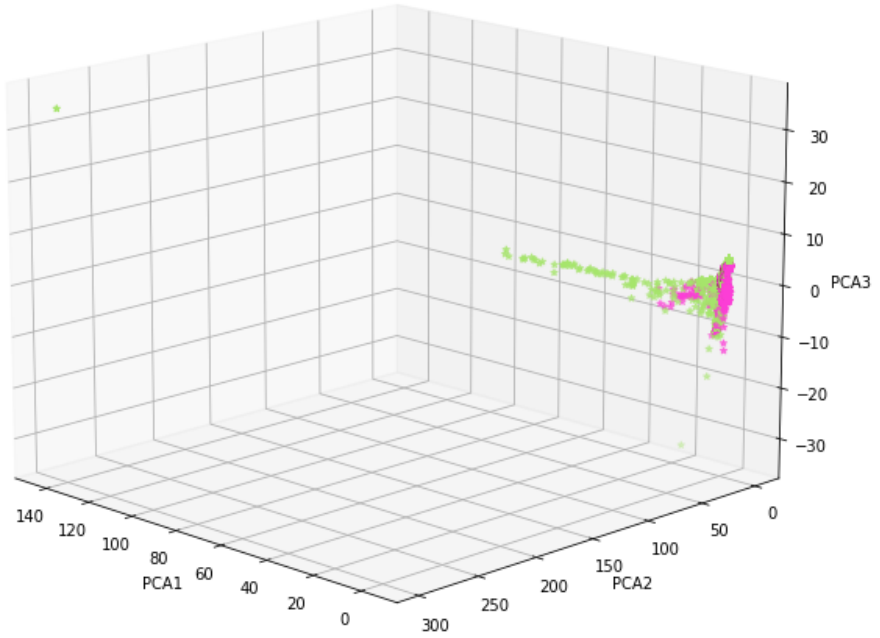


figura 4.7.2.2

Al igual que el gráfico anterior, podemos notar que sigue una distribución aceptable y marcada de los datos, correspondiente datos anómalos y no anómalos, por lo que podríamos continuar con la construcción de nuestro modelo capaz de detectar anomalías.

4.8.- VARIABLES RELEVANTES (PCA)

Se procede a reducir las dimensiones del dataset ya limpio, con el fin de basar la construcción de los modelos con las columnas que mejor explican los datos, vamos a usar las columnas que expliquen nuestros datos según PCA, hasta en un 90%.

Entrenamos y creamos nuestro nuevo dataset con las nuevas dimensiones

```
pca_, reduced_data, explained_variance = construirYTransformarDatosConPCA(data_km, porcentaje_ = 90)
print(explained_variance)
```

```
[(0, '11.6%'), (1, '10.97%'), (2, '7.5%'), (3, '6.64%'), (4, '5.35%'), (5, '4.63%'), (6, '4.29%'), (7, '4.1%'),
(8, '4.07%'), (9, '4.01%'), (10, '4.0%'), (11, '3.98%'), (12, '3.95%'), (13, '3.75%'), (14, '3.57%'), (15, '3.2
6%'), (16, '2.95%'), (17, '2.54%')]
```

figura 4.8.1

El análisis de componentes aplicado nos genera un set de datos con las siguientes dimensiones: filas 147907 y columnas 18, el set de datos original tiene: filas 147907 y columnas 25, reduciendo nuestro dataset en 7 columnas.
Podemos observar que 18 de nuestras variables generadas, explican el 91.16% de nuestros datos.

4.9.- CONSTRUYENDO SET DE DATOS PARA ENTRENAMIENTO Y PRUEBAS.

Como parte de nuestro proceso de construcción y pruebas de posibles modelos para detección de anomalías, se procede a separar el set de datos que explica el 91.16% de nuestros datos, el cual fue generado anteriormente con nuestro algoritmo de PCA.

Para ello, vamos a separar los datos en 2 set de datos, 20% para probar nuestros modelos con datos que no conoce y 80% de los datos restantes para entrenar nuestro modelo.

```
porcentajeDatosDePrueba = 0.20

train_, test_, train_lbl, test_lbl = train_test_split( reduced_data,
                                                       data_attack_flag.values,
                                                      test_size=porcentajeDatosDePrueba,
                                                      random_state=0)
```

```
print("Nuestro datos de entrenamientos({0}% - Data: {1} - Etiquetas: {2}) y prueba({3}% - Data: {4} - Etiquetas: {5})".format(
                      (100 - (porcentajeDatosDePrueba * 100)),
                      train_.shape,
                      train_lbl.shape,
                      (porcentajeDatosDePrueba * 100),
                      test_.shape,
                      test_lbl.shape
                      ))
```

```
Nuestro datos de entrenamientos(80.0% - Data: (118325, 18) - Etiquetas: (118325,)) y prueba(20.0% - Data: (29582, 18) - Etiquetas: (29582,))
```

figura 4.9.1

4.10.- EXPERIMENTANDO CON DIFERENTES TIPOS DE ALGORITMOS CON EL FIN DE CREAR UN MODELO DE DETECCIÓN DE ANOMALÍAS.

Como parte del proceso de construcción y definición del mejor modelo que permita lograr el objetivo de detectar anomalías en etapas tempranas de su evolución, vamos a proceder a probar los siguientes algoritmos y en función de estos, determinar cuál se adapta más a nuestras necesidades.
Algoritmos a probar con nuestro set de datos depurado y reducido:
Regresión LogísticaRandom Forest
Máquinas de Vector Soporte (SVM)
Árboles de decisión
Red neuronal enfocado en Keras

Experimentando con un modelo de Regresión Logística
Vamos a crear un modelo de regresión logística, el cual nos permitirá clasificará los registros en anomalía y no anomalía, lo que se busca explorar es la eficiencia del modelo, por lo que vamos a probar, generar y validar algunas métricas y observar la matriz de confusión la cual nos demostrará la cantidad de aciertos y fallos por cada una de las categorías o etiquetas a predecir.

Construimos y ejecutamos nuestro modelo

```
def ConstruirYEntrenarRegresionLogistica(_c = 1.0, _solver = 'lbfgs', _max_iter = 350, _multi_class ='multinomial'):
    start_time = time()
    logisticRegr = LogisticRegression(C=_c, solver=_solver, max_iter=_max_iter, multi_class=_multi_class)
    logisticRegr.fit(train_, train_lbl)
    elapsed_time = time() - start_time
    print("Tiempo de ejecución: ",convert(elapsed_time))

    return logisticRegr
```

```
logisticRegr = ConstruirYEntrenarRegresionLogistica()
```

```
Tiempo de ejecución:  0:00:01
```

figura 4.10.1.1

Validamos nuestro modelo y graficamos nuestra matriz de confusión.

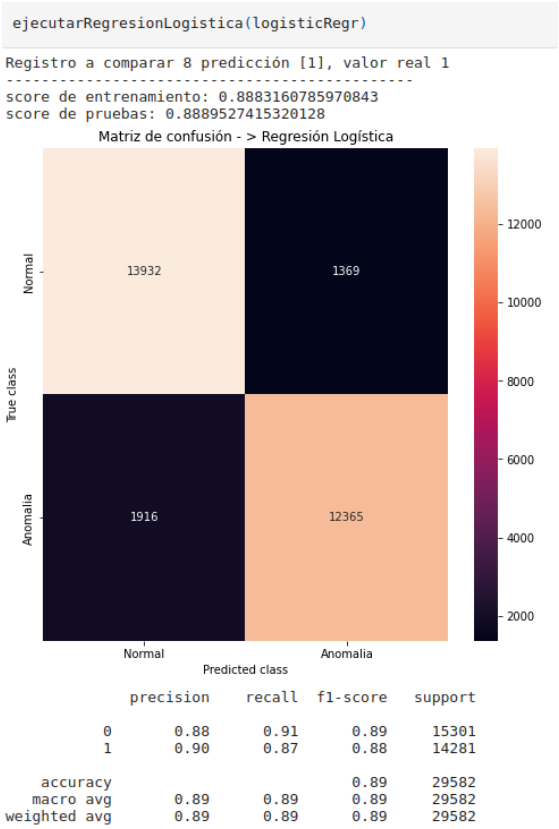
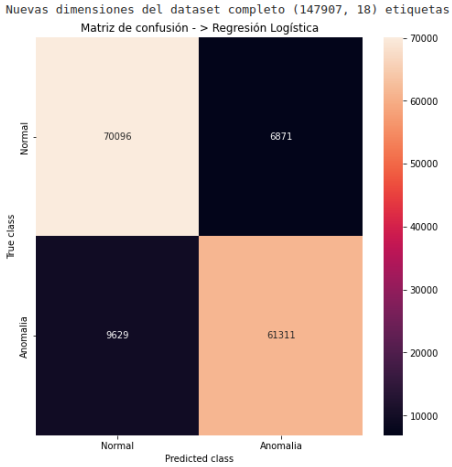


