Universidad Tecnológica de Bolívar

Inteligencia Artificial

Detección de intrusiones RT-IoT2022

Mauro Alonso González Figueroa, T00067622

Juan Jose Jiménez Guardo, T00068278

Revisado Por

Edwin Alexander Puertas Del Castillo

23 de noviembre de 2025

Índice

1. Comprensión del Negocio

Esta sección establece el marco contextual del proyecto, definiendo la problemática actual en el ámbito del Internet de las Cosas (IoT), los objetivos planteados y los criterios mediante los cuales se evaluará el éxito de la solución propuesta.

1.1. Contexto general

El Internet de las Cosas (IoT) ha experimentado un crecimiento exponencial en la última década, integrándose en sectores críticos como la industria, la salud y las ciudades inteligentes. Sin embargo, esta proliferación de dispositivos conectados ha ampliado significativamente la superficie de ataque disponible para los ciberdelincuentes.

A diferencia de los sistemas informáticos tradicionales, los dispositivos IoT suelen caracterizarse por tener recursos limitados de procesamiento y energía, lo que dificulta la implementación de mecanismos de seguridad robustos convencionales. Las vulnerabilidades comunes incluyen el uso de credenciales predeterminadas, falta de cifrado en las comunicaciones y firmware desactualizado.

En este escenario, los sistemas de seguridad perimetral tradicionales (como firewalls básicos) resultan insuficientes. Surge entonces la necesidad crítica de implementar Sistemas de Detección de Intrusos (IDS) basados en el análisis de tráfico de red, capaces de monitorear el comportamiento de los dispositivos y alertar sobre actividades sospechosas en tiempo real.

1.2. Problema a resolver

El problema central que aborda este proyecto es la dificultad para distinguir, de manera automatizada y precisa, entre el comportamiento legítimo de un dispositivo IoT y las anomalías generadas por ciberataques.

Específicamente, el desafío consiste en analizar flujos de tráfico de red complejos para realizar una clasificación multicategoría. No basta con una detección binaria (normal vs. ataque); es necesario identificar la naturaleza específica de la intrusión (por ejemplo, ataques

de denegación de servicio, escaneos de puertos o inyecciones de fuerza bruta) utilizando los datos proporcionados por el dataset RT-IoT2022.

1.3. Objetivo del proyecto

El objetivo principal de esta investigación es desarrollar, entrenar y validar un algoritmo de clasificación utilizando métodos de aprendizaje automático (Machine Learning). Este modelo debe ser capaz de procesar las características del tráfico de red contenidas en el dataset RT-IoT2022 para distinguir eficazmente entre tráfico normal y tráfico malicioso.

Se busca obtener un modelo que no solo sea preciso, sino también eficiente, considerando las restricciones de latencia y recursos típicas de los entornos IoT.

1.4. Métricas de éxito

Para evaluar el desempeño y la viabilidad de los modelos propuestos, se utilizarán las siguientes métricas estándar en la industria de la ciberseguridad y la ciencia de datos:

- **Accuracy (Exactitud):** Para medir el porcentaje global de predicciones correctas.
- **Precision (Precisión):** Para evaluar la confiabilidad de las alertas de ataque generadas (minimización de falsos positivos).
- **Recall (Sensibilidad):** Crucial en seguridad, para medir la capacidad del modelo de detectar la mayor cantidad posible de ataques reales (minimización de falsos negativos).
- **F1-Score:** Como media armónica entre Precision y Recall, proporcionando una métrica balanceada, especialmente útil dado el posible desbalance de clases en los datos de ataques
- **Matriz de Confusión:** Para visualizar detalladamente el desempeño del modelo en cada clase específica de ataque.

2. Comprensión de los Datos

Esta fase del proyecto se centra en la adquisición y el análisis preliminar del conjunto de datos RT-IoT2022. El objetivo es familiarizarse con la estructura de la información, identificar la naturaleza de los atributos disponibles y detectar patrones iniciales mediante un análisis exploratorio focalizado.

2.1. Descripción del dataset RT-IoT2022

El conjunto de datos utilizado, denominado RT-IoT2022, proviene de un entorno de pruebas controlado (testbed) diseñado para simular escenarios de ciberseguridad en tiempo real sobre infraestructuras de Internet de las Cosas.

La captura del tráfico de red se realizó utilizando herramientas de análisis de paquetes como Wireshark, registrando la interacción entre dispositivos IoT (víctimas) y la infraestructura atacante. Posteriormente, para transformar los paquetes crudos en datos tabulares aptos para el aprendizaje automático, se emplearon extractores de características de flujo de red como Zeek y CICFlowmeter. Esto permite representar las comunicaciones no como paquetes aislados, sino como flujos bidireccionales con atributos estadísticos definidos.

