Evaluación de modelos de Machine Learning para Sistemas de Detección de Intrusos en Redes IoT

Leonel Andrés Polanía Arias

Universidad de los Andes

Departamento de Ingeniería de Sistemas y Computación

Ingeniería de Sistemas y Computación

Bogotá D.C, Colombia

2021

Evaluación de modelos de Machine Learning para Sistemas de Detección de Intrusos en redes IoT

Leonel Andrés Polanía Arias

Proyecto de Grado presentado como requisito parcial para optar al título de:

Ingeniero de Sistemas y Computación

Director:

Ph.D. Carlos Andrés Lozano Garzón

Codirector:

Ph.D. German Adolfo Montoya

Grupo de Investigación:

COMIT

Universidad de los Andes

Departamento de Ingeniería de Sistemas y Computación

Ingeniería de Sistemas y Computación

Bogotá D.C, Colombia

2021

Resumen

Las redes *IoT* presentan un alto crecimiento y se espera una adopción masiva en los próximos años. Sin embargo, las diferencias de estas redes frente a las redes tradicionales así plantean retos para su seguridad. Los Sistemas de Detección de Intrusos cobran mayor importancia en este contexto. En este trabajo se entrenan dos modelos de *Machine Learning* para un Sistema de Detección de Intrusos con el fin de detectar ataques de tipo *DoS, Backdoor* y *Reconnaissance*: *Support Vector Machine* y *TabNet.* Los modelos son evaluados utilizando principalmente la métrica de *recall* y se realiza una comparación de estos. El modelo de mejor desempeño (TabNet) logra detectar aproximadamente 100% de los ataques, pero posee retos en el número de falsas alarmas al tener una tasa de estas de 50%, y también presenta limitaciones en la identificación del tipo exacto de ataque.

Palabras clave: IoT, Sistemas de Detección de Intrusos, Machine Learning, Deep Learning.

Abstract

Internet of Things networks show a high growth rate, and it is expected that massive adoption is achieved in the next few years. Nevertheless, differences between this kind of networks and traditional ones set challenges regarding privacy and security of users. In this context, Intrusion Detection Systems play a key role. This paper presents the training of two Machine Learning models for an IDS, aim to detect DoS, Backdoor and Reconnaissance attacks: Support Vector Machine and TabNet. Those two models are evaluated against their recall score and a comparison is done between them. The best model in this metric reaches a nearly 100% attack detection rate. However, challenges still remain in terms of high false alarms rate (50%). Also, the model struggles to point the exact attack type when an alarm is raised.

Keywords: Internet of Things, Intrusion Detection Systems, Machine Learning, Deep Learning.

Contenido

Lista de figuras

Pág.

Lista de tablas

Pág.

1. Introducción

El Internet de las cosas (*IoT, Internet of Things*) hace referencia a la interconexión de aparatos no tradicionales para que proveer servicios automatizados o aplicaciones a una amplia gama de usuarios [1]. Los casos de uso de IoT son variados y se pueden agrupar en categorías [2]:

- *Customer IoT*: Se pueden distinguir dos tipos de categorías de dispositivos. Por un lado, están aquellos cuyo uso está enfocado a la persona, entre ellos se encuentran los relojes inteligentes, gafas inteligentes, rastreadores de actividad física, monitores de indicadores de salud, etc. Por el otro, se encuentran los dispositivos pensados para el hogar, lo que incluye electrodomésticos inteligentes, y luces inteligentes.
- *Enterprise IoT*: Dispositivos que se enfocan en la generación de valor de las compañías de forma que proveen eficiencias operacionales o recolectan datos de los clientes, entre ellos se encuentran, por ejemplo, dispositivos autónomos y sensores que capturen variables ambientales.
- *Industrial IoT*: Dispositivos destinados a usarse en cadenas de producción, se resaltan los sensores y actuadores. Por ejemplo, las redes de sensores inalámbricas *WSN (Wireless Sensor Networks)*, que son usadas frecuentemente en el monitoreo de métricas medioambientales, y terminan siendo usadas en industrias como la agrícola y de recursos naturales.

Existe una elevada expectativa de crecimiento de uso de los dispositivos *IoT*. En 2019 habían 7.74 miles de millones de dispositivos *IoT* conectados y se estima que para 2025 sean 16.44 miles de millones, cifra que incrementa a 25.44 miles de millones para 2030 [3].

Sin embargo, la adopción masiva en hogares, empresas e industrias de los dispositivos *IoT* enfrenta un reto de seguridad digital. De acuerdo con Nokia [4], la participación de dispositivos infectados se incrementó en 100% de 2019 a 2020, es decir, de representar el 16.7% de los dispositivos infectados paso al 32.72%.

Dentro de las soluciones de defensa para redes *IoT* se encuentran los Sistemas de Detección de Intrusos (*IDS, Intrusión Detection Systems*) [1], los cuales se encargan de identificar los ataques cuando han ocurrido para implementar medidas de mitigación ante ellos, de forma que se minimiza el costo de ocurrencia de estos en términos de requerimientos de seguridad como disponibilidad, confidencialidad e integridad de la información.

Para estas soluciones existe la tendencia de la incorporación de modelos de *Machine Learning*, debido a que permite automatizar la detección de ataques la cual impacta de manera positiva la rapidez en la cual estos ataques pueden ser mitigados para el adecuado funcionamiento de la red. En [5] se presenta una revisión de la literatura del uso de *Machine Learning* para *IDS*, incluyendo los modelos más utilizados, y la tendencia del uso de modelos de *Deep Learning*.

En este sentido, en este trabajo se implementan dos modelos de *Machine Learning* utilizando el conjunto de datos UNSW-NB15 [6]: *Support Vector Machine* y uno de *Deep Learning*, TabNet. Posteriormente, se evalúan y comparan los modelos a partir de métricas de detección. Finalmente, se presentan conclusiones respecto a su posibilidad de uso en sistemas de detección de intrusos en escenario.

2. Objetivos

2.1 Objetivo general

Implementar dos modelos de Machine Learning para un sistema de detección de intrusos (*IDS*) en redes IoT.

2.2 Objetivos específicos

- Identificar los tipos de ataques más relevantes en redes *IoT*.
- Seleccionar dos modelos de clasificación multiclase de Machine Learning para un *IDS*.
- Entrenar los dos modelos seleccionados.
- Realizar un análisis comparativo del desempeño de los modelos implementados.

3. Marco teórico

3.1 Internet of Things

El Internet de las Cosas (*IoT, Internet of Things*) envuelve la conexión de dispositivos no tradicionales a la red. Para proveer servicios a personas, hogares, empresas e industrias.