figura 4.10.1.1

En la imagen anterior, podemos validar que tenemos un modelo bastante aceptable, con un score de entrenamiento: 88.83% y score de pruebas: 88.9%, una precisión del 88% clasificando datos no anómalos y un 90% clasificando datos anómalos, podemos seguir explorando las métricas, pero en resumen tenemos un modelo con grandes capacidades de predicción, muy rápido para entrenar y ser predecir. Ahora vamos a ejecutar ya no el entrenamiento, si no que, vamos a pasarle el modelo completo para ver el resultado final con los datos tanto de pruebas como entrenamiento.

```
datosCompletos = np.concatenate((train_, test_), axis=0)
datosCompletoslbl = np.concatenate((train_lbl, test_lbl), axis=0)

print("Nuevas dimensiones del dataset completo {0} etiquetas {1}".format(datosCompletos.shape, datosCompletoslbl.shape))

probarModeloDeRegresion(logisticRegr, datosCompletos, datosCompletoslbl)
```

```
Nuevas dimensiones del dataset completo (147907, 18) etiquetas (147907,)
```



	precision	recall	f1-score	support
0	0.88	0.91	0.89	76967
1	0.90	0.86	0.88	70940
accuracy			0.89	147907
macro avg	0.89	0.89	0.89	147907
weighted avg	0.89	0.89	0.89	147907

figura 4.10.1.2

Tenemos un modelo que parece que sabe clasificar de forma correcta los datos actuales, tanto los usados en su entrenamiento, como los datos que se usó para pruebas.

Experimentando con un modelo de Random Forest

Al igual que el punto anterior, se busca validar la eficiencia del modelo, mediante el uso de los mismos datos de entrenamiento y pruebas.

Construimos y ejecutamos nuestro modelo.

```
def construiYEntrenarModeloRandomForest():
    start_time = time()
    model = RandomForestClassifier(n_estimators=100,
                                   class_weight="balanced",
                                   bootstrap = True,
                                   max_features = 'sqrt',
                                   verbose=1,
                                   max_depth=6,
                                   oob_score=True,
                                   random_state=50,
                                   n_jobs = 10)


    model.fit(train_, train_lbl)
    elapsed_time = time() - start_time
    print("Tiempo de ejecución: ",convert(elapsed_time))
    return model
```

```
model = construiYEntrenarModeloRandomForest()
```

```
[Parallel(n_jobs=10)]: Using backend ThreadingBackend with 10 concurrent workers.
[Parallel(n_jobs=10)]: Done  30 tasks      | elapsed:    0.7s
[Parallel(n_jobs=10)]: Done 100 out of 100 | elapsed:    2.2s finished
Tiempo de ejecución:  0:00:02
```

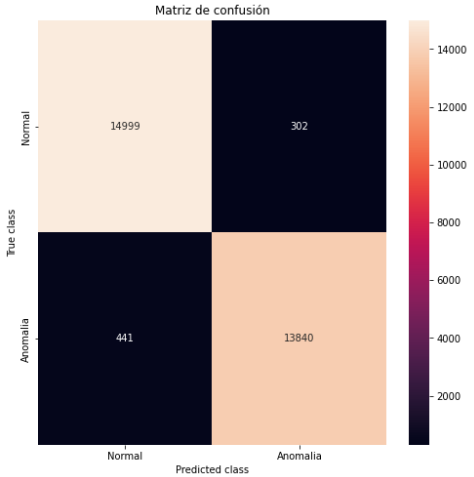
figura 4.10.2.1

Podemos notar que este es un modelo que dura un poco más en su entrenamiento, con relación al modelo de regresión logística y desde el punto de vista de uso de recursos, pareciera que requiere un alto consumo de uso de procesamiento para ejecutarse, pero vamos a proceder a probar a probar el modelo como tal.

Validamos nuestro modelo y graficamos nuestra matriz de confusión.

```
pred_y = model.predict(test_)
mostrar_resultados(test_lbl, pred_y)

roc_value = roc_auc_score( test_lbl, pred_y)
print("Calculo de roc auc: ", roc_value)
```

```
[Parallel(n_jobs=10)]: Using backend ThreadingBackend with 10 concurrent workers.
[Parallel(n_jobs=10)]: Done  30 tasks      | elapsed:    0.0s
[Parallel(n_jobs=10)]: Done 100 out of 100 | elapsed:    0.0s finished
```




```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98     15301
           1       0.98      0.97      0.97     14281

    accuracy                           0.97     29582
   macro avg       0.98      0.97      0.97     29582
weighted avg       0.97      0.97      0.97     29582

Calculo de roc auc:  0.9746912687317132
```

figura 4.10.2.2

A pesar de que se toma un poco más de tiempo en su entrenamiento, podemos validar que tenemos buenas métricas en este modelo, tenemos una precisión del 97% para no anómalos y 98% para anómalos, con un roc de 97%, pareciera que los datos se adaptan muy bien a los modelos utilizados.

Experimentando con un modelo de máquinas de vector soporte (svm)

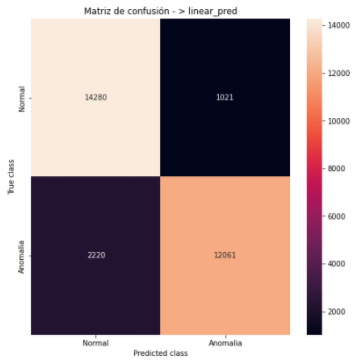
Este es un modelo que es usado con fines demostrativos únicamente, pero es un modelo que se ocupa altos niveles de procesamiento y a pesar de que logra su objetivo, sus tiempos de entrenamientos con la cantidad de datos suministrada y que consideramos baja, es muy lenta, pero igual vamos a proceder a construir y demostrar su eficiencia.

Construimos y ejecutamos a nuestro modelo, vamos a hacer diferentes pruebas con diferentes tipos de kernel: **linear, poly, sigmoid, rbf**. en cada uno de ellos, se obtienen tiempos muy elevados de entrenamiento y procesamiento, por lo que la eficiencia de construcción o reentrenamiento del modelo no es nada aceptable.

```
def EjecutarMaquinaDeSoporte(kenel = 'linear', decision_function_shape='ovo',degree=3, gamma='auto'):
    start_time = time()
    modelo = svm.SVC(kernel= kenel,
                     degree=degree,
                     C=1,
                     gamma=gamma,
                     decision_function_shape= decision_function_shape).fit(train_, train_lbl)

    prediccion_ = modelo.predict(test_)
    presicion_ = modelo.score(test_, test_lbl)

    print("Precisión usando el kernel {0}: {1}".format(kenel, presicion_))

    mostrar_resultados(test_lbl, prediccion_, "Matriz de confusión - > {0}_pred".format(kenel))
    elapsed_time = time() - start_time
    print("Tiempo de ejecución: ",convert(elapsed_time))
```

```
EjecutarMaquinaDeSoporte()
```

```
Precisión usando el kernel linear: 0.8904401325130147
```



	precision	recall	f1-score	support
0	0.87	0.93	0.90	15301
1	0.92	0.84	0.88	14281
accuracy			0.89	29582
macro avg	0.89	0.89	0.89	29582
weighted avg	0.89	0.89	0.89	29582

```
Tiempo de ejecución:  0:02:25
```

figura 4.10.3.1

Se puede observar que, para entrenar y probar este modelo, se consume más de 2 minutos en esta labor, si lo comparamos con los dos anteriores, donde sus tiempos

de entrenamientos no sobrepasaron los 2 segundos, podríamos decir que este no es un algoritmo que recomendamos, a pesar de sus buenos resultados, estos resultados lo podemos obtener con otro tipo de modelos en un menor tiempo.