2.2. Variables incluidas

El dataset se compone de una variedad de características que describen el comportamiento de los flujos de red. Estas variables pueden categorizarse en los siguientes grupos:

- **Identificadores de flujo:** Direcciones IP de origen y destino, y puertos de servicio (aunque estos suelen ser eliminados para evitar sesgos hacia direcciones específicas).
- **Protocolos:** Información sobre el protocolo de capa de transporte (TCP, UDP) y servicios de aplicación.
- **Estadísticas de volumen:** Contadores de paquetes enviados y recibidos, así como la longitud de los bytes transferidos (Payload).

- **Tiempos:** Duración del flujo y tiempos de llegada entre paquetes (IAT − Inter-Arrival Times), cruciales para detectar anomalías temporales.
- **Flags TCP:** Indicadores de estado de la conexión (SYN, FIN, RST, ACK), fundamentales para identificar escaneos y ataques de denegación de servicio.
- **Variable objetivo (Target):** La columna `Attack_type`, que etiqueta cada registro como tráfico normal o especifica la categoría del ataque (ej. DoS, Brute Force, MQTT, etc.).

2.3. Volumen y estructura

El conjunto de datos se proporciona dividido en dos archivos principales: un conjunto de entrenamiento (`train_data.csv`) y un conjunto de prueba (`test_data.csv`). Esta separación previa facilita la validación de los modelos y asegura que no haya fuga de información durante la fase de entrenamiento.

La estructura es tabular, donde cada fila representa un flujo de red único y las columnas corresponden a las características extraídas mencionadas anteriormente. Se verificó la consistencia estructural entre ambos archivos, asegurando que el número y tipo de columnas fuesen idénticos antes de proceder con el análisis.

2.4. Análisis exploratorio (EDA)

Dado que los datos provienen de extractores de flujo automatizados, el análisis exploratorio se centró principalmente en examinar las relaciones lineales entre las variables numéricas para identificar redundancia.

Se generó una matriz de correlación utilizando el coeficiente de Pearson para cuantificar la relación lineal entre los pares de atributos numéricos (tipos `Int32`, `Int64`, `Float32`, `Float64`).

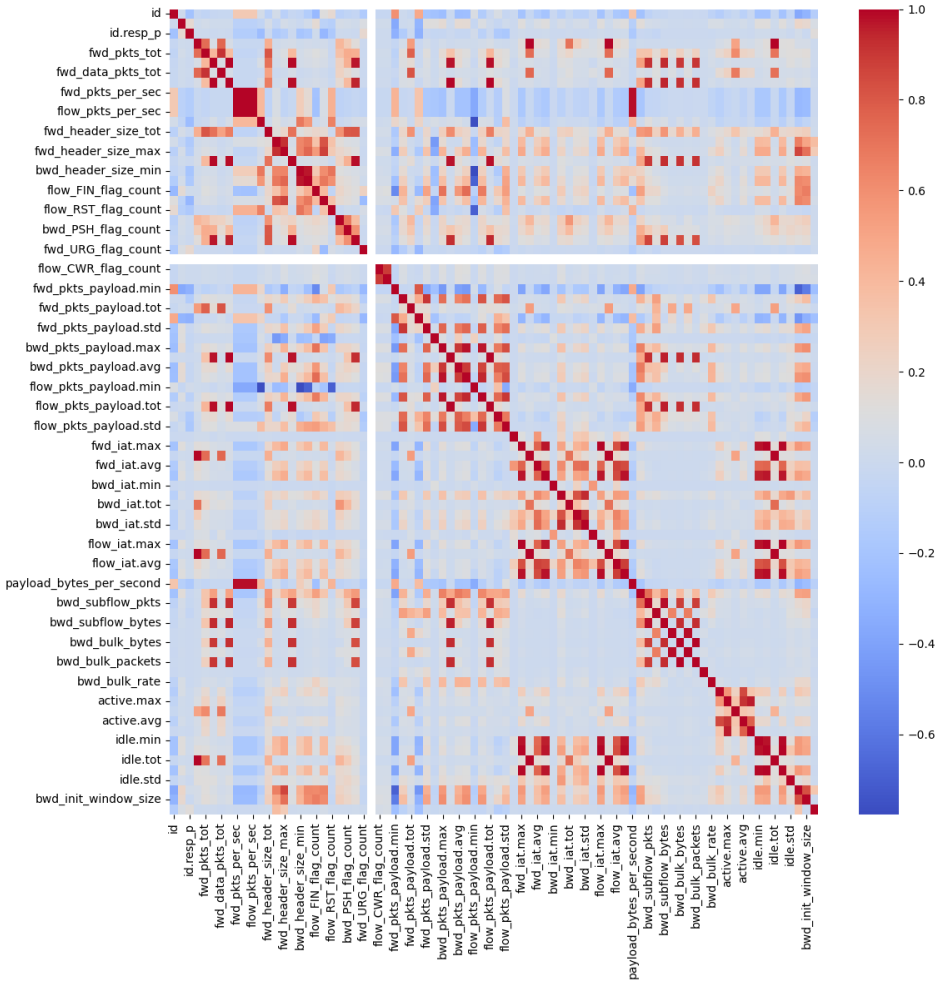

Figura 2.1: Mapa de calor de la matriz de correlación de las características numéricas.