En este sentido, las redes *IoT*, difieren significativamente de las redes de computadores tradicionales en varios aspectos, entre esos aspectos se encuentran [7]:

- Heterogeneidad: Hace referencia a la diversidad de los dispositivos y los protocolos.
- Escalabilidad: Las redes IoT en muchos casos de uso escalan a un número considerable de dispositivos. Adicionalmente, las redes deben soportar la conexión de más nodos, así como manejar su desconexión eficientemente.
- Comunicaciones: Utilización de distintas tecnologías inalámbricas como las redes *WiFi*, Bluetooth, Red Celular, etc.
- Consumo energético: Algunos dispositivos *IoT* tienen restricciones de consumo energético debido a razones de lejanía a una fuente de energía, movilidad o tamaño reducido.

Las redes *IoT* se puede ordenar en una arquitectura de 3 capas [8], [9]:

- Percepción: Encargada de medir variables medioambientales, recolectar información. Los componentes como sensores hacen parte de esta capa.
- Transporte: Encargada de transmitir la información recolectada en la capa anterior a un servicio exterior al dispositivo *IoT*. En esta capa se encuentran

funciones como el enrutamiento y la realización de enlaces en distintas formas como alámbrica, inalámbrica, satelital, etc.

- Aplicación: Encargada de procesar a mayor nivel la información recolectada de los dispositivos *IoT*, presentar la información al usuario final, así como tomar decisiones de actuación se serían enviadas de vuelta a los dispositivos de la red. Adicionalmente, está encargada de realizar algunas funciones administrativas como, la identificación de dispositivos.

3.2 Seguridad en redes IoT

En la seguridad de los datos se establecen tres requerimientos principales: integridad, disponibilidad y confidencialidad.

- Integridad: La información presentada es correcta y no sufre modificaciones no autorizadas.
- Disponibilidad: La información está disponible para los usuarios autorizados en el momento que lo requieran.
- Confidencialidad: Únicamente los usuarios autorizados tienen acceso a la información.

Teniendo en cuenta estos requerimientos, en *IoT* los ataques se clasifican nivel muy general en dos [9]: activos y pasivos. El primer tipo hace referencia al robo de datos o violación de la privacidad de los datos, en otras palabras, cuando el requerimiento de confidencialidad no se cumple. El segundo tipo, por otra parte, hace referencia a la destrucción de datos o su modificación no autorizada, es decir, cuando se vulnera el requerimiento de integridad.

No obstante, para otros autores la definición de estos dos tipos de ataques es diferente. En [1], los ataques pasivos son aquellos en los cuales, el atacantes busca pasar desapercibido u oculto. Mientras que, en los ataques activos, el atacante busca afectar el funcionamiento de la red *IoT*. En la Figura 1 se presenta una categorización de ataques en redes *IoT* más detallada.

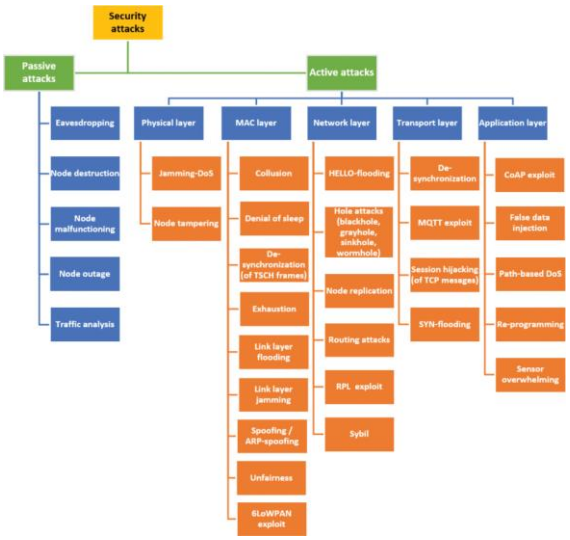

Figura 1 Categorización de ataques en redes *IoT* [1].

Los sistemas de detección, parte de las soluciones de defensa como se mencionó anteriormente, están enfocados en tres metodologías, así como la combinación de estas [9]:

- Mal-uso: Detecta ataques a partir de la correspondencia del tráfico monitoreado con una base de datos de ataques conocidos.
- Anomalía: Supone que el tráfico sigue un patrón de operación normal y cuando este se desvía de este patrón, detecta el ataque.
- Especificación: El ataque detectado por un modelo previo es confirmado por un experto a cargo.

Para los sistemas de detección de intrusos existe la tendencia a usar soluciones basadas en técnicas de *Machine Learning*. Por ejemplo, en [10] se propone un *framework* en el cual uno de los componentes en un agente de reacción basado en inteligencia artificial, el cual puede estar basado en múltiples algoritmos de aprendizaje supervisado, los autores también señalan la importancia de las métricas de evaluación de *IDS*, siendo el costo de los ataques relevante para la clasificación. En [11] se propone un *IDS* utilizando *clusters* de *Big Data* junto con *Deep Learning* a partir de *Convolutional Neural Networks*. Esta tendencia se da debido a que el uso de Machine Learning permite automatizar la detección de intrusos y así poder aplicar medidas de mitigación rápidamente minimizando el

costo de la ocurrencia de estos ataques bien sea en términos de disponibilidad del servicio o la integridad y confidencialidad de los datos de los usuarios.

3.3 Modelos de Machine Learning

3.3.1 Support Vector Machine

Support Vector Machine es un clasificador el cual busca realizar una separación de máximo margen con un hiperplano separador de los datos en un espacio dimensional más alto al que se encuentran originalmente. La Figura 2 ilustra esta idea. Para realizar esto no es necesario conocer la mapeamiento, en cambio, se requiere de una medida de similitud entre vectores, estos deben cumplir unas características para ser considerados *kernel*. En el caso más sencillo el producto punto es adecuado y se conoce como *kernel* lineal.

El problema de separación de máximo margen se puede plantear como un problema de optimización con restricciones para el cual existen algoritmos de solución como SMO.

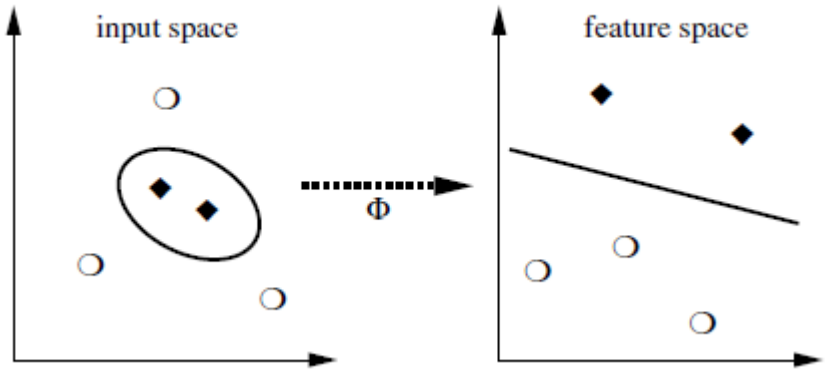

Figura 2 Idea de separación lineal en otro espacio dimensional. Tomado de [12].

3.3.2 TabNet

TabNet[13], es un modelo de *Deep Learning* realizado específicamente para trabajar con datos tabulares.