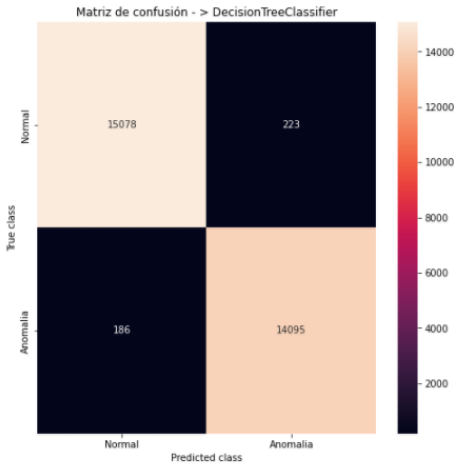
Experimentando con un modelo de Árboles de decisión

Los árboles de decisión son buenos modelos de predicción, aunque como todo tienen deficiencia y entre esas tenemos que son muy inestables, fáciles de que tengas sesgos. Aun así, vamos a proceder a probar su eficiencia.

```
def ejecutarArbolDeClasificacion():
    clf = DecisionTreeClassifier()
    clf = clf.fit(train_, train_lbl)
    train_acc = clf.score(train_, train_lbl)
    test_acc = clf.score(test_, test_lbl)
    y_pred = clf.predict(test_)
    print("score train {0} score test {1}".format(train_acc,test_acc))
    mostrar_resultados(test_lbl, y_pred, "Matriz de confusión - > DecisionTreeClassifier")
```

```
ejecutarArbolDeClasificacion()
```

```
score train 0.9990534544686246 score test 0.9861740247447772
```



	precision	recall	f1-score	support
0	0.99	0.99	0.99	15301
1	0.98	0.99	0.99	14281
accuracy			0.99	29582
macro avg	0.99	0.99	0.99	29582
weighted avg	0.99	0.99	0.99	29582

figura 4.10.4.1

El modelo tiene resultados muy buenos, pero es un modelo que es muy propenso al sesgo, a pesar de todo, casi que logró predecir muy bien todos los resultados, tanto los del set de entrenamiento como los de pruebas que nunca había visto al entrenar.

Experimentando con un modelo de Red neuronal usando Keras

Las redes neuronales a pesar de que son más complejas, su nivel de crecimiento y ajustes son exponenciales, son excelentes modelos de predicción y clasificación y permite mucha flexibilidad a la hora de construir modelos de inteligencia artificial, claro está que dependiendo de la red, es difícil de explicar su compleja de resolución, pero en los casos que se logran resolver las necesidades del negocio, lo

hacen muy, se va a proceder a ejecutar varios experimentos, haciendo uso de nuestro modelo de red neuronal.

Construyendo el modelo.

```
def construirModelo(tipoActivacion = 'sigmoid',
                    kernel='glorot_uniform',
                    bias = 'zeros',
                    valoresCapas = [32,64,128,64],
                    tipoCapaSalida = 'softmax',
                    activarBatchNormalizacion = False,
                    dropout = None,
                    cantidadCapasDeEntrada = 19,
                    cantidadCapasDeSalida = 2):

    model = Sequential()
    model.add(Dense(units=cantidadCapasDeEntrada,input_shape=(cantidadCapasDeEntrada,)))

    if(activarBatchNormalizacion):
        model.add(BatchNormalization())

    for valor in range(len(valoresCapas)):
        model.add( Dense(valoresCapas[valor],
                         activation=tipoActivacion,
                         bias_initializer=bias,
                         kernel_initializer=kernel))

        if(dropout):
            model.add(Dropout(dropout))

    model.add( Dense(cantidadCapasDeSalida, activation=tipoCapaSalida))
    return model
```

figura 4.10.5.1

Compilación del modelo y visualizamos la red que vamos a usar para nuestros experimentos.

```
def construirModeloAUsarEnreDNeuronal():
    model = construirModelo(tipoActivacion = 'ReLU',
                            valoresCapas = [32,64,128, 128,64],
                            cantidadCapasDeEntrada=train_.shape[1])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model

model = construirModeloAUsarEnreDNeuronal()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 18)                342
_________________________________________________________________
dense_1 (Dense)              (None, 32)                608
_________________________________________________________________
dense_2 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_3 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 130
=================================================================
Total params: 36,280
Trainable params: 36,280
Non-trainable params: 0
```

figura 4.10.5.1

Se puede observar que se va a usar una red neuronal, con 7 capas, la primera capa contiene la cantidad de columnas que tiene nuestro set de datos, lo que significa que cada valor de cada columna está respaldado por una neurona de entrada, luego tenemos 5 capas ocultas y una capa de salida con dos neuronas, las cuales representa cada una, si es o no anomalía.

Vamos a entrenar nuestro modelo y evaluar sus resultados, para ello vamos a indicarle que use un 64 el tamaño del lote y 40 épocas de entrenamiento.

```
batch_size = 64
epochs=20

history1 = model.fit(train_, train_lbl, batch_size=batch_size, epochs=epochs, validation_split = 0.2 , verbose=0)
print(model.evaluate(test_, test_lbl))
```

```
925/925 [==============================] - 1s 845us/step - loss: 0.0553 - accuracy: 0.9799
[0.05530357360839844, 0.9798864126205444]
```

figura 4.10.5.2

En la imagen anterior, podemos observar una pérdida bastante baja en su evaluación, lo que resulta positivo y una precisión o accuracy muy aceptable de casi un 98%, esto se traduce a un modelo muy bueno, pero, además, vamos a correr diferentes pruebas sobre nuestro modelo entrenado, con el fin de validar si es capaz de predecir de forma correcta los datos que se le pasen, para ello, vamos llamar a un método que usa el modelo entrenado y valida los registros que le pasemos.

```
prediccionUsandoElModelo(model)
prediccionUsandoElModelo(model, 72, 320)
prediccionUsandoElModelo(model, 107, 124)
prediccionUsandoElModelo(model, 678, 560)
```

```
Probabilidad de no ser anomalía 0 - Probabilidad de ser anomalía 1 -> Etiqueta real Anomalia
Probabilidad de no ser anomalía 1 - Probabilidad de ser anomalía 0 -> Etiqueta real Normal
Probabilidad de no ser anomalía 1 - Probabilidad de ser anomalía 0 -> Etiqueta real Normal
Probabilidad de no ser anomalía 0 - Probabilidad de ser anomalía 1 -> Etiqueta real Anomalia
Probabilidad de no ser anomalía 1 - Probabilidad de ser anomalía 0 -> Etiqueta real Anomalia
Probabilidad de no ser anomalía 1 - Probabilidad de ser anomalía 0 -> Etiqueta real Normal
Probabilidad de no ser anomalía 0 - Probabilidad de ser anomalía 1 -> Etiqueta real Anomalia
Probabilidad de no ser anomalía 1 - Probabilidad de ser anomalía 0 -> Etiqueta real Normal
```

figura 4.10.5.3

En la imagen anterior, observamos los resultados de evaluar nuestro modelo, con los registros que le pasamos, por ejemplo, la dos primeros resultados, son del registro 0 y 8 de nuestro set de datos. vemos que dice: Probabilidad de ser

anomalía es 1 y la etiqueta real es efectivamente anomalía y así sucesivamente podemos valorar los demás registros.