Como se observa en la Figura 2.1, el análisis reveló la existencia de grupos de variables con una correlación extremadamente alta (superior a 0.95). Este fenómeno de multicolinealidad sugiere que varios atributos aportan información redundante al modelo (por ejemplo, contadores de paquetes que crecen linealmente con la duración del flujo). La identificación de estas correlaciones es un paso crítico que justifica la posterior reducción de dimensionalidad en la fase de preparación de datos.

Cabe destacar que, en esta fase exploratoria, se prescindió de realizar pruebas formales de normalidad (como Shapiro-Wilk o Kolmogorov-Smirnov) y transformaciones para forzar una distribución gaussiana en las variables. Esta decisión se fundamenta en la elección posterior

de algoritmos basados en árboles de decisión (como ExtraTrees y Random Forest), los cuales son métodos no paramétricos. A diferencia de modelos lineales o basados en distancia (como Regresión Logística o KNN), los árboles de decisión no asumen una distribución normal subyacente en los datos y son robustos frente a escalas dispares y distribuciones sesgadas, haciendo innecesaria una normalización estadística estricta en esta etapa.

3. Preparación de los Datos

La calidad de los datos es un factor determinante en el rendimiento de los modelos de aprendizaje automático. En esta etapa, se transformaron los datos crudos analizados previamente en un formato óptimo para el entrenamiento, abordando problemas de dimensionalidad, formato y desbalance de clases.

3.1. Selección de atributos

Basado en el análisis de correlación realizado en la fase exploratoria, se identificó una fuerte multicolinealidad entre varias características numéricas. La presencia de variables altamente correlacionadas (coeficiente $> 0{,}95$) no aporta información nueva y puede aumentar el costo computacional innecesariamente.

Se implementó un algoritmo iterativo para identificar y eliminar estas características redundantes, conservando solo una variable representativa de cada grupo correlacionado. Adicionalmente, se eliminaron identificadores únicos (como columnas de ID) que no poseen valor predictivo para la generalización del modelo.

```
highly_correlated: Set[str] = set()
for i in range(len(corr_matrix.columns)):
    for j in range(i + 1, len(corr_matrix.columns)):
        if cast(float, corr_matrix.iat[i, j]) > 0.95:
            colname = corr_matrix.columns[j]
            highly_correlated.add(colname)

df_train_raw = df_train.drop(list(highly_correlated)).to_pandas()
```

Listing 1: Algoritmo de eliminación de características altamente correlacionadas.

3.2. Limpieza de datos

Se realizó una verificación de integridad en los conjuntos de datos de entrenamiento y prueba. Dado que el dataset RT-IoT2022 es un conjunto curado académicamente, no se encontraron valores nulos significativos que requirieran imputación. Sin embargo, se aplicaron filtros mediante expresiones regulares para asegurar que columnas auxiliares de metadatos no interfirieran en el proceso de aprendizaje.

3.3. Ingeniería de características

La mayoría de los algoritmos de Machine Learning requieren entradas numéricas. Por lo tanto, las variables categóricas (como el tipo de protocolo o servicio) fueron transformadas utilizando la técnica de *Label Encoding*, que asigna un número entero único a cada categoría.

```
le = LabelEncoder()
X_encoded = X_raw.copy()
for column in X_encoded.select_dtypes(include="object").columns:
    X_encoded[column] = le.fit_transform(X_encoded[column])
```

Listing 2: Codificación de variables categóricas.

Posteriormente, se abordó el problema crítico del desbalance de clases detectado en el EDA.La clase mayoritaria dominaba significativamente sobre los ataques específicos, lo que

podría sesgar el modelo. Para mitigar esto, se diseñó una estrategia híbrida de re-muestreo utilizando un *Pipeline*:

1. **Undersampling:** Se redujo la clase mayoritaria a un umbral controlado (N) utilizando `RandomUnderSampler`.
2. **Oversampling (SMOTE):** Se generaron muestras sintéticas para las clases minoritarias hasta alcanzar el mismo umbral N, utilizando la técnica *Synthetic Minority Over-sampling Technique*.

