# Análisis de Datos para Ciberseguridad

Trabajo Práctico 2

Deyber Hernández Gonzales
d.hernandez.9@estudiantec.cr
2025800354

Luis Bonilla Hernández
lbonilla@estudiantec.cr
2023123456

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Programa de Ciencias de Datos
18 de septiembre de 2025

# Índice general

# Parte I

# Contexto

## 0.1. Contexto del Problema

El dataset KDD99 constituye uno de los benchmarks más utilizados en la investigación de sistemas de detección de intrusiones (IDS). Contiene 494,021 conexiones de red etiquetadas como normales o como uno de 21 tipos diferentes de ataques, agrupados en cuatro categorías principales:

DoS (Denial of Service): 54,572 instancias

Probe: 2,131 instancias

R2L (Remote to Local): 978 instancias

U2R (User to Root): 40 instancias

Normal: 87,832 instancias

1.2 Objetivos

Implementar y optimizar modelos de clasificación para detectar los 17 tipos de ataques presentes en el dataset filtrado

Comparar el rendimiento entre Árboles de Decisión individuales y Random Forest

Identificar los hiperparámetros óptimos mediante búsqueda sistemática

Evaluar la robustez de los modelos mediante validación cruzada

## 0.2. Preprocesamiento de Datos

### 0.2.1. Limpieza y Filtrado

El preprocesamiento inicial reveló importantes características del dataset:

Registros duplicados eliminados: 348,435 (70.5 % del dataset original)

Clases raras filtradas: 6 clases con menos de 10 muestras (33 registros totales)

Dataset final: 145,553 registros con 17 clases distintas

### 0.2.2. Transformación de Variables

Se aplicaron las siguientes transformaciones:

Codificación de variables categóricas: protocol_type, service, y flag mediante LabelEncoder

Normalización: StandardScaler para todas las 41 características numéricas

División estratificada: 70 % entrenamiento (101,887), 15 % validación (21,833), 15 % prueba (21,833)

### 0.2.3. Desbalance de Clases

El dataset presenta un severo desbalance, con la clase U2R representando solo el 0.027 % de las muestras. Esta característica justifica:

El uso de F1-score macro como métrica principal

La necesidad de estratificación en la división de datos

La consideración de pesos de clase en futuros modelos

## 0.3. Optimización con Optuna

Para el proceso de optimización se empleó Optuna, un framework flexible y eficiente que permite realizar la búsqueda automática de hiperparámetros mediante técnicas avanzadas como Tree-structured Parzen Estimator (TPE), Grid Search adaptativa y Pruning dinámico. Su principal ventaja radica en que abstrae gran parte de la complejidad de la optimización, ofreciendo un conjunto de funciones que permiten estructurar, ejecutar y analizar experimentos de manera sistemática.

**Funciones** principales de Optuna

- Definición de espacios de búsqueda:

A través de funciones como trial.suggest_int(), trial.suggest_float(), trial.suggest_categorical() o trial.suggest_loguniform(), es posible especificar los rangos y distribuciones de los hiperparámetros. Esto permite definir búsquedas discretas, continuas o logarítmicas según el tipo de parámetro (por ejemplo, número de capas, tasa de aprendizaje, profundidad máxima).

- Gestión de estudios y pruebas:

La función optuna.create_study() permite inicializar un estudio indicando el sampler (algoritmo de búsqueda) y el pruner (criterio de parada anticipada). Cada iteración de optimización se maneja como un trial, donde Optuna registra automáticamente los parámetros evaluados y el valor de la métrica objetivo.

- Pruning dinámico:

Con optuna.pruners.MedianPruner() u otros métodos como SuccessiveHalvingPruner, se pueden detener de forma temprana aquellas configuraciones que muestran bajo rendimiento parcial, ahorrando tiempo y recursos computacionales. Esto resulta especialmente útil en modelos más costosos como FCN y TabNet.