En la siguiente imagen, vamos a ver la evolución del entrenamiento de nuestro modelo, por época de entrenamiento, podemos notar como va mejorando el entrenamiento, hasta lograr un buen modelo, en el primer gráfico vemos el accuracy y en el segundo gráfico, como va bajando la pérdida o Loss.

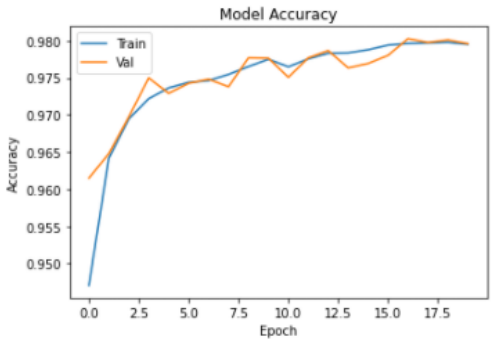


figura 4.10.5.4

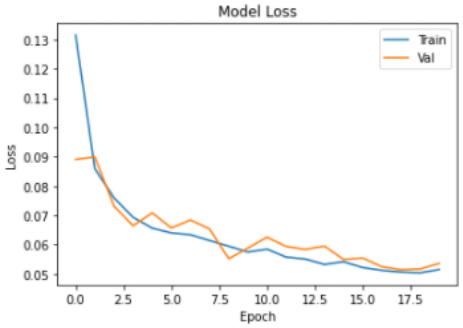


figura 4.10.5.5

Como parte de nuestro proceso de prueba del modelo, vamos a proceder a probar el modelo con solo datos anómalos y luego lo probamos con solo datos no anómalos, con el fin de validar el nivel de predicción del modelo, para cada tipo.

Creamos nuestro set de datos con solo datos anómalos y con datos normales.

```
df_EntrenarRedNeuronal = crearDatasetNuevo(data_km, data_attack_flag)

df_EntrenarRedNeuronalAnomalia = df_EntrenarRedNeuronal[ df_EntrenarRedNeuronal['labels'] == 1]
df_EntrenarRedNeuronalNoAnomalia = df_EntrenarRedNeuronal[ df_EntrenarRedNeuronal['labels'] == 0]

df_EntrenarRedNeuronalAnomalia_lb = df_EntrenarRedNeuronalAnomalia['labels']
df_EntrenarRedNeuronalNoAnomalia_lb = df_EntrenarRedNeuronalNoAnomalia['labels']

df_EntrenarRedNeuronalAnomalia = df_EntrenarRedNeuronalAnomalia.drop(['labels'], axis=1)

df_EntrenarRedNeuronalNoAnomalia = df_EntrenarRedNeuronalNoAnomalia.drop(['labels'], axis=1)

df_EntrenarRedNeuronal = df_EntrenarRedNeuronal.drop(['labels'], axis=1)

print("Cantidad de datos anómalos: {0} - Cantidad de datos NO anómalos {1}".format(df_EntrenarRedNeuronalAnomalia.shape[0],
                                                                                   df_EntrenarRedNeuronalNoAnomalia.shape[0])
     )
```

```
Cantidad de datos anómalos: 70940 - Cantidad de datos NO anómalos 76967
```

figura 4.10.5.6

Tenemos 70,940.00 registros anómalos contra 76,967.00 registros no anómalos o normales.

Evaluamos nuevamente nuestro modelo ya entrenado y validamos los resultados obtenidos.

```
print("Evaluamos con todos los datos", model.evaluate(reduced_dataNoAnomalias, df_EntrenarRedNeuronalNoAnomalia_lb))
print("Evaluamos con solo datos No Anómalos", model.evaluate(reduced_dataNoAnomalias, df_EntrenarRedNeuronalNoAnomalia_lb))
print("Evaluamos con solo datos Anómalos", model.evaluate(reduced_dataAnomalias, df_EntrenarRedNeuronalAnomalia_lb))
```

```
2406/2406 [==============================] - 2s 803us/step - loss: 0.0573 - accuracy: 0.9788
Evaluamos con todos los datos [0.05731293559074402, 0.9787701368331909]
2406/2406 [==============================] - 2s 787us/step - loss: 0.0573 - accuracy: 0.9788
Evaluamos con solo datos No Anómalos [0.05731293559074402, 0.9787701368331909]
2217/2217 [==============================] - 2s 794us/step - loss: 0.0443 - accuracy: 0.9821
Evaluamos con solo datos Anómalos [0.04433780163526535, 0.9821116328239441]
```

figura 4.10.5.7

Podemos observar que obtenemos excelentes resultados tanto en precisión como en pérdida; Una pérdida baja y una presión alta, lo que es positivo para nuestro objetivo, se comporta muy bien nuestro modelo con todos los datos, con solo datos anómalos y si le pasamos solo datos normales o no anómalos.

Ahora vamos hacer nuevamente una prueba, pero esta vez, con solo registros que tenga tiempos de duración menores a 2 segundos, según el set de datos original, para ello vamos a construir nuevamente nuestro set de datos, pero esta vez, con registros menores a los 2 segundos.

```
def contruirNuevoModeloDeAcuerdoAlTiempoDeDuracion(tiempoDuracion_ = 0):
    dt_dos_sec_ = dataDeTrabajo.copy()
    dt_dos_sec_['labels'] = data_attack_flag
    dt_dos_sec_ = dt_dos_sec_[(dt_dos_sec_['duration'] <= tiempoDuracion_)]
    dt_dos_sec_lb_ = dt_dos_sec_['labels']
    dt_dos_sec_ = dt_dos_sec_.drop(['labels'], axis=1)
    dt_dos_sec_=scaler.transform(dt_dos_sec_)
    return crearDatasetNuevo(dt_dos_sec_.copy() , dt_dos_sec_lb_.copy() )
```

```
tiempoDuracion_ = 2

df_EntrenarRedNeuronal = contruirNuevoModeloDeAcuerdoAlTiempoDeDuracion(tiempoDuracion_)

df_EntrenarRedNeuronalAnomalia = df_EntrenarRedNeuronal[ df_EntrenarRedNeuronal['labels'] == 1]
df_EntrenarRedNeuronalNoAnomalia = df_EntrenarRedNeuronal[ df_EntrenarRedNeuronal['labels'] == 0]

df_EntrenarRedNeuronalAnomalia_lb = df_EntrenarRedNeuronalAnomalia['labels']
df_EntrenarRedNeuronalNoAnomalia_lb = df_EntrenarRedNeuronalNoAnomalia['labels']

df_EntrenarRedNeuronalAnomalia = df_EntrenarRedNeuronalAnomalia.drop(['labels'], axis=1)

df_EntrenarRedNeuronalNoAnomalia = df_EntrenarRedNeuronalNoAnomalia.drop(['labels'], axis=1)

df_EntrenarRedNeuronal = df_EntrenarRedNeuronal.drop(['labels'], axis=1)

print("Cantidad de datos anómalos: {0} - Cantidad de datos NO anómalos {1}".format(df_EntrenarRedNeuronalAnomalia.shape[0],
                                                                                   df_EntrenarRedNeuronalNoAnomalia.shape[0])
      )
```

```
Cantidad de datos anómalos: 66720 - Cantidad de datos NO anómalos 71165
```

figura 4.10.5.8

Evaluamos nuevamente nuestro modelo con solo datos con tiempos menos a 2 segundos y validamos sus resultados.

```
print("Evaluamos con todos los datos", model.evaluate(reduced_dataNoAnomalias, df_EntrenarRedNeuronalNoAnomalia_lb))
print("Evaluamos con solo datos No Anómalos", model.evaluate(reduced_dataNoAnomalias, df_EntrenarRedNeuronalNoAnomalia_lb))
print("Evaluamos con solo datos Anómalos", model.evaluate(reduced_dataAnomalias, df_EntrenarRedNeuronalAnomalia_lb))
```

```
Datos con tiempos menores o igual a: 2 segundos
2224/2224 [==============================] - 2s 745us/step - loss: 0.0542 - accuracy: 0.9788
Evaluamos con todos los datos [0.054155752062797546, 0.9788098335266113]
2224/2224 [==============================] - 2s 826us/step - loss: 0.0542 - accuracy: 0.9788
Evaluamos con solo datos No Anómalos [0.054155752062797546, 0.9788098335266113]
2085/2085 [==============================] - 2s 868us/step - loss: 0.0404 - accuracy: 0.9832
Evaluamos con solo datos Anómalos [0.04042049124836922, 0.9832284450531006]
```

figura 4.10.5.9

De forma positiva, seguimos obteniendo excelentes resultados, lo que podemos traducirlo como un modelo con la capacidad de evaluar tráfico de red y detectar anomalías en los primeros segundos del ataque.