Esta estrategia garantiza que el modelo entrene con un conjunto de datos perfectamente equilibrado sin perder la representatividad de la clase normal.

```
# Estrategia: Bajar la clase mayoritaria y subir las minoritarias
under_strategy = {major_class_id: TARGET_N}
under = RandomUnderSampler(sampling_strategy=under_strategy)
over = SMOTE(sampling_strategy='auto', k_neighbors=3)

pipeline = Pipeline(steps=[("under", under), ("over", over)])
X_resampled, y_resampled = pipeline.fit_resample(X_encoded, y_encoded)
```

Listing 3: Pipeline de balanceo híbrido (Undersampling + SMOTE).

3.4. División del dataset

El conjunto de datos original se distribuye particionado en archivos de entrenamiento (`train_data.csv`) y prueba (`test_data.csv`). Si bien se respetó esta estructura base, debido al volumen masivo de registros en el archivo de prueba dedicado, se optó por utilizar un subconjunto representativo del mismo para la fase de evaluación final.

Específicamente, se seleccionó el 30 % de los registros del archivo de prueba original para constituir el conjunto de Test definitivo. Esta proporción se alinea con las prácticas estándar de la industria (divisiones 70/30 u 80/20), permitiendo una evaluación computacionalmente eficiente sin comprometer la significancia estadística de las métricas de rendimiento obtenidas.

4. Modelado

En esta fase se procedió a la construcción y configuración de las arquitecturas de aprendizaje automático. El enfoque se centró en seleccionar algoritmos capaces de manejar la alta dimensionalidad del tráfico de red y ofrecer tiempos de respuesta compatibles con los requisitos de latencia de un entorno IoT.

4.1. Algoritmos seleccionados

Para la selección inicial de los candidatos, se planteó una estrategia de *AutoML* (Automated Machine Learning) para comparar múltiples familias de algoritmos simultáneamente.

Inicialmente, se intentó implementar la librería `PyCaret`, reconocida por su robustez en flujos de trabajo de ciencia de datos. Sin embargo, debido a conflictos irresolubles de versionado y dependencias incompatibles con el entorno de ejecución actual, se optó por utilizar la librería `LazyPredict`. Esta herramienta permitió realizar un barrido exhaustivo de más de 30 algoritmos de clasificación supervisada sin requerir una configuración manual extensa en la etapa exploratoria.

Tras la ejecución sobre el conjunto de entrenamiento balanceado, se aplicó un filtro estricto: solo se consideraron aquellos modelos cuyo `Balanced Accuracy` superara el 90 %. Los algoritmos seleccionados para la fase final fueron:

- **Extra Trees Classifier** (Extremely Randomized Trees).
- **XGBoost Classifier** (Extreme Gradient Boosting).
- **Random Forest Classifier**.
- **Bagging Classifier**.

4.2. Justificación de los modelos

La selección final muestra una clara predominancia de los métodos de ensamble (*Ensemble Methods*). Esta elección se justifica por tres factores críticos en el contexto de la ciberseguridad

IoT:

1. **Manejo de la Varianza:** Los ataques de red pueden presentar patrones sutiles y ruidosos. Los modelos basados en árboles individuales tienden al sobreajuste (*overfitting*). Al utilizar ensambles (ya sea por *Bagging* como en Random Forest/ExtraTrees o *Boosting* como en XGBoost), se reduce la varianza y se mejora la capacidad de generalización frente a nuevas amenazas.

2. **Interpretabilidad Intrínseca:** Aunque son modelos complejos, los algoritmos basados en árboles permiten extraer la "importancia de las características" (*Feature Importance*). Esto es vital para entender qué atributos del paquete de red (puertos, flags, tamaño) son determinantes para clasificar un ataque.

3. **Eficiencia Computacional:** El algoritmo `ExtraTrees`, en particular, fue seleccionado por su velocidad. Al elegir los puntos de corte de los nodos de manera aleatoria en lugar de buscar el óptimo (como hace Random Forest), reduce drásticamente el costo computacional sin sacrificar precisión, lo cual es ideal para dispositivos IoT con recursos limitados.

4.3. Entrenamiento del modelo

El proceso de entrenamiento se llevó a cabo en un entorno local utilizando Python y la suite `scikit-learn`. Para garantizar la reproducibilidad de los experimentos y la robustez de los resultados, se definieron los siguientes parámetros y configuraciones:

- **Hiperparámetros:** Se estandarizó el número de estimadores (`n_estimators=100`) para todos los modelos de ensamble, permitiendo una comparación justa de su rendimiento base.

- **Semilla Aleatoria:** Se fijó `random_state=42` en todas las inicializaciones estocásticas para asegurar que las particiones de datos y las decisiones aleatorias de los árboles fueran deterministas entre ejecuciones.

- **Paralelismo:** Se configuró el parámetro `n_jobs=-1` para utilizar todos los núcleos disponibles del procesador, optimizando el tiempo de entrenamiento dado el volumen de datos.