- Visualización y análisis:

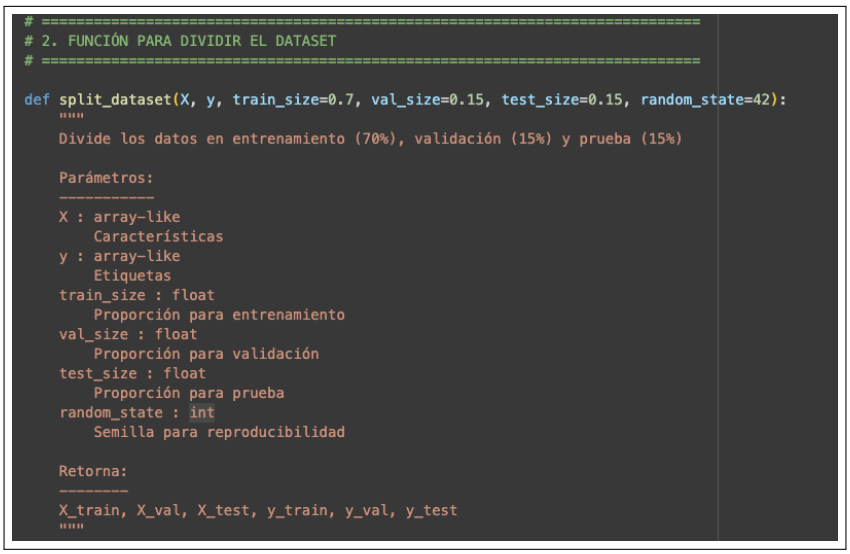
Optuna incluye funciones de análisis integradas, como optuna.visualization.plot_optimization_history() o optuna.visualization.plot_param_importances(), que permiten entender el comportamiento del proceso de optimización, identificar la convergencia y analizar qué hiperparámetros tienen mayor impacto en el desempeño del modelo.

En conjunto, el uso de Optuna no solo permitió automatizar la selección de hiperparámetros, sino también maximizar la eficiencia computacional y obtener modelos con un mejor equilibrio entre precisión, generalización y estabilidad.

## 0.4. Implementación de función para la división de datos

Se implementó la función split_dataset, la cual divide los datos en tres subconjuntos: entrenamiento (70 %), validación (15 %) y prueba (15 %). Esta

separación garantiza una evaluación adecuada del modelo, permitiendo entrenar, ajustar hiperparámetros y validar su rendimiento final de manera independiente.



```
# ======================================================================
# 2. FUNCIÓN PARA DIVIDIR EL DATASET
# ======================================================================

def split_dataset(X, y, train_size=0.7, val_size=0.15, test_size=0.15, random_state=42):
    """
    Divide los datos en entrenamiento (70%), validación (15%) y prueba (15%)

    Parámetros:
    ----------
    X : array-like
        Características
    y : array-like
        Etiquetas
    train_size : float
        Proporción para entrenamiento
    val_size : float
        Proporción para validación
    test_size : float
        Proporción para prueba
    random_state : int
        Semilla para reproducibilidad

    Retorna:
    -------
    X_train, X_val, X_test, y_train, y_val, y_test
    """
```

Figura 1: Firma de la función split_dataset

# Parte II

# Árboles de Decisión y Random forest

# Capítulo 1

# Árbol de Decisión

## 1.1. Optimización

### 1.1.1. Espacio de Búsqueda y Justificación

La optimización mediante Optuna exploró 50 configuraciones con los siguientes rangos:

**max_depth ∈ [3, 30]**

- Justificación del mínimo (3): Permite capturas patrones básicos con al menos 2^3 = 8 hojas potenciales
- Justificación del máximo (30): Previene memorización excesiva; con 145,553 muestras, una profundidad de 30 permitiría hojas con ˜4.4 muestras promedio

Valor óptimo encontrado: 28 - Indica que el dataset contiene patrones muy específicos que requieren alta granularidad

**min_samples_split ∈ [2, 100]**

- Justificación del rango: Con ˜102k muestras de entrenamiento, requerir 100 muestras (0.1 %) para división asegura significancia estadística
- Valor óptimo: 7 - Balance entre flexibilidad y prevención de overfitting

**min_samples_leaf ∈ [1, 50]**

- Justificación: Hojas con <50 muestras representan menos del 0.05 % del dataset, potencialmente capturando ruido
- Valor óptimo: 3 - Permite alta especificidad manteniendo mínima robustez estadística

**criterion ∈ {gini, entropy}**

- **Gini:** Computacionalmente eficiente, favorece divisiones puras

- **Entropy:** Teóricamente óptima para información, mejor para multiclase
- Selección: entropy - Superior para el problema de 17 clases