En primer lugar, cabe señalar que los modelos que hay retos relacionados con el entrenamiento de modelos de *Deep Learning* cuando los datos son tabulares. Entre ellos se encuentran [14]:

- Calidad de los datos: La presencia de valores faltantes o de valores atípicos, o inconsistentes. Adicionalmente, debido a que los datos son tomados en escenarios reales generalmente resultan en conjuntos altamente desbalanceados.
- No hay correlación espacial: La continuidad de características (columnas) no tiene significados o correlación, o esta es compleja.
- Preprocesamiento excesivo: Los datos requieren procesamiento para mejorar las métricas de desempeño o para poder usar los modelos. Por ejemplo, la necesidad de realizar codificaciones en las características categóricas que puede resultar en matrices dispersas.
- Sensibilidad del modelo: Los modelos de *Deep Learning* son generalmente sensibles a los cambios en los datos, a los parámetros e hiperparámetros.

No obstante, estos modelos poseen ventajas que los hacen atractivos para su uso en datos tabulares. Algunas de estas ventajas son [13]:

- Codificación eficiente: Permiten trabajar con datos como imágenes de manera eficiente.
- Alivia la ingeniería de características: Reduce el trabajo necesario en obtener características con poder de decisión a partir de los datos obtenidos.
- Permite el entrenamiento en *streaming*: Debido a estar basados en algoritmos como descenso del gradiente, permiten trabajar en *batch* (conjuntos pequeños de datos), por lo que se pueden seguir entrenando a medida que se obtienen más dados y por ello mismo se pueden entrenar conjuntos de una gran cantidad de muestras.

En este sentido TabNet, busca responder a algunos de los retos mencionados anteriormente. El modelo entrena los datos tabulares crudos utilizando descenso del gradiente. Para responder al reto de correlación espacial utiliza atención secuencial de forma que selecciona las características relevantes para la toma de decisión en cada etapa. Añade una máscara que da información acerca de la importancia de las características, buscando añadir interpretabilidad al modelo, característica de modelos tradicionales como los árboles de decisión.

Precisamente, la arquitectura de TabNet presentada en la Figura 3, busca emular un árbol de decisión, mientras que permite emular el ensamble de modelos al añadir más *steps* (etapas).

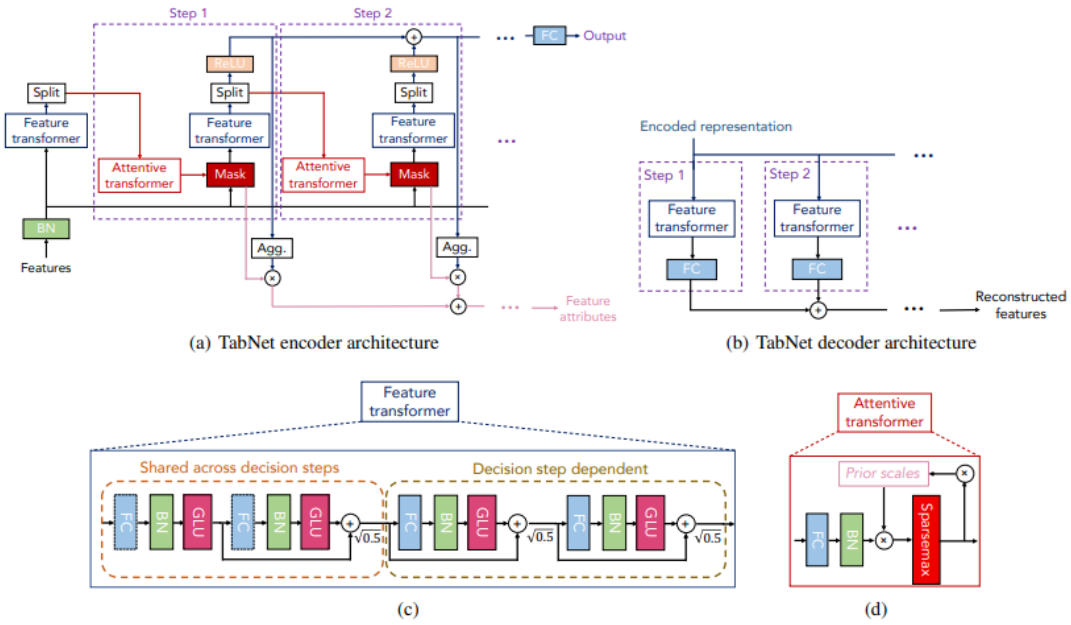

Figura 3 Arquitectura TabNet .Tomado de [13].

De esta forma TabNet presenta las siguientes ventajas:

- No requiere un escalamiento previo de los datos. Bloque *BN (Batch Normalizer*).
- No requiere la codificación de características categóricas.

- Selección e ingeniería de características. Bloques *Feature Transformer, Attentive Transformer.*
- Interpretación del modelo según importancia de características. Bloque *Mask*.
- Permite el entrenamiento en *batch* por usar descenso del gradiente.

3.4 Métricas de desempeño de IDS

Las métricas usuales para identificar el desempeño de modelos de Machine Learning son [5]:

Métricas por clase:

- *Precision*: Indica el porcentaje de muestras clasificadas correctamente sobre el número total de muestras clasificadas en esa clase. Una baja precisión indica que hay un alto número de falsos positivos.

$$Precision: \frac{\text{True positives}}{\text{True positives} + \text{False positives}}$$

- *Recall*: Indica el porcentaje de muestras clasificadas correctamente sobre el número de muestras que pertenecen a esta clase. Un bajo r*ecall* indica que no se están detectando muestras de esa clase.

$$Precision: \frac{True\ positives}{True\ positives + False\ negatives}$$

- *F1-Score*: Media armónica entre la *precision* y el *recall*, de forma que indica un desempeño general del modelo para esa clase.

$$Precision: 2\ X \frac{Precision\ X\ Recall}{Precision + Recall}$$

Métricas globales del modelo:

- *Accuracy*: Indica el porcentaje de aciertos del modelo, es decir, la cantidad de muestras clasificadas correctamente sobre el total de muestras.

$$Accuraccy = \frac{Correctly\ classified}{Total\ number}\ (\%)$$

- *Precision*: Promedio simple (*precision-macro*) o ponderado (*precision-weighted*) por número de muestras de la precisión de cada clase.
- *Recall*: Promedio simple (*recall-macro*) o ponderado (*recall-weighted*) por número de muestras de cada clase.

Es importante señalar que en muchas ocasiones la métrica más común para evaluar modelo es el *accuracy*, esta no es adecuada cuando se presenta un desbalance en el número de muestras de clases. Por ejemplo, debido a que la mayoría de tráfico no son ataques un modelo en un IDS que siempre indicara que no hay ataque tendría un alto *accuracy*, pero no detectaría ningún ataque. Algo similar ocurre con las métricas ponderadas, cuando las clases no son balanceadas.