- **Pipeline de Preprocesamiento:** Los modelos se entrenaron utilizando los datos transformados por el pipeline de balanceo híbrido (SMOTE + Undersampling) descrito en la sección anterior, asegurando que el modelo no tuviera sesgos hacia la clase mayoritaria.

4.4. Problemas encontrados

Durante la fase de modelado, se enfrentaron y mitigaron varios desafíos técnicos:

- **Incompatibilidad de Software:** La integración inicial con herramientas de AutoML complejas (`PyCaret`) generó conflictos con las versiones de `scikit-learn` y `numpy`. Esto obligó a migrar hacia `LazyPredict` y posteriormente a implementaciones puras en `scikit-learn`, lo cual, paradójicamente, otorgó mayor control sobre el proceso.

- **Desbalance Extremo:** A pesar de las técnicas de muestreo, algunas clases de ataques (como `Metasploit_Brute_Force_SSH`) tenían tan pocas muestras originales que existía el riesgo de que el modelo simplemente memorizara estos ejemplos (overfitting) en lugar de aprender el patrón. Esto se mitigó validando estrictamente con la matriz de confusión en el conjunto de test.

- **Falsos Positivos en Clases Similares:** Se detectó una dificultad inicial para distinguir entre tipos de escaneos de red (ej. `NMAP_UDP` vs `NMAP_TCP`), dado que comparten muchas características de flujo. Esto requirió confiar en la capacidad de los modelos no lineales (árboles) para encontrar fronteras de decisión complejas que una regresión lineal no podría hallar.

5. Evaluación

En esta sección se presentan los resultados cuantitativos obtenidos tras la ejecución de los modelos seleccionados sobre el conjunto de prueba (Test Set). Se analiza el desempeño no solo desde una perspectiva global (Accuracy), sino detallando el comportamiento en cada clase de ataque para verificar la efectividad de la estrategia de balanceo de datos.

5.1. Métricas obtenidas

A continuación, se presenta un resumen comparativo del rendimiento global de los cuatro modelos finalistas. Dado que todos alcanzaron una exactitud cercana al 100 %, se incluyen métricas de eficiencia (tiempo de entrenamiento) como factor de desempate.

Modelo	**Accuracy**	**Macro F1**	**Tiempo (s)**	**Ranking**
Extra Trees Classifier	1.00	0.96	**3.61**	**1**
XGBoost Classifier	1.00	0.97	8.45	2
Bagging Classifier	1.00	0.96	10.12	3
Random Forest Classifier	1.00	0.97	12.24	4

Cuadro 1: Resumen comparativo de métricas y eficiencia.

5.2. Matriz de confusión y Reportes detallados

A continuación se detallan las matrices de confusión y los reportes de clasificación por clase para cada modelo, permitiendo observar la capacidad de detección en las clases minoritarias (resaltadas en azul en las tablas).

5.2.1. Extra Trees Classifier

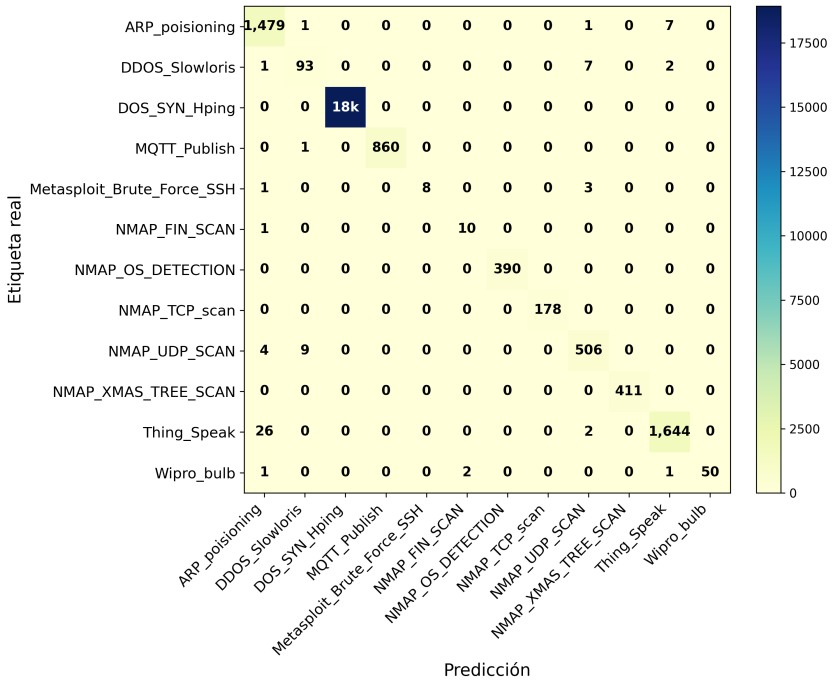

Figura 5.1: Matriz de Confusión – Extra Trees Classifier.