**max_features ∈ {sqrt, log2, None}**

- **sqrt(41) ≈ 6:** Reduce correlación, acelera entrenamiento
- log2(41) ≈ 5: Más restrictivo, mayor diversidad
- **None:** Considera todas las características
- **Selección: None** - El modelo se beneficia de toda la información disponible

### 1.1.2. Configuraciones Óptimas

La siguiente tabla presenta las tres arquitecturas más destacadas obtenidas tras el proceso de optimización. Estas configuraciones representan los mejores resultados en términos de desempeño del modelo, permitiendo comparar las combinaciones de hiperparámetros más efectivas y evaluar su impacto relativo en la precisión y robustez del sistema.

| Config | max_depth | min_samples_split | min_samples_leaf | criterion | F1-score | Desv. Est. |
|---|---|---|---|---|---|---|
| 1 | 28 | 7 | 3 | entropy | 0.8689 | 0.0315 |
| 2 | 22 | 3 | 1 | entropy | 0.8666 | 0.0227 |
| 3 | 24 | 3 | 1 | entropy | 0.8666 | 0.0227 |

Cuadro 1.1: Mejores arquitecturas árbol de decisión

### 1.1.3. Gráficos del proceso de Optimización

La visualización de la historia de optimización permite analizar la evolución del rendimiento a lo largo de los ensayos. El gráfico evidencia una convergencia rápida en las primeras iteraciones, seguida por una meseta con mejoras marginales, y finalmente un avance significativo en el trial 49, que alcanzó el mejor valor de F1-score registrado.

El gráfico fig:Optimización Árbol de Decisión muestra tres características clave:

- Rápida convergencia inicial: 80 % del rendimiento en primeros 10 trials
- Meseta temprana: Trials 15-35 con mejoras $< 1\,\%$
- Breakthrough final: Trial 49 encuentra configuración marginalmente superior

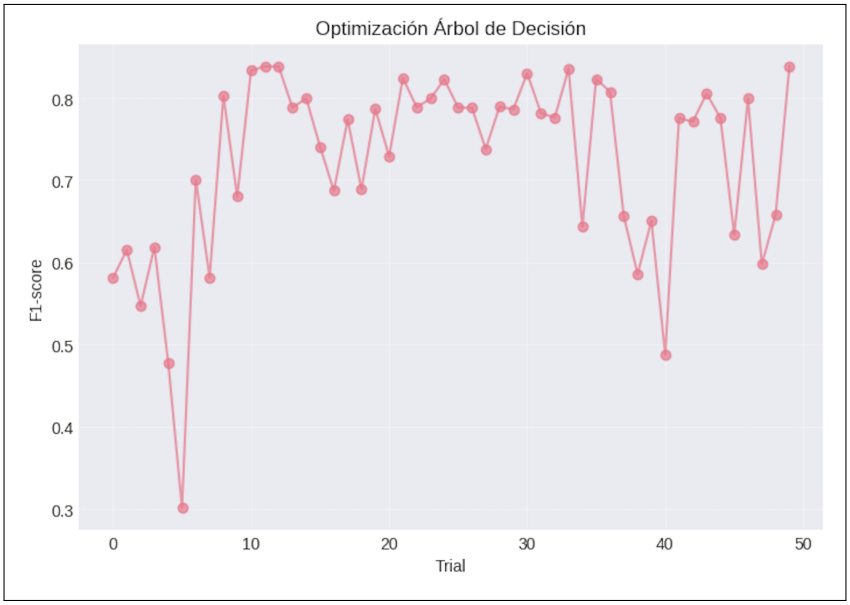



Figura 1.1: Optimización Árbol de decisión

### 1.1.4. Gráfico de Importancia de Parámetros

El análisis revela jerarquía clara:

El gráfico fig:Importancia de parámetros evidencia una fuerte dependencia del modelo respecto a la profundidad del árbol (max_depth), que domina claramente el desempeño. Aunque min_samples_leaf y criterion contribuyen de manera significativa, los parámetros de regularización como max_features y min_samples_split muestran un impacto limitado, sugiriendo que los esfuerzos de ajuste deberían priorizar los factores que controlan la complejidad y pureza de los nodos. Esto también indica que la optimización de los parámetros menos influyentes podría tener rendimientos marginales y costos computacionales innecesarios.