4. Implementación de modelos

4.1 Perfilamiento de datos y preparación de datos

Se utilizó el conjunto de datos UNSW-NB15 [6], fue realizada a partir de la recolección de paquetes de red, provenientes tanto de ataques reales como de ataques sintéticos, por la Universidad de Nueva Gales del Sur, Australia. Estos ataques fueron clasificados en nueve familias. Para este trabajo se utiliza, en particular, los datos en formato .csv que contienen 47 características, junto con dos etiquetas de clasificación, una que indica si la muestra corresponde a un ataque o no, en caso de serlo una segunda etiqueta indica el tipo. El conjunto de datos tiene más de 2 millones de muestras, la mayoría correspondiendo a tráfico normal (no ataque/ benigno). En la Figura 4 se muestra la distribución de muestras del conjunto de datos de acuerdo con su tipo de ataque.

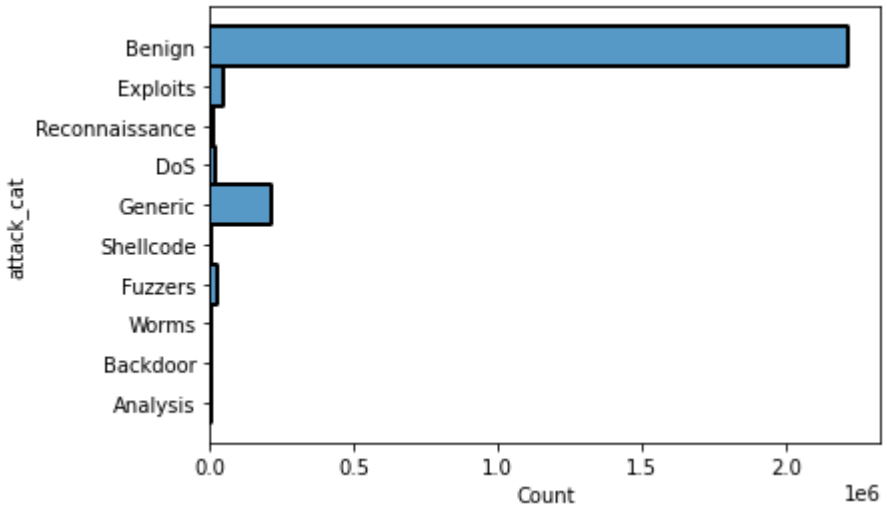

Figura 4 Número de ataques en UNSW-NB15 por tipo de ataque.

A partir de una comparación con otros conjuntos de datos relevantes, presentados en [15] para ataques de red en redes *IoT*, se seleccionaron los 3 ataques más comunes:

- *DoS: Denial of Service*, ataques que pretenden sobrecargar la red para evitar que usuarios legítimos puedan hacer uso de esta.
- *Backdoor*: Técnica que busca sobrepasar los mediante respuestas a aplicaciones clientes.
- *Reconnaissance*: Técnica para obtener información acerca de la red y sus hosts.

Las 47 características se encuentran se agrupan de acuerdo con el autor en las siguientes:

- Características de flujo: Identificación, IPs y puertos fuente y destino, junto con el protocolo.
- Características básicas: Se encuentran conteo de bytes, número de paquetes, duración de la muestra, estado de la conexión, servicio (protocolo de capa de aplicación) etc.
- Características de contenido: Números de secuencia, media del tamaño del paquete, tamaño del contenido, etc.
- Características de tiempo: Tiempo inicio y fin, *jitter*, entre otros.
- Características adicionales: Métricas calculadas, clasificadas en propósito general y de conexión.

Acorde con lo indicado en [15], las características de flujo identifican los atacantes y los host objetivo de los ataques, por lo cual, con excepción del protocolo no se consideran apropiados para el entrenamiento de un modelo de *Machine Learning*, por ello en este trabajo fueron eliminados. Estas características serían útiles para la aplicación de medidas de mitigación, como el bloqueo de IPs que han realizado ataques.

Entre otras características de preprocesamiento, de menor detalle, valores nulos de algunas características fueron reemplazados por 0 cuando correspondían a contadores. Las etiquetas de los ataques fueron corregidas ya que algunas tenían escritura distinta o espacios antes o después de la palabra. En la Figura 5 se observa el código en detalle del preprocesamiento inicial común a todos los algoritmos.

```
# Fill null attack to benign
df_clean['attack_cat'] = df_clean['attack_cat'].fillna('Benign')
# Change inconsistency in backdoor class
df_clean.loc[df_clean['attack_cat'] == 'Backdoors', 'attack_cat'] = 'Backdoor'
# Remove leading and trailing spaces
df['attack_cat'] = df['attack_cat'].str.strip()
# Fill null in service to unknown
df_clean['service'] = df_clean['service'].fillna('unknown')
# Fill null is_ftp_login to 0
df_clean['is_ftp_login'] = df_clean['is_ftp_login'].fillna(0)
# Change 'is_ftp_login' to int
df_clean = df_clean.astype({'is_ftp_login': int})
# Remove rows where 'is_ftp_login' is not valid (i.e between 0 and 1)
df_clean = df_clean.drop(df_clean[~df_clean['is_ftp_login'].isin([0, 1])].index)
# Fill null 'ct_flw_http_mthd' to 0
df_clean['ct_flw_http_mthd'] = df_clean['ct_flw_http_mthd'].fillna(0)
# Change 'ct_flw_http_mthd' to int
df_clean = df_clean.astype({'ct_flw_http_mthd': int})
# Fill null 'ct_ftp_cmd' to 0
df_clean['ct_ftp_cmd'] = df_clean['ct_ftp_cmd'].fillna(0)
# Change 'ct_flw_http_mthd' to int
df_clean = df_clean.astype({'ct_ftp_cmd': int})
# Drop columns with user dependent information
df_clean = df_clean.drop(["srcip"], axis=1)
df_clean = df_clean.drop(["sport"], axis=1)
df_clean = df_clean.drop(["dstip"], axis=1)
df_clean = df_clean.drop(["dsport"], axis=1)
# Remove not interested in attacks
df_clean = df_clean.drop(df_clean[~df_clean['attack_cat'].isin(['Benign', 'DoS', 'Reconnaissance','Backdoor'])].index)
```

Figura 5. Código de limpieza de datos.

El total de datos fue separado en tres conjuntos (código en la Figura 6): Entrenamiento, utilizados para entrenar los modelos; Validación, para selección de modelo e hiperparámetros y Evaluación, obtención de métricas para el modelo seleccionado.

```
X = df_clean.drop(['attack_cat', 'Label'], axis=1)
X = pd.get_dummies(X)
X = X.drop(['service_unknown'], axis=1)
y = df_clean['attack_cat']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.10, random_state=42)
```

Figura 6. Distribución de conjuntos de datos.

4.2 Implementación *Support Vector Machine*

Para la implementación de *Support Vector Machine* se utilizó la librería scikit-learn [16] creando un *pipeline* (secuencia de pasos). En particular, se utilizó un *kernel* lineal debido a que a diferencia de los otros kernels disponibles, este es escalable en el número de datos. Para el manejo del problema multiclase se utilizó la estrategia *OneVsRest*, en la cual se crea un modelo binario por cada clase, siendo la estrategia por defecto de scikit-learn, el modelo es el último paso del *pipeline* (Figura 7).