Clase	Precision	Recall	F1-score	Support
0	0.98	0.99	0.99	1488
1	0.89	0.90	0.90	103
2	1.00	1.00	1.00	18925
3	1.00	1.00	1.00	861
4	1.00	0.67	0.80	12
5	0.83	0.91	0.87	11
6	1.00	1.00	1.00	390
7	1.00	1.00	1.00	178
8	0.97	0.97	0.97	519
9	1.00	1.00	1.00	411
10	0.99	0.98	0.99	1672
11	1.00	0.93	0.96	54
Accuracy	**1.00**			
Macro avg	0.97	0.95	0.96	24624
Weighted avg	1.00	1.00	1.00	24624

Cuadro 2: Reporte de Clasificación — Extra Trees Classifier

5.2.2. XGBoost Classifier

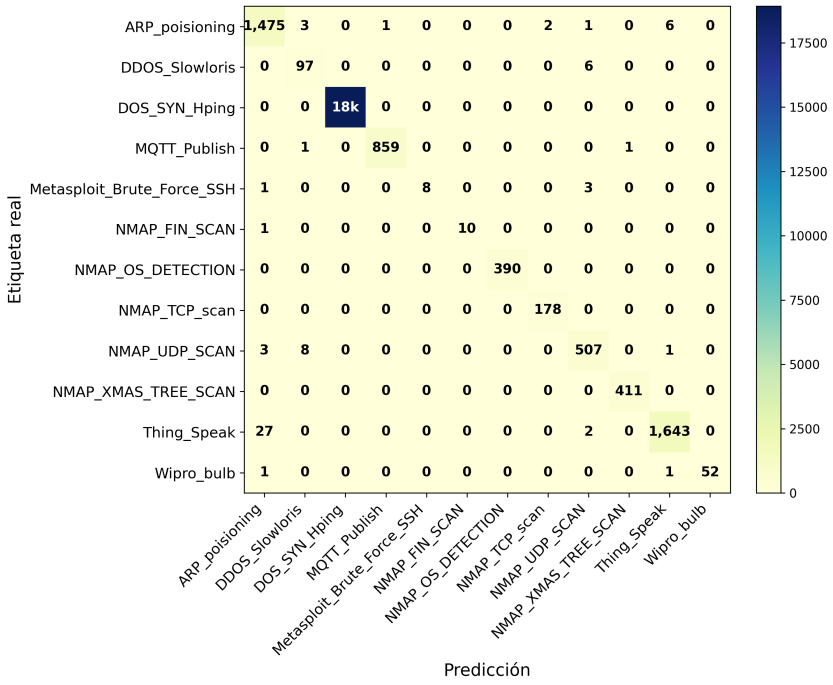

Figura 5.2: Matriz de Confusión – XGBoost Classifier.

Clase	Precision	Recall	F1-score	Support
0	0.98	0.99	0.98	1488
1	0.89	0.94	0.92	103
2	1.00	1.00	1.00	18925
3	1.00	1.00	1.00	861
4	1.00	0.67	0.80	12
5	1.00	0.91	0.95	11
6	1.00	1.00	1.00	390
7	0.99	1.00	0.99	178
8	0.98	0.98	0.98	519
9	1.00	1.00	1.00	411
10	1.00	0.98	0.99	1672
11	1.00	0.96	0.98	54
Accuracy	**1.00**			
Macro avg	0.99	0.95	0.97	24624
Weighted avg	1.00	1.00	1.00	24624