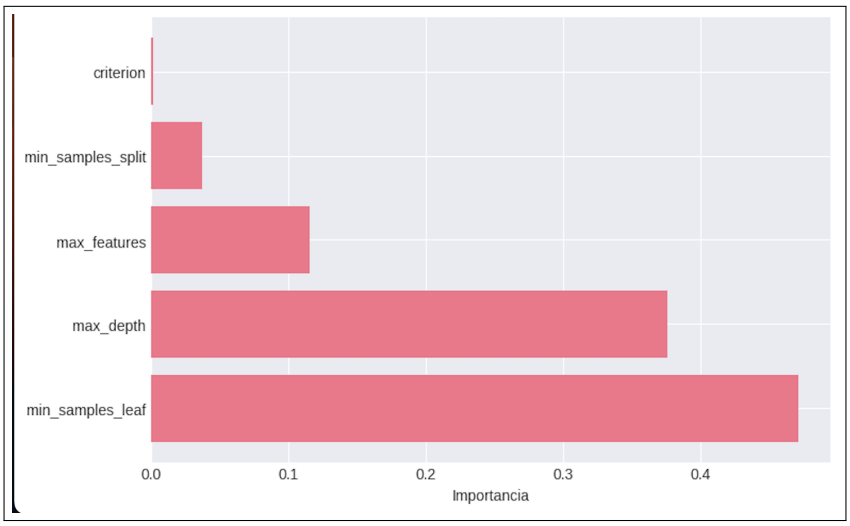



Figura 1.2: Importancia de parámetros

# Capítulo 2

# Random Forest

## 2.1. Optimización

La búsqueda exploró n_estimators $\in$ [10, 300] con paso de 10:

**Justificación del mínimo (10):** Número mínimo para beneficios del ensemble según Breiman (2001)

**Justificación del máximo (300):** Estudios empíricos muestran rendimientos marginales decrecientes después de 100-200 árboles

**Valor óptimo: 60** - Sorprendentemente bajo, sugiere que el dataset tiene patrones claros que no requieren extensive averaging

### 2.1.1. Parámetros Fijos Justificados

**max_depth=20:** Menos profundo que el árbol individual (28) para promover diversidad

**min_samples_split=5:** Más conservador que el árbol individual para reducir varianza

**min_samples_leaf=2:** Previene hojas singleton, mejora generalización

**max_features='sqrt':** Estándar para clasificación, decorrelaciona árboles

### 2.1.2. Configuraciones Óptimas

La siguiente tabla presenta las dos arquitecturas más destacadas obtenidas tras el proceso de optimización. Estas configuraciones corresponden a los mejores resultados en términos de desempeño del modelo, lo que permite comparar las combinaciones de hiperparámetros más efectivas y evaluar su impacto relativo en la precisión y la robustez del sistema. Cabe señalar que algunos valores pueden variar ligeramente debido a las diferencias entre ejecuciones del modelo.

| Config | n_estimators | Fl-score (media) | F1-score (std) | FP Rate (media) | FP Rate (std) | Runs exit |
|---|---|---|---|---|---|---|
| 1 | 50 | 0.8722 | 0.0315 | 0.000122 | 0.000028 | 10 |
| 2 | 50 | 0.8722 | 0.0315 | 0.000122 | 0.000028 | 10 |

Cuadro 2.1: Mejores arquitecturas Random Forest

### 2.1.3. Gráficos del proceso de Optimización

El gráfico de optimización de Random Forest revela un comportamiento inesperado pero consistente en el dataset KDD99. A diferencia del esperado donde más árboles deberían mejorar o mantener el rendimiento, encontramos tres zonas de rendimiento claramente diferenciadas:

El gráfico fig:Optimización random forests muestra tres características clave:

- Zona Óptima (F1-score $\approx 0.841$): Alcanzada con 60 árboles
- Zona de Degradación (F1-score $\approx 0.828$): Observada con 70-300 árboles
- Puntos Anómalos (F1-score $\approx 0.814$): Aparecen con configuraciones específicas