Para las características categóricas se utilizó la estrategia de *OneHotEncoding*, en la cual se crea una nueva característica binaria por cada categoría, para ello se utiliza *get_dummies* de la librería pandas [17] como se observa en la Figura 6. Mientras que las demás características fueron escaladas siguiendo de acuerdo con su mínimo y máximo valor de forma que sus valores estuvieran todos entre 0 y 1. Este paso es el primero del pipeline como se presenta en la Figura 7.

Debido al desbalance de datos fue necesario aplicar estrategias para evitar que el modelo quedara sesgado a la clase mayoritaria (Benigno). Para este modelo se utilizó estrategias de muestreo en dos pasos. Primero un sub-muestreo aleatorio de la clase mayoritaria y luego un sobre-muestreo generando ejemplos sintéticos utilizando SMOTE de las demás clases (los ataques). Esto se evidencia en los en los pasos *rus* y *smote* del pipeline entrenado, en la Figura 7.

Para *Support Vector Machine Lineal* el parámetro más relevante es la constante de regularización C, en este sentido se utilizó validación cruzada k-fold con k=2 para la selección de este parámetro. La métrica que se consideró más relevante para el problema fue el *recall*, y para evitar el sesgo por el desbalanceo de clases se utilizó el promedio no ponderado.

```
# Scale between 0 and 1
sc = MinMaxScaler()
# Linear Support Vector Machine
svm = LinearSVC()
# Under sample mayority class
rus = RandomUnderSampler(sampling_strategy = 'majority',random_state = 0)
# SMOTE not mayority class
smote = SMOTE(random_state = 0)
pipeline = Pipeline([("sc", sc), ("rus", rus), ("smote", smote), ("svm", svm)], verbose = True)
```

Figura 7 Código Python pipeline Support Vector Machine.

En primer lugar, se realizó una búsqueda en una escala logarítmica entre 10-3 y 103.La Figura 8 muestra cómo se comporta esta métrica variando la constante de regularización en esta primera búsqueda.

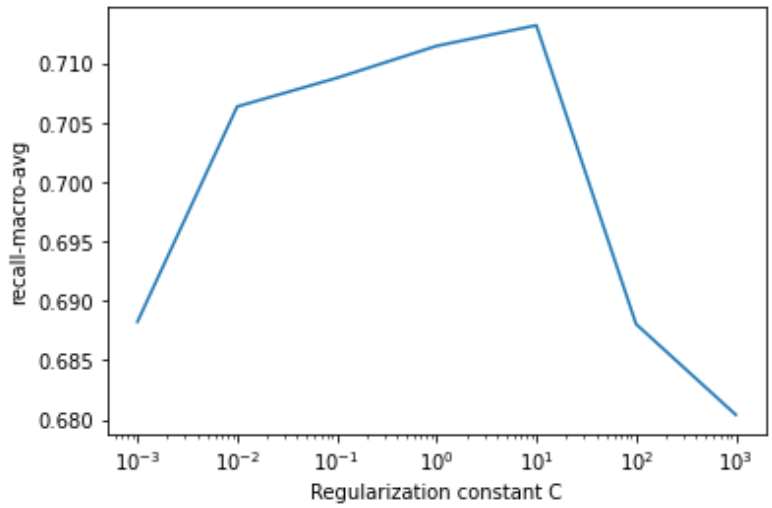

Figura 8 Recall-macro de Support Vector Machine según constante de regularización.

Como se puede observar la mejor métrica se obtuvo en 101, por ello se decidió realizar una segunda búsqueda con valores entre 0-100, el comportamiento de la métrica se presenta en la Figura 9.

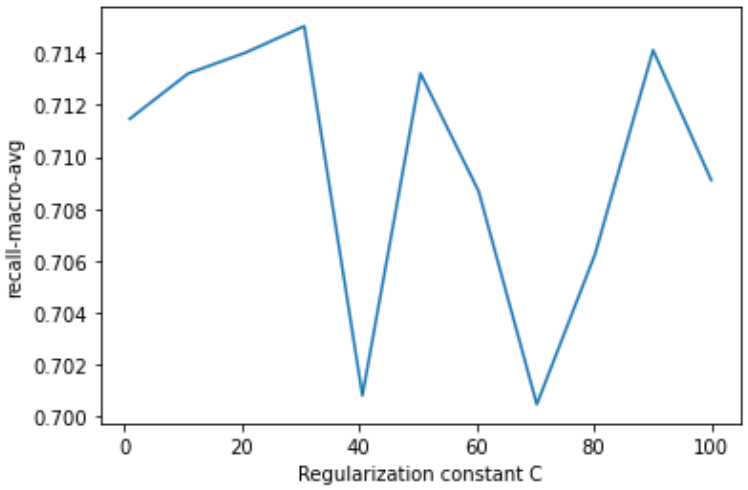

Figura 9 Recall-macro de Support Vector Machine según constante de regularización entre 0-100.

Finalmente, el mejor modelo es aquel con C=30, cuyo desempeño detallado se presenta en la Tabla 1.

Tabla 1 Reporte de clasificación *Support Vector Machine*.

	Precision	***Recall***	***F1-Score***	***Support***
Backdoor	0.12	0.31	0.17	129
Benign	1.00	0.98	0.99	125412
DoS	0.44	0.69	0.54	1020
Reconnaissance	0.34	0.85	0.48	836
Accuracy			0.98	127397
Macro Avg	0.47	0.71	0.55	127397
Weighted Avg	0.99	0.98	0.98	127397

La matriz de confusión del modelo en datos de validación se presenta en la Figura 10. Como se puede observar el modelo tiene problemas para la detección

del tipo de ataque exacto principalmente para *Backdoor* y *DoS*. Sin embargo, se observa que solo pocos ataques de *DoS* pasaron como muestra benigna.

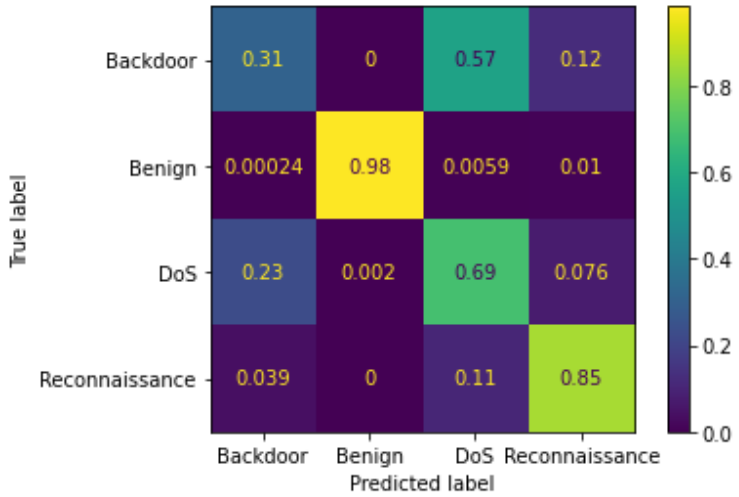

Figura 10 Matriz de confusión *Support Vector Machine*.