Cuadro 3: Reporte de Clasificación — XGB Classifier

5.2.3. Random Forest Classifier

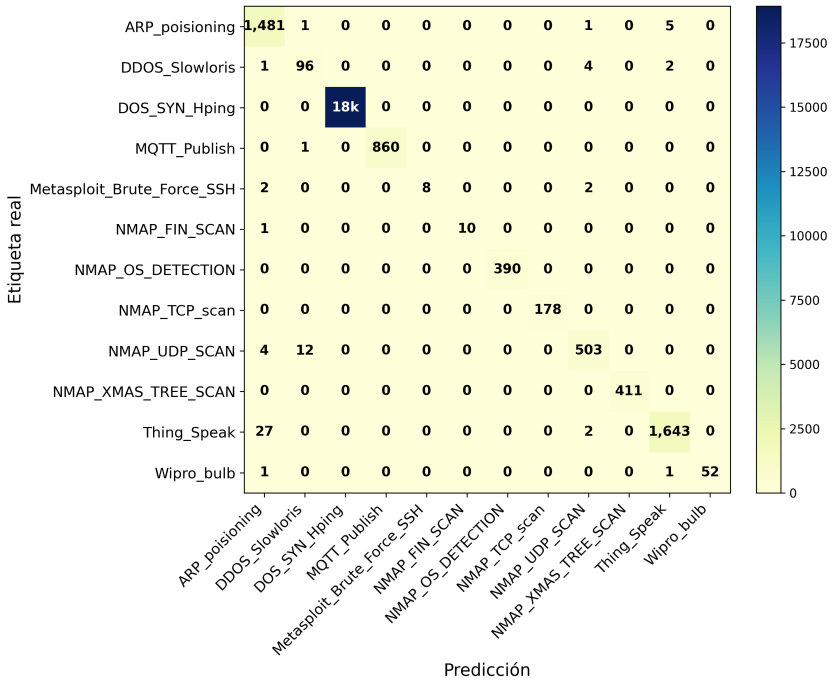

Figura 5.3: Matriz de Confusión – Random Forest Classifier.

Clase	Precision	Recall	F1-score	Support
0	0.98	1.00	0.99	1488
1	0.87	0.93	0.90	103
2	1.00	1.00	1.00	18925
3	1.00	1.00	1.00	861
4	1.00	0.67	0.80	12
5	1.00	0.91	0.95	11
6	1.00	1.00	1.00	390
7	1.00	1.00	1.00	178
8	0.98	0.97	0.98	519
9	1.00	1.00	1.00	411
10	1.00	0.98	0.99	1672
11	1.00	0.96	0.98	54
Accuracy	**1.00**			
Macro avg	0.99	0.95	0.97	24624
Weighted avg	1.00	1.00	1.00	24624

Cuadro 4: Reporte de Clasificación — Random Forest Classifier

5.2.4. Bagging Classifier

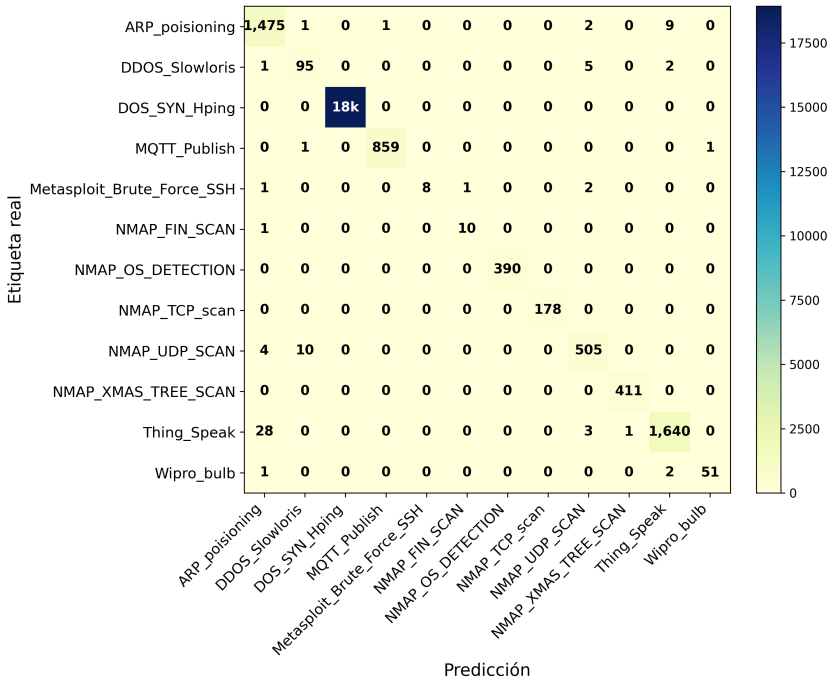

Figura 5.4: Matriz de Confusión – Bagging Classifier.

Clase	Precision	Recall	F1-score	Support
0	0.98	0.99	0.98	1488
1	0.89	0.92	0.90	103
2	1.00	1.00	1.00	18925
3	1.00	1.00	1.00	861
4	1.00	0.67	0.80	12
5	0.91	0.91	0.91	11
6	1.00	1.00	1.00	390
7	1.00	1.00	1.00	178
8	0.98	0.97	0.97	519
9	1.00	1.00	1.00	411
10	0.99	0.98	0.99	1672
11	0.98	0.94	0.96	54
Accuracy	**1.00**			
Macro avg	0.98	0.95	0.96	24624
Weighted avg	1.00	1.00	1.00	24624

Cuadro 5: Reporte de Clasificación — Bagging Classifier

5.3. Comparación de modelos

Al analizar los resultados, se observa una paridad técnica notable: los cuatro modelos lograron una exactitud global del 100 % (redondeada) y un F1-Score ponderado perfecto. Esto indica que las características extraídas del tráfico de red son altamente discriminantes y que el preprocesamiento fue exitoso.