Experimentando con un modelo de Red neuronal usando Keras y nuevos datos generados por el equipo.

Como parte de un nuevo análisis sobre nuestros datos, vamos a realizar el siguiente experimento basado en los siguientes datos. Se sabe que tenemos:

1- Cantidad de registros con duración mayor a 0: ***13,281.00*** de ***135,937.00*** lo que representan un 9.77% de datos con duración superior a 0, lo que es una representación muy baja de ese tipo de datos. por lo que vamos a proceder a eliminar la columna duración. Esto implica entrenar un nuevo modelo de PCA, para reducir nuevamente nuestro dataset original, pero esta vez sin la columna duración.

2- Cantidad de registros con src_bytes mayores a 0: ***87,668.00*** de ***135,937.00*** lo que representa un 64.49% de valores mayores a 0.

3- Cantidad de registros con dst_bytes mayores a 0: ***69,889.00*** de ***135,937.00*** lo que representa un 51.41% de valores mayores a 0.

Con los últimos dos puntos, se va a proceder a generar nuevos valores, reduciendo estos valores porcentualmente, lo que nos permitiría generar una mayor cantidad de registros y de esta forma, valorar cómo se comporta el modelo, con estos nuevos datos, cuando el valor de bytes para el mismo registro es alterado.

Vamos a reducir nuestros datos porcentualmente, en este caso vamos a reducir los registros mayor a 0 en bytes en un 90, 85, 75, 65 y 55 %.

```
def reducirPorcentualmeteColumnasBytes(reducirEnUn = 90):
    reducirEnUn = round((reducirEnUn / 100),2)
    data_lab_ = data_lab.copy()
    data_lab_['src_bytes'] = data_lab_.src_bytes.map(lambda a: 0 if a <= 0 else (a - (a* reducirEnUn ) ))
    data_lab_['dst_bytes'] = data_lab_.dst_bytes.map(lambda a: 0 if a <= 0 else (a - (a* reducirEnUn) ))

    return data_lab_
```

```
nuevosDatosLaboratorio = pd.concat([reducirPorcentualmeteColumnasBytes(),
                          reducirPorcentualmeteColumnasBytes(reducirEnUn = 85),
                          reducirPorcentualmeteColumnasBytes(reducirEnUn = 70),
                          reducirPorcentualmeteColumnasBytes(reducirEnUn = 65),
                          reducirPorcentualmeteColumnasBytes(reducirEnUn = 55)])
```

figura 4.10.6.1

Quitamos registros duplicados y las columnas que no vamos a usar para entrenar nuestro modelo y validamos nuestras nuevas dimensiones.

```
nuevosDatosLaboratorio = nuevosDatosLaboratorio.drop(['duration'], axis=1)

nuevosDatosLaboratorio = nuevosDatosLaboratorio.drop_duplicates()

nuevosLabels = nuevosDatosLaboratorio['labels']
nuevosDatosLaboratorio = nuevosDatosLaboratorio.drop(['labels'], axis=1)

nuevosDatosLaboratorio = nuevosDatosLaboratorio.reset_index()
del nuevosDatosLaboratorio['index']
```

```
print( "Total de registros con datos nuevos generados {0} labels {1} y cantidad datos originales {2} labels {3}".format(
                        nuevosDatosLaboratorio.shape,
                        nuevosLabels.shape,
                        dataDeTrabajo.shape,
                        data_attack_flag.shape))
```

```
Total de registros con datos nuevos generados (564539, 24) labels (564539,) y cantidad datos originales (147907, 25) labels (147907,)
```

figura 4.10.6.2

Se puede observar que se pasó de ***147,907.00*** registros a ***564,539.00*** registros, una diferencia de ***416,632.00*** registros nuevos generados mediante el escalado porcentual de las dos columnas byte, lo que equivale a casi un 73% más de registros nuevos.

Se vuelven a solicitar a PCA que genere un nuevo set de datos que vuelva a explicar el 90% de nuestros datos, pero esta vez, no se incluye la columna duración, lo que nos permite reducir las dimensiones a de 18 del modelo anterior a 17.

Además, para esta nueva prueba, se usarán menos datos para entrenamiento y más datos para pruebas, se usará un 70% de registros para entrenamiento y 30% de registros para probar nuestro modelo. Además, volvemos a construir nuestro modelo, pero ahora con una capa de entrada de 17 neuronas.

```
scaler.fit(nuevosDatosLaboratorio)
data_km=scaler.transform(nuevosDatosLaboratorio)

pca_, reduced_data, explained_variance = construirYTransformarDatosConPCA(data_km, porcentaje_ = 90)
print(explained_variance)
```

```
[(0, '12.65%'), (1, '11.45%'), (2, '7.63%'), (3, '6.05%'), (4, '5.54%'), (5, '5.0%'), (6, '4.29%'), (7, '4.26%'), (8, '4.18%'),
```

```
train_, test_, train_lbl, test_lbl = train_test_split( reduced_data, nuevosLabels.values, test_size=0.7, random_state=0)
```

Entrenamos nuevamente nuestra red neuronal con nuevos datos de laboratorio

```
model = construirModelo(tipoActivacion = 'ReLU', cantidadCapasDeEntrada=train_.shape[1])
model = construirModeloAUsarEnreDNeuronal()
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_52 (Dense)             (None, 17)                306
_________________________________________________________________
dense_53 (Dense)             (None, 32)                576
_________________________________________________________________
dense_54 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_55 (Dense)             (None, 128)               8320
_________________________________________________________________
dense_56 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_57 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_58 (Dense)             (None, 2)                 130
=================================================================
Total params: 36,212
Trainable params: 36,212
Non-trainable params: 0
```

figura 4.10.6.3

Se entrena el nuevo modelo, con los nuevos datos generados, sin la columna duración y se valida su efectividad.

```
history1 = model.fit(train_, train_lbl, batch_size=batch_size, epochs=epochs, validation_split = 0.2 , verbose=0)
print(model.evaluate(test_, test_lbl))


valorAComparar = 8 ## debe de dar 1
predictions = model.predict(test_[valorAComparar].reshape(1,-1))
print("Normal {0} - Anomalía {1} -> lblReal {2}"
      .format(round(predictions[0][0]), round(predictions[0][1]), test_lbl[valorAComparar]))
valorAComparar = 0 ## debe de dar 0
predictions = model.predict(test_[valorAComparar].reshape(1,-1))
print("Normal {0} - Anomalía {1} -> lblReal {2}"
      .format(round(predictions[0][0]), round(predictions[0][1]), test_lbl[valorAComparar]))
```

```
12350/12350 [==============================] - 10s 827us/step - loss: 0.0560 - accuracy: 0.9796
[0.05603606626391411, 0.9795560240745544]
Normal 0 - Anomalía 1 -> lblReal 1
Normal 1 - Anomalía 0 -> lblReal 0
```

figura 4.10.6.4

Se puede notar, que el nuevo modelo entrenado con un 70% de los datos, a la hora de validarlo, arroja excelentes resultados en su precisión y pérdida. lo que nos demuestra que seguimos teniendo un excelente modelo para lograr predecir registros anómalos.