Por otra parte, es importante mencionar que, siguiendo la misma estrategia para seleccionar C, se evaluaron modelos utilizando otras técnicas de sobre-muestreo: Sobre muestreo aleatorio de las clases minoritarias, SMOTE *borderline* y ADASYN. Sin embargo, la que obtuvo mejores métricas fue utilizando SMOTE regular, que fue presentada anteriormente.

4.3 Implementación *TabNet*

Para la implementación de TabNet se utilizó una librería desarrollada en PyTorch [18]. En este caso debido a las ventajas mencionadas anteriormente. Los datos fueron preprocesados más allá de lo que se indicó en Perfilamiento de Datos. Es de señalar, que las características categóricas son adecuadamente manejadas al indicarle el índice de estas al modelo, en este sentido, en lugar de realizar *OneHotEncoding*, se utiliza *LabelEncoding*, que mapea los valores a números sin añadir nuevas dimensiones. Ello se evidencia en la Figura 11, donde se utiliza un *LabelEndoder* en las características *proto*, *state* y *service*.

```
X = df_clean.drop(['attack_cat', 'Label'], axis=1)
le_proto = LabelEncoder()
le_state = LabelEncoder()
le_service = LabelEncoder()
X['proto'] = le_proto.fit_transform(X['proto'])
X['state'] = le_state.fit_transform(X['state'])
X['service'] = le_service.fit_transform(X['service'])

y = df_clean['attack_cat']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.10, random_state=42)
```

Figura 11. Codificación de características categóricas y distribución de datos.

Los parámetros más relevantes de TabNet son los siguientes:

- N_{STEPS}: Número de pasos o etapas en la arquitectura. Mayores valores, mayor complejidad del modelo.
- N_D: Tamaño de la capa de decisión. Mayores valores, mayor complejidad del modelo.
- N_A: Tamaño del *attention embedding* de la máscara.

Siguiendo los ejemplos y recomendaciones dados en el artículo de TabNet[13] se entrenaron tres modelos con distintos niveles de complejidad, los parámetros de cada uno se muestran en la Tabla 2.

Tabla 2 Parámetros modelos de TabNet

	N_D	**N_A**	**N_{STEPS}**
TabNet-S	32	32	5
TabNet-M	64	64	7
TabNet-L	128	128	5

La Figura 12 muestra un código de ejemplo de TabNet para la utilización del modelo TabNet, se observan los parámetros mencionados anteriormente. Es importante, también resaltar la necesidad de indicar cuáles son las características categóricas y la cantidad de categorías de cada una en los parámetros *cat_idxs* y *cat_dims* respectivamente.

```
model = TabNetClassifier(
                        n_d = 64,
                        n_a = 64,
                        n_steps = 7,
                        optimizer_fn=torch.optim.Adam,
                        optimizer_params=dict(lr=2e-2),
                        scheduler_params={"step_size":10,
                                          "gamma":0.9},
                        scheduler_fn=torch.optim.lr_scheduler.StepLR,
                        cat_idxs = [0, 1, 9],
                        cat_dims = [df_clean['proto'].value_counts().shape[0], df_clean['state'].value_counts().shape[0], df_clean['service'].value_counts().shape[0] ]
                       )


model.fit(
    X_res,y_res,
    eval_set=[(X_t, y_train), (X_val, y_val)],
    eval_name=['train', 'valid'],
    eval_metric=['balanced_accuracy'],

    batch_size=2048*4, virtual_batch_size=256*4,
    num_workers=0,
    weights=1,
    drop_last=False
)
```

Figura 12. Código modelo TabNet.

Los resultados en métricas macro se presentan en la Tabla 3

Tabla 3 Reporte de clasificación modelos TabNet.

Modelo	***Accuracy***	***Macro-Precision***	***Macro-Recall***	***Macro-F1-Score***
TabNet-S	98%	50%	74%	56%
TabNet-M	98%	59%	76%	59%
TabNet-L	98%	53%	77%	60%

Como se puede observar el modelo de mejor desempeño fue TabNet-L, de acuerdo con la métrica seleccionada *Macro-Recall*. De este modelo se presenta su desempeño en mayor detalle. La gráfica de la Figura 13 muestra la evolución de la métrica de interés según el número de épocas de entrenamiento.

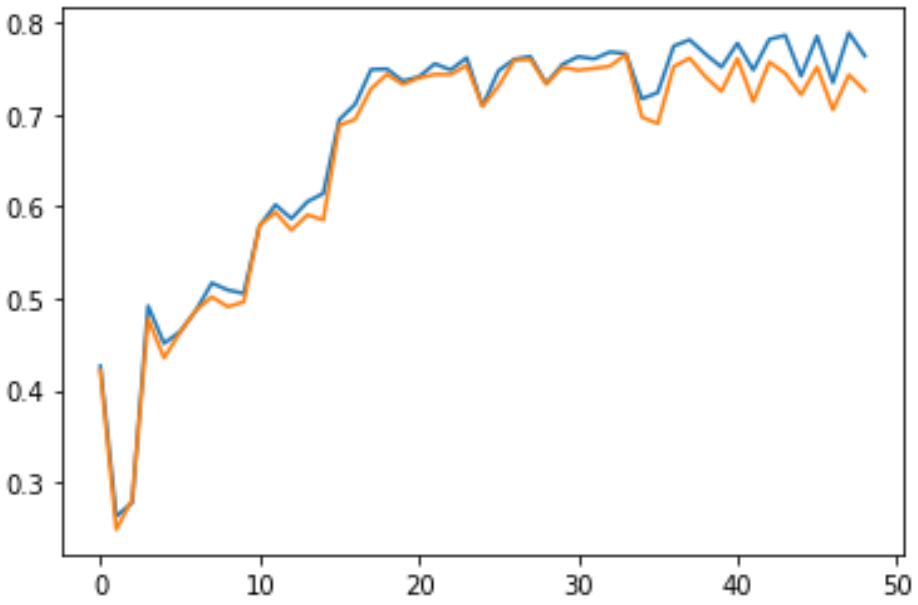

Figura 13 *Macro-Recall* de TabNet-L por número de épocas. Azul – Entrenamiento, Naranja – Validación.

En la Tabla 4 se muestra el reporte de clasificación mostrando métricas por clase y globales. Como se observa, la clase de peor desempeño fue Backdoor, explicado en parte por la poca cantidad de datos independientes en entrenamiento, mientras que, la clase benigno muestra el mejor desempeño.

Tabla 4 Reporte de clasificación TabNet-L.