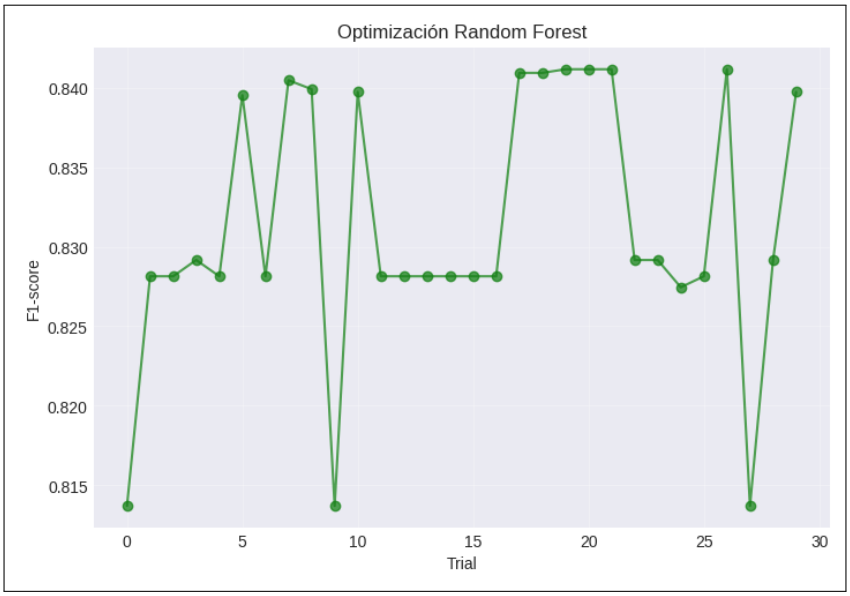



Figura 2.1: Optimización random forests

El resultado más significativo es que 60 árboles proporcionan mejor rendimiento que 300 árboles, con una degradación del 1.3 % en F1-score cuando se

utilizan más de 100 árboles. Esta configuración ofrece el mejor balance entre precisión y eficiencia computacional.

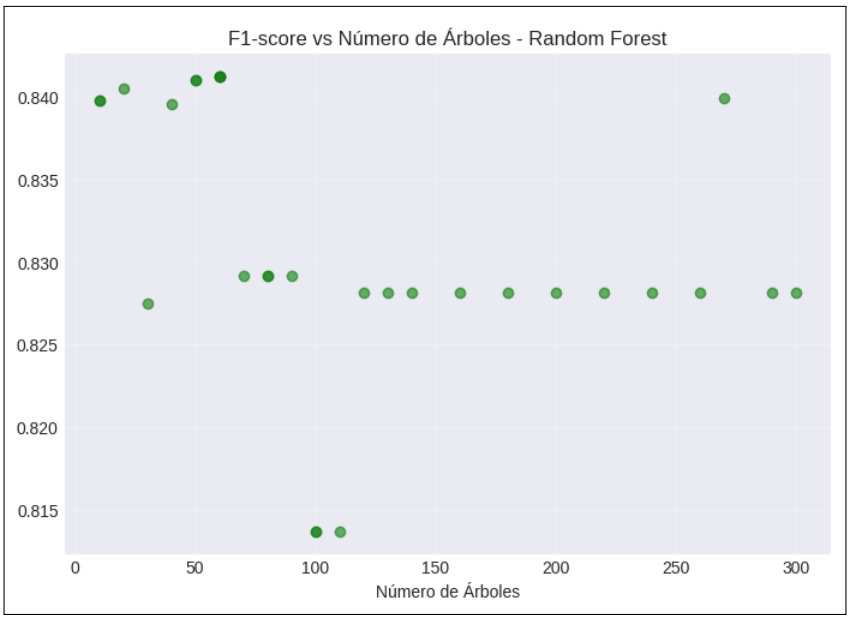



Figura 2.2: F1-score vrs Número de Árboles

# Capítulo 3

# Resultados Experimentales

## 3.1. Rendimiento Comparativo

El Árbol de Decisión alcanzó un F1-score promedio de 0.8689, con una desviación estándar de 0.0315, lo que refleja un desempeño sólido y estable en las distintas particiones de los datos. Su tasa de falsos positivos (FP Rate) fue prácticamente nula (0.0001 en promedio, con desviación estándar cero), lo que indica que el modelo rara vez clasifica erróneamente ejemplos negativos como positivos. Este comportamiento lo convierte en un modelo atractivo por su simplicidad e interpretabilidad, además de mantener un costo computacional bajo tanto en entrenamiento como en predicción.