De igual forma, válida las gráficas de entrenamiento, tanto los resultados de pérdida o Loss como de precisión o accuracy.

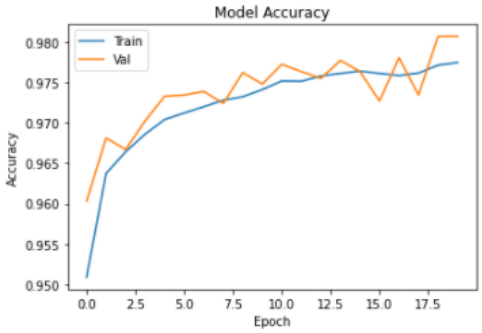


figura 4.10.6.5

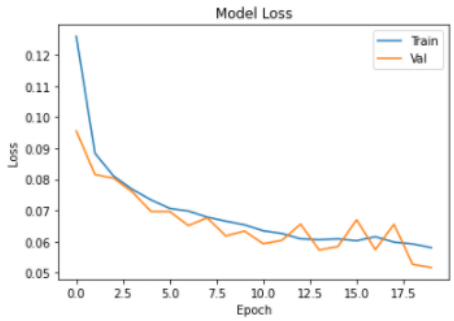


figura 4.10.6.6

Se ejecuta una prueba rápida del nuevo modelo, con el fin de determinar su efectividad.

```
prediccionUsandoElModelo(model)
prediccionUsandoElModelo(model, 72, 320)
prediccionUsandoElModelo(model, 107, 124)
prediccionUsandoElModelo(model, 678, 560)
```

```
Probabilidad de no ser anomalía 0 - Probabilidad de ser anomalía 1 -> Etiqueta real Anomalia
Probabilidad de no ser anomalía 1 - Probabilidad de ser anomalía 0 -> Etiqueta real Normal
Probabilidad de no ser anomalía 0 - Probabilidad de ser anomalía 1 -> Etiqueta real Anomalia
Probabilidad de no ser anomalía 1 - Probabilidad de ser anomalía 0 -> Etiqueta real Normal
Probabilidad de no ser anomalía 1 - Probabilidad de ser anomalía 0 -> Etiqueta real Normal
Probabilidad de no ser anomalía 0 - Probabilidad de ser anomalía 1 -> Etiqueta real Anomalia
Probabilidad de no ser anomalía 1 - Probabilidad de ser anomalía 0 -> Etiqueta real Normal
Probabilidad de no ser anomalía 1 - Probabilidad de ser anomalía 0 -> Etiqueta real Normal
```

figura 4.10.6.7

Experimentando con los nuevos datos, usando una regresión logística.

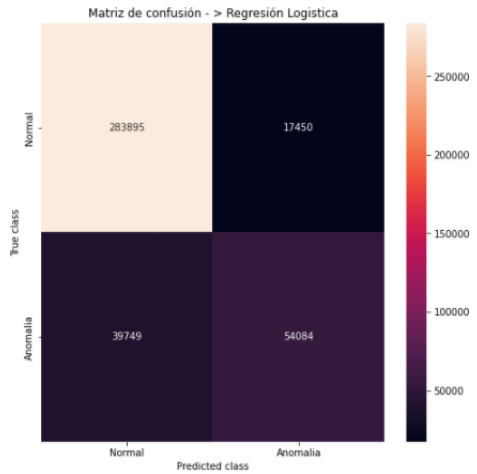
Como parte del proceso de selección del mejor modelo, para nuestro sistema de detección de anomalías, vamos a volver a validar el algoritmo anteriormente

ejecutado de regresión logística, para validar su eficiencia con los nuevos datos de entrenamiento y pruebas usados anteriormente para probar nuestra red neuronal.

Entrenamos y validamos las métricas del nuevo modelo de regresión logística y su matriz de confusión.

```
logisticRegr.fit(train_, train_lbl)

valorAComparar = 8
print("Predicción {0}, valor real {1}".format(
                    logisticRegr.predict(test_[valorAComparar].reshape(1,-1)),test_lbl[valorAComparar]))

print("score de entrenamiento: {0}".format(logisticRegr.score(train_, train_lbl)))
print("score de pruebas: {0}".format(logisticRegr.score(test_, test_lbl)))

y_pred = logisticRegr.predict(test_)

mostrar_resultados(test_lbl, y_pred, "Matriz de confusión - > Regresión Logistica")
```

```
Predicción [1], valor real 1
score de entrenamiento: 0.8559762873388796
score de pruebas: 0.8552576307385532
```



	precision	recall	f1-score	support
0	0.88	0.94	0.91	301345
1	0.76	0.58	0.65	93833
accuracy			0.86	395178
macro avg	0.82	0.76	0.78	395178
weighted avg	0.85	0.86	0.85	395178

figura 4.10.7.1

Podemos notar, que en este caso, el modelo bajo su rendimiento al quitar la columna duración y aumentar la data, si recordamos, tenemos un modelo primer modelo que arrobo hasta un 90% en precisión de la categoría anomalía y en este caso, bajó a un 76%, lo que nos demuestra que empeoró su resultado, además los score de entrenamiento bajaron de casi un 89% del primero modelo de regresión entrenado a un 85%.

4.11.- CONCLUSIÓN: DETECCIÓN FUNCIONA CON RED NEURONAL

Como parte del proceso de análisis, limpieza y experimentación con diferentes modelos de clasificación y predicción, se logra determinar que el mejor modelo es respaldado por las redes neuronales, arrojando en diferentes escenarios, buenas métricas de resolución; apoyado de la data suministrada y generada, se logran buenos resultados; las predicciones del modelo, logró resolver satisfactoriamente el problema planteado; con precisiones mayores a 90% tanto en datos de entrenamiento, pruebas, registros con tiempos menores a 2 segundos, con datos anómalos o no anómalos, sin la columna duración y con nuevos datos generados ficticiamente.

Como parte del proceso, se logró determinar, que el modelo es capaz de detectar anomalía en los datos, ya sean generados con ventanas de 2 segundos de tiempo o superior, demostrado haciendo de uso de este valor como parte del modelo o quitando dicha columna, el modelo es capaz de predecir anomalías de acuerdo a los patrones en los datos suministrados.

Este valor de 2 segundos de tiempo se ha elegido de manera arbitraria por que es el tiempo en el cual se recoge los datos de volúmenes de conexiones (número de conexiones por periodo de 2 segundos) en el dataset NSL-KDD como lo describe en este trabajo [7]. Significa que cada 2 segundos tenemos nuevos datos generalizados y que no nos sirve predecir ataques en intervalos más cortos.

5. RESUMEN Y RECOMENDACIONES A LA EMPRESA

El método que se aplica a la detección de anomalías en la red es de elegir unos ataques representativos de las más grandes categorías de ataques para hacer una prueba de concepto con la empresa. Empezamos explorando los datos proporcionados por la empresa obteniendo resultados no satisfactorios por la falta de volumen y de calidad de los datos del dataset. Decidimos elegir un nuevo dataset público NSL- KDD, volver a hacer el estudio y conseguimos un modelo red neuronal capaz de detectar en tiempo real 4 tipos de ataque con buenas métricas de calidad. Esto nos permite, al detectar la anomalía, diseñar una estrategia para parar el ataque antes de que haga daños a la red.