	Precision	***Recall***	***F1-Score***	***Support***
Backdoor	0.05	0.43	0.09	129
Benign	1.00	0.98	0.99	125387
DoS	0.46	0.84	0.59	1014
Reconnaissance	0.62	0.82	0.71	853
Accuracy			0.98	127397
Macro Avg	0.53	0.77	0.60	127397
Weighted Avg	0.99	0.98	0.99	127397

La Figura 14 muestra la matriz de confusión normalizada por filas (en la diagonal el *recall*). Es de notar que un muy bajo porcentaje de ataques paso desapercibido

(clasificado como benigno). No obstante, el modelo no logra distinguir adecuadamente entre Backdoor y DoS.

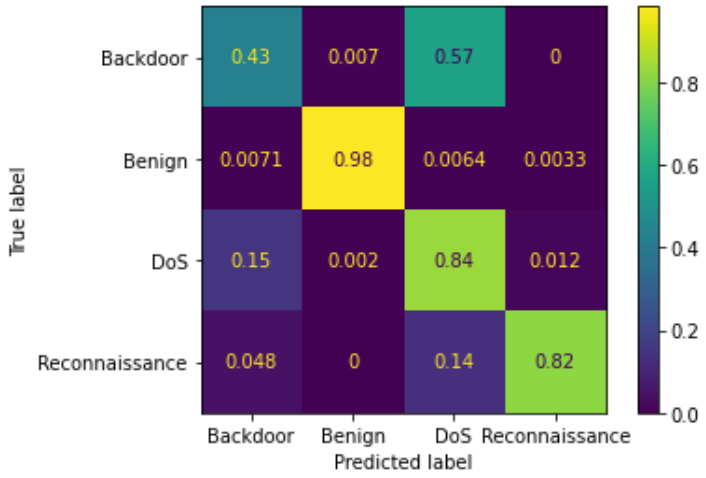

Figura 14 Matriz de confusión TabNet-L.

Por las características de TabNet, es posible obtener una métrica que indica la importancia de las características para la toma de decisión del modelo. La Figura 15 muestra la importancia de las características. Como se puede observar la característica de mayor importancia es *sttl* que mide el *Time to Live* de los paquetes que van de la fuente al destino.

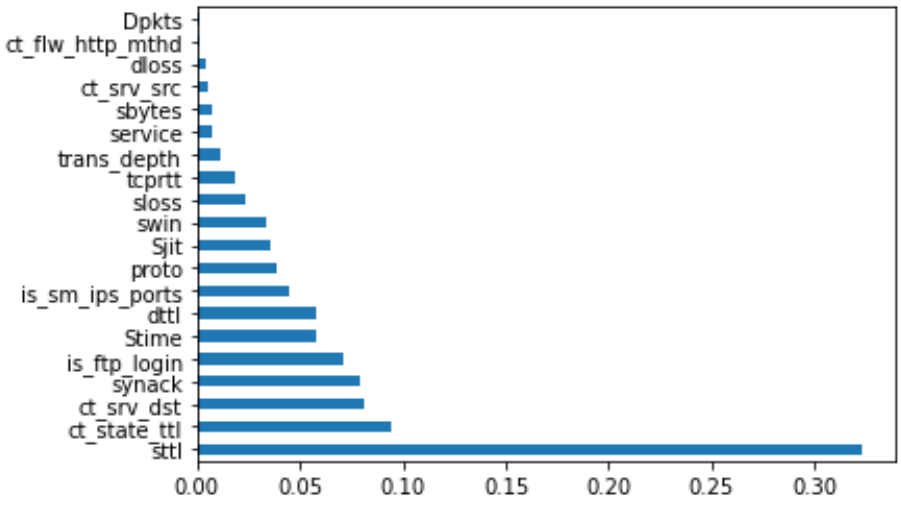

Figura 15 Importancia de características TabNet-L.

5. Discusión

La Tabla 5 muestra el desempeño de los modelos mostrados en este trabajo en los datos de validación. Es relevante mencionar que todos los modelos de TabNet superan en todas las métricas al modelo de *Support Vector Machine*. Como se puede observar, debido al desbalance de datos, todos los modelos tienen un *accuracy* aproximado de 98%, por lo que se reafirma que no es una métrica adecuada para evaluar el modelo. También se ha observado en los reportes de clasificación mostrados anteriormente que las métricas ponderadas tienen a dar muy altas, esto por el buen desempeño de los modelos en clasificar cuando no se ha realizado un ataque. Se resalta la importancia del *recall* ya que un número bajo en la clase ataque, indicaría que un ataque pasó desapercibido por el IDS.

Tabla 5 Comparación de modelos

Modelo	***Accuracy***	***Macro-Precision***	***Macro-Recall***	***Macro-F1-Score***
Support Vector Machine	98%	47%	71%	55%
TabNet-S	98%	50%	74%	56%
TabNet-M	98%	59%	76%	59%
TabNet-L	98%	53%	77%	60%

Como se puede observar, el mejor modelo en datos de validación fue TabNet-L. En este sentido se calcularon métricas de desempeño definitivas en el conjunto de datos de evaluación. Para este conjunto de datos el modelo tardó en clasificar las muestras 11 segundos utilizando una GPU Tesla K80.

Tabla 6 Reporte de clasificación de evaluación de TabNet-L.

	Precision	***Recall***	***F1-Score***	***Support***
Backdoor	0.05	0.37	0.09	368
Benign	1.00	0.98	0.99	313353
DoS	0.47	0.84	0.61	2634
Reconnaissance	0.64	0.81	0.71	2136
Accuracy			0.98	318491
Macro Avg	0.54	0.75	0.60	318491
Weighted Avg	0.99	0.98	0.99	318491

La Figura 11 muestra la matriz de confusión en datos de evaluación para el modelo seleccionado, TabNet-L. Se observan las mismas características que en datos de validación. El modelo tiene limitaciones para detectar el tipo de ataque: en el mejor de los casos si indica un ataque de tipo *Reconaissance*, solo se está seguro en el 67% de que este es en efecto el tipo de ataque. En el peor caso solo se tiene una certeza del 5%(*Backdoor*). No obstante, el modelo da pistas ya que no discrimina correctamente entre *Backdoor* y *DoS*, si la alarma es de este alguno de estos dos tipos, el equipo encargado del *IDS* debería plantear que fue cualquiera de esos dos y no debería investigar si se trata de un *Reconaissance*.