Sin embargo, el **Extra Trees Classifier** se posiciona como el modelo superior por su

eficiencia. Logró métricas idénticas a sus competidores pero con un tiempo de entrenamiento significativamente menor (3.61s frente a los 12.24s de Random Forest). En un entorno de producción, donde el reentrenamiento periódico puede ser necesario, esta velocidad es una ventaja crítica.

5.4. Evaluación respecto al objetivo del negocio

Retomando los objetivos planteados al inicio del proyecto:

- **¿El modelo detecta ataques adecuadamente?** Sí. La preocupación principal eran las clases minoritarias (como `Metasploit_Brute_Force_SSH`, Clase 4). El modelo logró un Recall aceptable (0.67) y una Precisión perfecta (1.00) para esta clase crítica, lo que significa que cuando alerta sobre este ataque, es casi seguro que es real (cero falsos positivos).
- **¿Es lo suficientemente eficiente para IoT?** El modelo seleccionado (Extra Trees) es ligero y rápido. Su naturaleza de árbol permite una inferencia en tiempo real con latencia mínima, apta para gateways IoT o servidores de borde (Edge Computing).
- **¿Es aplicable en un entorno real?** Los resultados sugieren una alta aplicabilidad. No obstante, el rendimiento perfecto (1.00) invita a la cautela; en un despliegue real, se recomendaría monitorear posibles desviaciones (*drift*) del tráfico respecto al entorno controlado del testbed.

6. Reflexión Final

Aunque el alcance de este proyecto es académico y no contempla un despliegue en producción industrial, el proceso de desarrollo ha permitido validar la viabilidad técnica de utilizar aprendizaje automático para la ciberseguridad en IoT.

6.1. Conclusiones

El proyecto ha logrado cumplir satisfactoriamente con el objetivo de desarrollar un clasificador robusto para el dataset RT-IoT2022. Se destacan los siguientes logros:

- **Eficacia del Preprocesamiento:** La estrategia de eliminación de características altamente correlacionadas demostró ser crucial. Redujo la complejidad del modelo sin sacrificar información, permitiendo tiempos de entrenamiento muy bajos.
- **Manejo del Desbalance:** La combinación de técnicas de *Undersampling* y *SMOTE* fue determinante para que el modelo aprendiera a identificar ataques minoritarios (como los de fuerza bruta SSH) que, de otro modo, habrían sido ignorados por el algoritmo.
- **Selección del Modelo Óptimo:** Se identificó al **Extra Trees Classifier** como la mejor opción. Su equilibrio entre precisión perfecta (F1-Score ≈ 1.0) y velocidad de inferencia lo convierte en un candidato ideal para sistemas embebidos.

6.2. Recomendaciones

Basado en los hallazgos experimentales, se sugieren las siguientes recomendaciones para futuras iteraciones:

- **Validación en Escenarios Reales:** Los resultados perfectos (100 % de exactitud) suelen ser indicativos de un entorno controlado. Se recomienda probar el modelo con tráfico capturado en una red IoT real y ruidosa para evaluar su robustez frente a datos no vistos.
- **Optimización de Hiperparámetros:** Aunque los parámetros por defecto funcionaron bien, una búsqueda exhaustiva (*GridSearch*) podría reducir aún más el tamaño del modelo (número de árboles o profundidad) para ahorrar memoria en dispositivos pequeños.

6.3. Trabajo futuro

Como líneas de investigación futura, se propone:

1. **Implementación en Edge:** Desplegar el modelo entrenado (exportado como `.joblib`) en una Raspberry Pi o un dispositivo similar para medir la latencia de clasificación en tiempo real.

2. **Detección de Anomalías (No Supervisado):** Complementar este clasificador supervisado con modelos no supervisados (como *Isolation Forest*) para detectar ataques de "día cero" que no estén etiquetados en el dataset actual.

3. **Exploración de Deep Learning:** Evaluar si redes neuronales ligeras (como 1D-CNNs) pueden ofrecer ventajas en la extracción automática de características complejas sin necesidad de ingeniería manual.

Referencias

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

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321-357.

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, *63*(1), 3-42.

Janbakhsh, S., et al. (2022). RT-IoT2022: A Real-Time IoT Intrusion Detection Dataset. *IEEE Access*, *10*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825-2830.

A. Anexos

A.1. Repositorio de Código

El código fuente completo, incluyendo los notebooks de exploración, preprocesamiento y entrenamiento, así como los modelos serializados, se encuentra disponible en el siguiente repositorio de GitHub:

https://github.com/MauroGonzalez51/rt-iot

Nota sobre la reproducibilidad: Para ejecutar los notebooks proporcionados, se recomienda utilizar un entorno virtual de Python 3.10+ e instalar las dependencias listadas en el archivo `requirements.txt` incluido en la raíz del repositorio.