Para esta prueba de concepto, recomendamos desplegar nuestro algoritmo entre dos smart firewalls. El primero comprueba que el entrante esté bien en la whitelist y si es positivo deja pasar el tráfico. Se prepara la metadata para procesarla con nuestra red neuronal que detecta si hay anomalía o no. Después puede pasar al segundo firewall clásico si no detecta anomalía. Si detecta una anomalía, deja la conexión en cuarentena para poder comprobar con otros métodos si es un verdadero positivo o falso positivo, averiguar el riesgo y decidir si permitir o no la conexión. Determinamos que la manera óptima de analizar el de hacerlo en multithreading para una mayor eficacia y poder procesar volúmenes de datos consecuentes sin afectar la velocidad de la red. Recomendamos de colectar los datos cada 2 segundos por 2 razones como está explicado en el trabajo [7]. 2 segundos es el tiempo mínimo para que puedan destacarse los patrones de un ataque o sea si se detecta dentro de esta ventana, y que se corta la conexión, se puede proteger la red. También se puede analizar los datos mientras los nuevos lleguen lo que va a limitar la necesidad de acumular series de datos en el tiempo para la detección sin tener que acumular cantidades de datos y ocupar más recursos de hardware.

Este proceso se puede repetir con otros datasets y combinar las soluciones de seguridad para defender la empresa contra otros tipos de ataques y seguir mejorando el sistema con la evolución de los ataques.

GLOSARIO.

A)Tipo de Ataques

Virus: programa autorreplicante que se adhiere a un programa existente e infecta un sistema sin permiso o conocimiento del usuario.

Gusano (Worms): programa autorreplicante que se propaga sin usar archivos infectados; normalmente se propaga a través de los servicios de red en los ordenadores o a través de email.

Troyano (Trojans): una pieza de programa hecha para realizar una determinada acción benigna, pero en realidad realizar código diferente con fines maliciosos.

Spyware. Malware instalado en un sistema informático sin permiso y/o conocimiento por parte del operador, con fines de espionaje y recopilación de información. Los Keylogger entran en esta categoría.

Adware. Malware que inyecta material publicitario no solicitado (por ejemplo, ventanas emergentes, banners, videos) en una interfaz de usuario, a menudo cuando un usuario está navegando por la web.

Ransomware. Malware diseñado para restringir la disponibilidad de los sistemas informáticos hasta que se pague una suma de dinero (rescate).

Rootkit. Una colección de software (a menudo) de bajo nivel diseñado para permitir el acceso u obtener el control de un sistema informático. ("Root" denota el nivel más poderoso de acceso a un sistema).

Puerta trasera (Backdoor). Un orificio intencional colocado en el perímetro del sistema para permitir futuros accesos que pueden eludir las protecciones perimetrales.

Bot. Variante de malware que permite a los atacantes tomar el control remoto y controlar máquinas. El nombre deriva de una abreviatura de robots, estos dispositivos también se conocen como zombies.

Botnet. Es una red de máquinas comprometidas (bot) que explota vulnerabilidades específicas de equipos, aplicaciones o software. El bootmaster normalmente empleaba el protocolo Internet Relay Chat (IRC) para administrar sus bots. Actualmente, también usan arquitecturas Peerto-Peer (P2P), ya que son más difíciles de rastrear y cerrar para las autoridades. Las botnets han demostrado ser herramientas ideales para enviar spam y lanzar ataques distribuidos de denegación de servicio (DDoS) entre muchos otros usos maliciosos

Explotar (Exploit). Una pieza de código o software que explota vulnerabilidades específicas en otras aplicaciones o marcos de software.

Sondas (**Probing).-**Las sondas de red suelen ser ataques que escanean las redes para recopilar información. El objetivo de esta recopilación de información es averiguar sobre la computadora y los servicios que están presentes en una red, así como detectar la posibilidad de ataque basado en vulnerabilidades conocidas.

IPSweep. Ataque probing que determina qué hosts están escuchando en las redes a través de un exploración de vigilancia.

Portsweep. Ataque probing que se utiliza para analizar qué puertos de un competidor especificado se abren en una red (o subred).

Escaneo *(Scanning)* Ataques que envían una variedad de solicitudes a los sistemas informáticos, con el objetivo de encontrar puntos débiles y vulnerabilidades, así como la recopilación de información.

Francotiradores (Sniffing). Observación y registro silenciosos del tráfico y los procesos de la red y en el servidor sin el conocimiento de los operadores de red.

Keylogger. Una pieza de hardware o software que (a menudo de forma encubierta) registra las teclas presionadas en un teclado o dispositivo de entrada de computadora similar.

Spam. Mensajes masivos no solicitados, generalmente con fines publicitarios. Por lo general, el correo electrónico, pero podría ser SMS o a través de un proveedor de mensajería (por ejemplo, WhatsApp). Constituyen alrededor del 60% del tráfico mundial de correo electrónico.

Ataque de inicio de sesión (*Login attack*). Múltiples intentos, generalmente automatizados, de adivinar credenciales para sistemas de autenticación, ya sea de manera de fuerza bruta o con credenciales robadas / compradas.
Adquisición de cuentas (*Account takeover* ATO). Obtener acceso a una cuenta que no es la suya, generalmente con fines de venta posterior, robo de identidad, robo monetario, etc. Por lo general, el objetivo de un ataque de inicio de sesión, pero también puede ser a pequeña escala y altamente dirigido (por ejemplo, spyware, ingeniería social).

Phishing. (**Masquerading**) Comunicaciones con un ser humano que pretende ser una entidad o persona de buena reputación para inducir la revelación de información personal o para obtener activos privados, por ejemplo, eludir el mecanismo de autenticación mediante el uso de ID de inicio de sesión y contraseñas robados. Un sitio web de phishing es un sitio web que intenta obtener la contraseña de su cuenta u otra información personal haciendo pensar que está en un sitio web legítimo.

Spear phishing. Phishing que está dirigido a un usuario en particular, haciendo uso de la información sobre ese usuario obtenida de fuentes externas. I

Ingeniería social: Es un método no técnico de ataque que engaña a una víctima mediante la persuasión o el uso de otras habilidades interpersonales para obtener información de autenticación o acceso a un sistema.

Discurso incendiario. Discurso discriminatorio, desacreditador o dañino dirigido a un individuo o grupo.

Denegación de servicio (DoS) y denegación de servicio distribuida (DDoS). En un ataque de denegación de servicio (DoS), el atacante envía un gran número de solicitudes de conexión o información a un objetivo. Se realizan tantas solicitudes que el sistema de destino se sobrecarga y no puede responder a las solicitudes legítimas de servicio que solicitan los usuarios, de esta manera se rompe la integridad del sistema. En un ataque distribuido de denegación de servicio (DDoS), se lanza un flujo coordinado de solicitudes contra un objetivo desde muchas ubicaciones al mismo tiempo. La mayoría de los ataques DDoS están precedidos por una fase de preparación en la que muchos sistemas, tal vez miles, se ven comprometidos. Las máquinas comprometidas se convierten en bots o zombies, que son dirigidas de forma remota por el atacante. Es un tipo de ataque que nos interesa particularmente para este trabajo.

Amenazas persistentes avanzadas (APT) Redes altamente dirigidas o ataque de host en el que un intruso sigiloso permanece intencionalmente sin ser detectado durante largos períodos de tiempo para robar y afectar los datos.
Vulnerabilidad de día cero. Una debilidad o error en diseño o implementación del software que es desconocido por el fabricante, lo que permite una posible explotación (llamada ataque de día cero) antes de que el fabricante tenga la oportunidad de solucionar el problema.

Ataques de desbordamiento de búfer (Buffer Overflow) Se produce cuando un programa intenta almacenar más datos en el búfer de los que su capacidad lo permite, los datos adicionales se desbordan en los búferes adyacentes y, por lo tanto, corrompe y sobrescriben los datos válidos que contienen estos búferes. Los ataques de desbordamiento de búfer aprovechan esta vulnerabilidad y colocan el código malicioso en el área de desbordamiento de búfer. Cuando se ejecuta el código de ataque este puede desencadenar acciones que pueden afectar al sistema completo.

Amenazas internas Es una acción tomada por un empleado de la organización cuyo efecto es potencialmente perjudicial para la organización. Estos pueden incluir acciones como la transferencia de datos no autorizados o el sabotaje de recursos.

Amenazas polimórficas. Las características de un ataque polimórfico es que este se transforma, cambiando su tamaño y otras características de archivos externos para eludir su detección e identificación.