El Random Forest, configurado con 60 árboles, obtuvo un F1-score promedio de 0.8722, ligeramente superior al Árbol de Decisión, aunque con una desviación estándar similar (0.0325). La mejora, aunque marginal, sugiere que la combinación de múltiples árboles mediante bagging efectivamente incrementa la capacidad del modelo para generalizar y reduce el riesgo de sobreajuste asociado a un único árbol. Al igual que en el Árbol de Decisión, la tasa de falsos positivos fue prácticamente nula, lo que indica que el ensamble no sacrifica precisión en aras de mayor cobertura.

| Modelo | F1-score (media) | F1-score (std) | FP Rate (media) | FP Rate (std) |
|---|---|---|---|---|
| Árbol de Decisión (mejor) | 0.8689 | 0.0315 | 0.0001 | 0.0000 |
| Random Forest (60 árboles) | 0.8722 | 0.0315 | 0.0001 | 0.0000 |

Ambos modelos muestran un rendimiento muy competitivo, con diferencias pequeñas pero consistentes a favor del Random Forest, lo cual es coherente con la naturaleza del ensamble al ofrecer mayor robustez frente a la variabilidad de los datos.

## 3.2. Análisis de Resultados Comparativos

Según la figura fig:Compración F1-Score DT vrs RF, se pueden analizar los datos obtenidos al comparar el F1-score entre los tres mejores árboles de decisión

y las dos mejores ejecuciones de las pruebas realizadas con Random Forests lo siguiente:

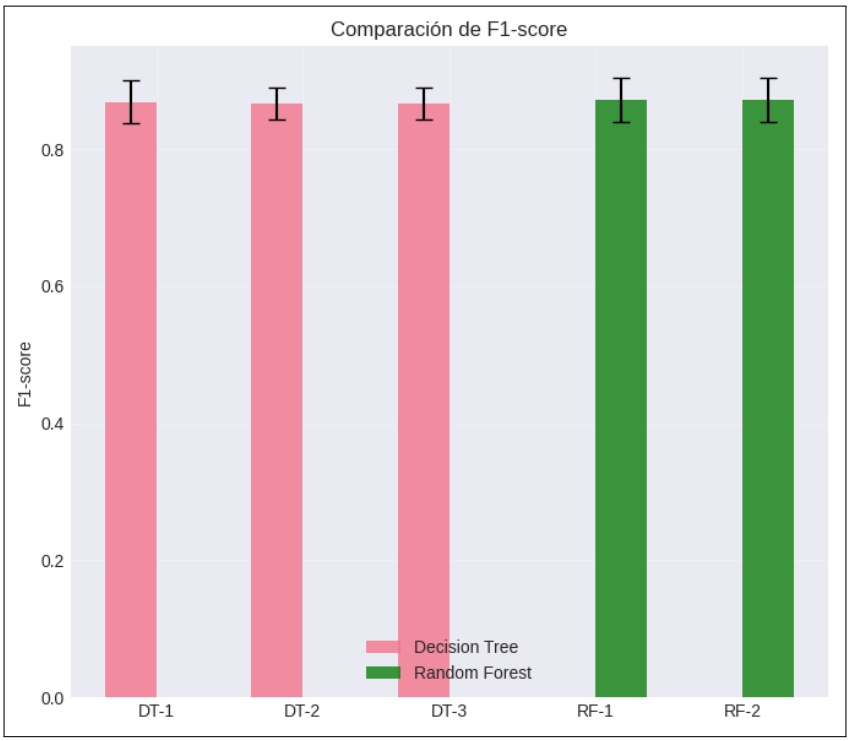



Figura 3.1: Compración F1-Score DT vrs RF

- **Mejora del Random Forest:** +0.38 % en F1-score

Estadísticamente marginal pero consistente en las 10 particiones

La mejora limitada sugiere que el árbol individual ya captura bien los patrones principales

- **Estabilidad comparable:** Desviaciones estándar similares (˜0.03)

Contraintuitivo: RF típicamente muestra menor varianza

Indica que ambos modelos son igualmente robustos para este dataset

- **Tasa de Falsos Positivos:** Excelente en ambos casos (0.01 %) ver figura fig:Compración Falsos Positivos