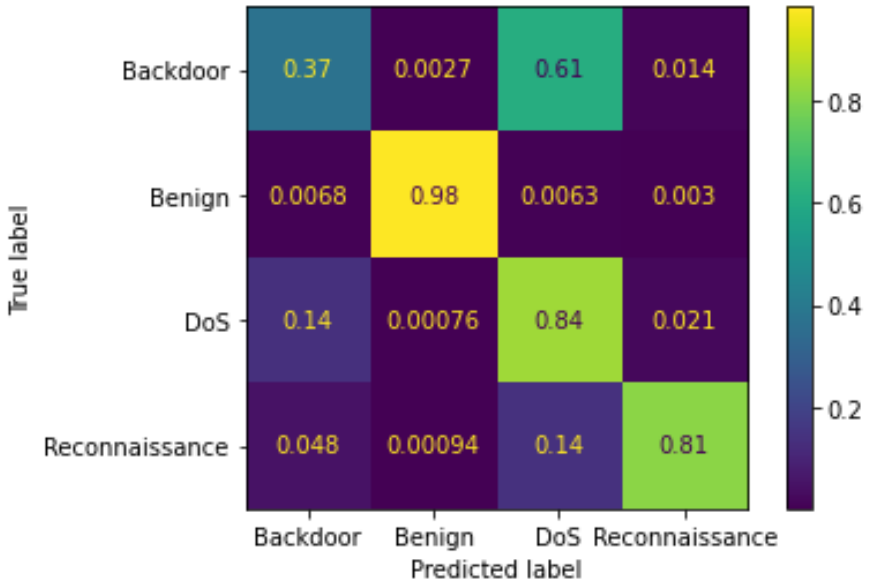

Figura 16 Matriz de confusión en datos de evaluación TabNet-L.

Debido a la importancia de clasificar adecuadamente si los ataques son detectados independientemente de a qué tipo de ataque fue asignado, se decidió analizar el mejor modelo como binario. La Tabla 7 muestra las métricas de desempeño de este análisis. Se evidencia que la métrica de mayor relevancia, el *recall* de ataques es de 1.00, ello indica que el modelo detecta casi el 100% de los ataques. Sin embargo, la precisión es de 51%, por lo que 1 de cada dos alarmas de provocaría el *IDS* sería falsa, lo cual podría llevar a que se apliquen medidas de bloque a usuarios legítimos o que el equipo encargado de actuar frente a las alarmas se vea alerta e investigue casos que no son ataques, resultando en un desperdicio de recursos humanos.

Tabla 7 Reporte de clasificación TabNet-L como modelo binario.

	Precision	***Recall***	***F1-Score***	***Support***
Attack	0.51	1.00	0.67	5207
Benign	1.00	0.98	0.99	313284
Accuracy			0.98	318491
Macro Avg	0.75	0.99	0.83	318491
Weighted Avg	0.99	0.98	0.99	318491

6. Conclusiones y recomendaciones

6.1 Conclusiones

El modelo seleccionado, TabNet-L, logra detectar aproximadamente el 100% de los ataques. Adicionalmente, da información de sobre la importancia de las características, lo que resulta importante para la investigación por el personal de seguridad ante la ocurrencia de un ataque. Sin embargo, el modelo no se considera apto para un ambiente de producción debido a la alta tasa de falsas alarmas que causarían bloqueos a un 2% del tráfico de usuarios legítimos o visto de otra forma una de cada dos alarmas alertaría al equipo de seguridad de manera innecesaria.

El modelo tiene problemas para determinar el tipo de ataques, en particular, le cuesta diferenciar entre *Backdoor* y *Denial of Service*.

6.2 Trabajo futuro

Como trabajo futuro se propone explorar las razones por las que el modelo le cuesta diferencian entre tipos de ataques, esto se podría realizar mediante técnicas de *clustering*, modelos de asociación o revisando en detalle la máscara de características de TabNet. También se propone buscar mejorar el desempeño de TabNet con una búsqueda más extensiva de hiperparámetros, o utilizando el pre-entrenador, el cual es un modelo no supervisado disponible en la librería de TabNet. También se propone comparar el desempeño de este modelo con otros clasificadores, en particular, clasificadores combinados, los cuales son objeto de comparación en el artículo de TabNet.

Bibliografía

[1] I. Butun, P. Osterberg, and H. Song, “Security of the Internet of Things: Vulnerabilities, Attacks, and Countermeasures,” *IEEE Commun. Surv. Tutorials*, vol. 22, no. 1, pp. 616–644, Jan. 2020, doi: 10.1109/COMST.2019.2953364.

[2] 5G Americas, “5G: The Future of IoT,” *5G Am. White Pap.*, 2019.

[3] Statista, “IoT connected devices worldwide 2019-2030,” 2021. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[4] Nokia, “Nokia: Threat Intelligence Report 2020.” 2020, [Online]. Available: https://onestore.nokia.com/asset/210088.

[5] R. Ahmad and I. Alsmadi, “Machine learning approaches to IoT security: A systematic literature review,” *Internet of Things*, vol. 14, p. 100365, Jun. 2021, doi: 10.1016/J.IOT.2021.100365.

[6] N. Moustafa and J. Slay, “UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set),” *2015 Mil. Commun. Inf. Syst. Conf. MilCIS 2015 - Proc.*, Dec. 2015, doi: 10.1109/MILCIS.2015.7348942.

[7] A. Balte, A. Kashid, and B. Patil, “Security Issues in Internet of Things (IoT): A Survey,” *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 5, no. 4, p. 2277, 2015, [Online]. Available: www.ijarcsse.com.

[8] E. Borgia, “The Internet of Things vision: Key features, applications and open issues,” 2014, doi: 10.1016/j.comcom.2014.09.008.

[9] E. Benkhelifa, T. Welsh, and W. Hamouda, “A critical review of practices and challenges in intrusion detection systems for IoT: Toward universal and resilient systems,” *IEEE Commun. Surv. Tutorials*, vol. 20, no. 4, pp. 3496–3509, Oct. 2018, doi: 10.1109/COMST.2018.2844742.

[10] M. Bagaa, T. Taleb, J. B. Bernabe, and A. Skarmeta, “A Machine Learning Security Framework for Iot Systems,” *IEEE Access*, vol. 8, pp. 114066–114077, 2020, doi: 10.1109/ACCESS.2020.2996214.

[11] W. Zhong, N. Yu, and C. Ai, “Applying big data based deep learning system to intrusion detection,” *Big Data Min. Anal.*, vol. 3, no. 3, pp. 181–195, Sep. 2020, doi: 10.26599/BDMA.2020.9020003.

[12] B. Schölkopf and A. J. Smola, “Learning with kernels: support vector machines, regularization, optimization, and beyond.” 2002.

[13] S. O. Arik and T. Pfister, “TabNet: Attentive Interpretable Tabular Learning,” Aug. 2019, [Online]. Available: https://arxiv.org/abs/1908.07442.

[14] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, “Deep Neural Networks and Tabular Data: A Survey,” Oct. 2021, [Online]. Available: http://arxiv.org/abs/2110.01889.

[15] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, “NetFlow Datasets for Machine Learning-based Network Intrusion Detection Systems,” *Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng. LNICST*, vol. 371 LNICST, pp. 117–135, Nov. 2020, doi: 10.1007/978-3-030-72802-1_9.

[16] “scikit-learn: machine learning in Python — scikit-learn 1.0.1 documentation.” https://scikit-learn.org/stable/.

[17] “pandas - Python Data Analysis Library.” https://pandas.pydata.org/ (accessed Jan. 18, 2022).

[18] “pytorch-tabnet · PyPI.” https://pypi.org/project/pytorch-tabnet/.