B) Características DATA SET NSL-KDD

	Caracteristica	Descripción
1.	Duration	Duración de la conexión
2.	Protocol_type	Protocolo utilizado en la conexión.
3.	Service	Servicio de red de destino utilizado.
4.	Flag	Status of the connection – Normal or Error.
5.	Src_bytes	Número de bytes de datos transferidos de origen a destino en una sola conexión.
6.	Dst_bytes	Número de bytes de datos transferidos de destino a origen en una sola conexión.
7.	Land	Si las direcciones IP de origen y destino y los números de puerto son iguales, entonces, esta variable toma el valor 1 de lo contrario 0.
8.	Wrong_fragment	Número total de fragmentos erróneos a este respecto.
9.	Urgent	Número de paquetes urgentes en este sentido. Los paquetes urgentes son paquetes con el bit urgente activado.
10.	Hot	Número de paquetes urgentes en este sentido. Los paquetes urgentes son paquetes con el bit urgente activado Número de indicadores "hot" en el contenido como: entrar en un directorio del sistema, crear programas y ejecutar programas.
11.	Num_failed_logins	Recuento de intentos de inicio de sesión fallidos.

12.	Logged_in	Estado de inicio de sesión: 1 si ha iniciado sesión correctamente; 0 de lo contrario.
13.	Num_compromised	Número de condiciones "comprometidas".
14.	Root_shell	1 si se obtiene la cáscara de la raíz; 0 en caso contrario.
15.	Su_attempted	1 si se intenta o utiliza el comando "su root"; 0 de lo contrario.
16.	Num_root	Número de accesos "root" o número de operaciones realizadas como root en la conexión.
17.	Num_file_creations	Número de operaciones de creación de archivos en la conexión.
18.	Num_shells	Número de mensajes de shell.
19.	Num_access_files	Número de operaciones en archivos de control de acceso .
20.	Num_outbound_cmds	Número de comandos salientes en una sesión ftp.
21.	Is_hot_login	1 si el inicio de sesión pertenece a la lista "caliente", es decir, root o admin; de lo contrario 0.
22.	Is_guest_login	1 si el inicio de sesión es un inicio de sesión "invitado"; 0 de lo contrario .
23.	Count	Número de conexiones al mismo host de destino que la conexión actual en los últimos dos segundos.
24.	Srv_count	Número de conexiones al mismo servicio (número de puerto) que la conexión actual en los últimos dos segundos.
25.	Serror_rate	El porcentaje de conexiones que han activado el indicador (4) s0, s1, s2 o s3, entre las conexiones agregadas en recuento (23).

26. Srv_serror_rate	El porcentaje de conexiones que han activado el indicador (4) s0, s1, s2 o s3, entre las conexiones agregadas en srv_count (24) .
27. Rerror_rate	El porcentaje de conexiones que han activado el indicador (4) REJ, entre las conexiones agregadas en recuento (23).
28. Srv_rerror_rate	El porcentaje de conexiones que han activado el indicador (4) REJ, entre las conexiones agregadas en srv_count (24).
29. Same_srv_rate	El porcentaje de conexiones que fueron al mismo servicio, entre las conexiones agregadas en recuento (23) .
30. Diff_srv_rate	El porcentaje de conexiones que fueron a diferentes servicios, entre las conexiones agregadas en recuento (23).
31. Srv_diff_host_rate	El porcentaje de conexiones que fueron a diferentes máquinas de destino entre las conexiones agregadas en srv_count (24) .
32. Dst_host_count	Número de conexiones que tienen la misma dirección IP de host de destino.
33. Dst_host_srv_count	Número de conexiones con el mismo número de puerto.
34. Dst_host_same srv_rate	El porcentaje de conexiones que fueron al mismo servicio, entre las conexiones agregadas en dst_host_count (32) .
35. Dst_host_diff srv_rate	El porcentaje de conexiones que fueron a diferentes servicios, entre las conexiones agregadas en dst_host_count (32)
36. Dst_host_same src_port_rate	El porcentaje de conexiones que estaban en el mismo puerto de origen, entre las conexiones agregadas en dst_host_srv_c ount (33) .

37. Dst_host_srv diff_host_rate:	El porcentaje de conexiones que fueron a diferentes máquinas de destino, entre las conexiones agregadas en dst_host_srv_count (33).
38. Dst_host_serror_rate	El porcentaje de conexiones que han activado el indicador (4) s0, s1, s2 o s3, entre las conexiones agregadas en dst_host_count (32).
39. Dst_host_srv_serror_rate	El porcentaje de conexiones que han activado el indicador (4) s0, s1, s2 o s3, entre las conexiones agregadas en dst_host_srv_c ount (33).
40. Dst_host_rerror_rate	El porcentaje de conexiones que han activado el indicador (4) REJ, entre las conexiones agregadas en dst_host_count (32).
41. Dst_host_srv_rerror_rate	El porcentaje de conexiones que han activado el indicador (4) REJ, entre las conexiones agregadas en dst_host_srv_c ount (33).
42. Label:	Tipo de ataque/No ataque

REFERENCIAS.

[1] Hansman, S., & Hunt, R. (2005). A taxonomy of network and computer attacks. Computers & Security, 24(1), 31-43.

[2] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," Journal of Network and Computer Applications, vol. 60, pp. 19-31, 2016.

[3] K. Scarfone and P. Mell, “Guide to intrusion detection and prevention systems (idps),” NIST special publication, vol. 800, no. 2007, 2007.

[4] A. H. Farooqi and F. A. Khan, “Intrusion detection systems for wireless sensor networks: A survey,” in Proc. Future Generation Information Technology Conference, Jeju Island, Korea. Springer, 2009, pp. 234–241

[5] O. Y. Al-Jarrah, O. Alhussein, P. D. Yoo, S. Muhaidat, K. Taha, and K. Kim, “Data randomization and cluster-based partitioning for botnet intrusion detection,” IEEE Trans. Cybern., vol. 46, no. 8, pp. 1796–1806, 2015.

[6] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," Ieee communications surveys & tutorials, vol. 16, no. 1, pp. 303-336, 2014.

[7] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, pages 1–6. IEEE, 2009.

[8] Navlani, A., Fandango, A., & Idris, I. (2021). Python Data Analysis: Perform data collection, data processing, wrangling, visualization, and model building using Python. Packt Publishing Ltd.

[9] Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering, 160(1), 3-24.

[10] Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

[11] Mueller, J. P., & Massaron, L. (2021). Machine learning for dummies. John Wiley & Sons.

[12] Tin Kam Ho, "Random decision forests," Proceedings of 3rd International Conference on Document Analysis and Recognition, 1995, pp. 278-282 vol.1, doi: 10.1109/ICDAR.1995.598994.

[13] Cristianini, N., & Shawe-Taylor, J. (2000). An introduction to support vector machines and other kernel-based learning methods. Cambridge university press.

[14] Evgeniou, T., & Pontil, M. (1999, July). Support vector machines: Theory and applications. In Advanced Course on Artificial Intelligence (pp. 249-257). Springer, Berlin, Heidelberg.

[15] Azzaoui, H., Boukhamla, A. Z. E., Arroyo, D., & Bensayah, A. (2022). Developing new deep-learning model to enhance network intrusion classification. Evolving Systems, 13(1), 17-25

[16] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[17] https://www.unb.ca/cic/datasets/nsl.htm

[18] Asta, B. (2018). Analysis of Machine Learning in Intrusion Detection Systems. In The 55th NYSETA Fall 2018 Conference.

[19] Dhanabal, L., & Shantharajah, S. P. (2015). A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. International journal of advanced research in computer and communication engineering, 4(6), 446-452.

ANEXOS.

Ficheros adjuntos: los jupyter notebooks originales para el dataset de la empresa y el de NSL-KDD usados para la investigación y el diseño de los modelos así que sus copias en versión HTML.