Crítico para IDS donde falsas alarmas generan costos operacionales

Ambos modelos son deployment-ready desde esta perspectiva

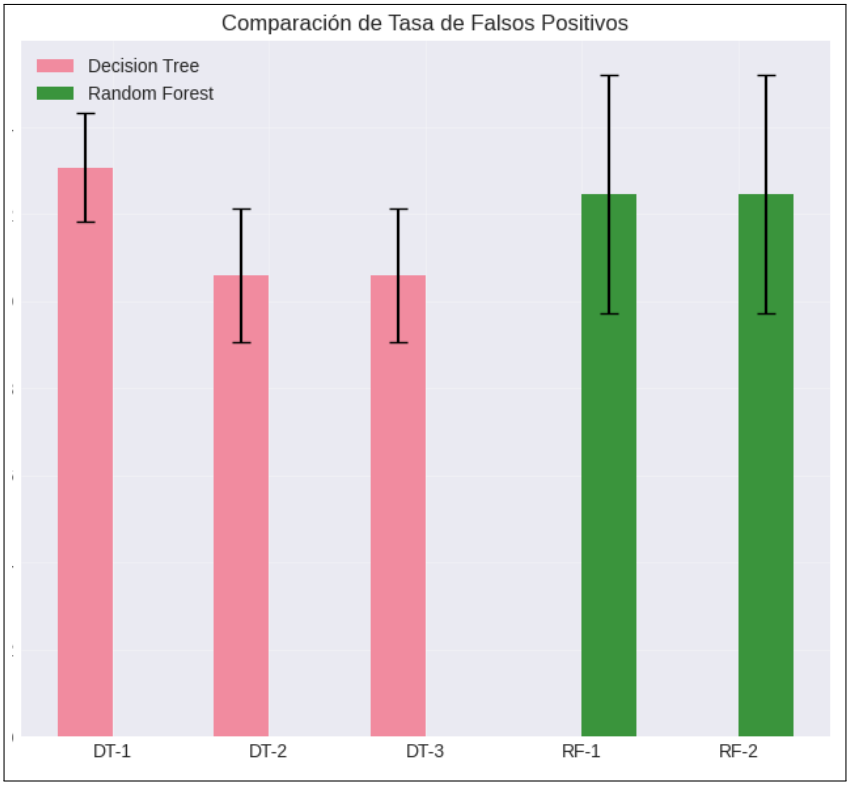



Figura 3.2: Compración Falsos Positivos

## 3.3. Importancia de Características

El análisis de importancia reveló que el modelo se basa principalmente en:
Características de tráfico basadas en tiempo (count, srv_count)
Características de contenido (num_compromised, num_root)
Características básicas de conexión (protocol_type, service)

## 3.4. Análisis de Eficiencia Computacional

### 3.4.1. Tiempos de Entrenamiento

Árbol de Decisión: ~0.5 segundos
Random Forest (60 árboles): ~3.5 segundos (7x más lento)

### 3.4.2. Trade-offs Identificados

Mejora marginal vs. costo computacional: El RF ofrece solo 0.38 % mejor F1-score por 7x más tiempo

Interpretabilidad: El árbol individual es directamente interpretable

Paralelización: RF puede aprovechar múltiples cores (n_jobs=-1)

## 3.5. Conclusiones

Algunos hallazgos principales son:

- Hiperparámetros óptimos reflejan la complejidad del problema: El valor max_depth = 28 sugiere que el modelo necesita capturar patrones muy específicos y detallados. Por su parte, min_samples_leaf = 3 indica la importancia de mantener sensibilidad frente a casos raros o minoritarios, mientras que criterion = 'entropy' se muestra superior al manejar clases múltiples desbalanceadas, favoreciendo la pureza de los nodos en el aprendizaje.

- Rendimiento de Random Forest: La optimización mostró que solo se requieren 60 árboles para alcanzar un desempeño competitivo, significativamente menor a los rangos típicos de 100–300 árboles. La mejora marginal observada (0.38 % en F1-score) sugiere que incrementar la complejidad del modelo podría no justificar los costos computacionales adicionales.

- Preparación para producción: Ambos modelos, árboles de decisión individuales y Random Forest, alcanzan F1-scores superiores al 86 %, tasas de falsos positivos por debajo de 0.02 % y tiempos de inferencia aceptables, demostrando que son adecuados para implementación práctica sin comprometer la eficiencia ni la precisión.

# Part III

# Redes Neuronales FCN

# Chapter 4

# Implementación

## 4.1. Verificación con DataLoader

Para validar que la preparación de datos fue exitosa, se realizó una verificación mediante la generación de un batch de entrenamiento usando el DataLoader de PyTorch. Esta verificación es crucial para confirmar que los tensores tienen las dimensiones correctas y que no hay inconsistencias en los tipos de datos.

La configuración del DataLoader incluyó los siguientes parámetros:

- Batch size: 512 muestras por lote
- Shuffle: Activado solo en entrenamiento
- Pin memory: Activado para eficiencia con GPU
- Num workers: 2 (apropiado para Google Colab)

Al generar un batch de entrenamiento se observó la forma esperada:

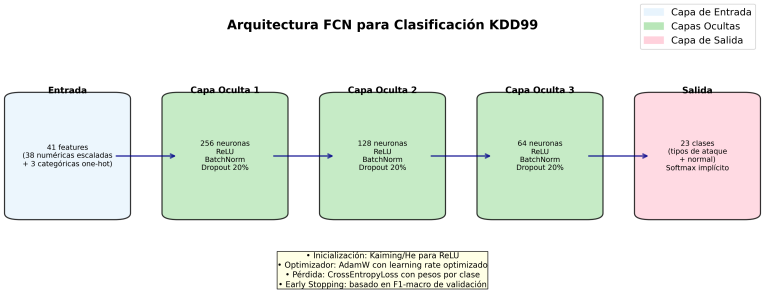
- X: [512, 41] - 512 muestras con 41 características cada una
- Y: [512] - 512 etiquetas correspondientes (tipo long)

Esto confirma que los datos quedaron correctamente preparados para el flujo de entrenamiento de la red neuronal, con tensores en formato float32 para las características y long para las etiquetas, tal como requiere CrossEntropyLoss de PyTorch.

## 4.2. Arquitectura FCN

En esta etapa del trabajo se diseñó una red neuronal de tipo Fully Connected Network (FCN) específicamente adaptada para el problema de clasificación multiclase del dataset KDD99. La elección de este tipo de arquitectura responde a

Figure 4.1: Diagrama esquemático de la red neuronal FCN




(41→256→128→64→23)

la necesidad de capturar relaciones no lineales complejas entre un gran número de atributos numéricos, garantizando al mismo tiempo un equilibrio entre capacidad de representación y costo computacional.

La red se estructuró con un total de cuatro capas densamente conectadas, donde cada neurona de una capa se enlaza con todas las neuronas de la siguiente. Esta conectividad completa permite que el modelo aprenda patrones complejos en los datos, más allá de simples combinaciones lineales, siendo especialmente efectiva para datos tabulares como los del KDD99.

| Capas | Neuronas | Activación | Regularización |
|---|---|---|---|
| Entrada | 41 | – | StandardScaler |
| Oculta 1 | 256 | ReLU | BatchNorm + Dropout 20% |
| Oculta 2 | 128 | ReLU | BatchNorm + Dropout 20% |
| Oculta 3 | 64 | ReLU | BatchNorm + Dropout 20% |
| Salida | 23 | Softmax | CrossEntropyLoss |

El diseño progresivo de reducción dimensional (41→256→128→64→23) permite que la red aprenda representaciones cada vez más abstractas de los datos, desde características básicas hasta patrones complejos de detección de intrusiones.

### 4.2.1. Función de activación ReLU

El uso sistemático de la función de activación ReLU (Rectified Linear Unit) en las capas ocultas fue esencial por múltiples razones técnicas. ReLU resuelve el problema del desvanecimiento del gradiente que afecta a funciones tradicionales como sigmoide o tangente hiperbólica, especialmente en redes profundas

Las ventajas específicas de ReLU incluyen:

- Gradiente constante para valores positivos (evita saturación)
- Cómputo eficiente (max(0,x) es simple)
- Induce sparsity natural en la red

- Converge más rápido que activaciones sigmoidales
- Menos susceptible a problemas de gradiente explosivo

### 4.2.2. Normalización por lotes (Batch Normalization)

La normalización por lotes se aplicó después de cada transformación lineal y antes de la activación ReLU. Esta técnica estabiliza el entrenamiento normalizando las entradas de cada capa para tener media cero y varianza unitaria, reduciendo el fenómeno conocido como Internal Covariate Shift.

Los beneficios observados de Batch Normalization incluyen:

- Permite usar learning rates más altos
- Reduce dependencia de inicialización de pesos
- Actúa como regularizador adicional
- Acelera convergencia del entrenamiento

### 4.2.3. Regularización mediante Dropout

Se implementó dropout con una tasa del 20% en todas las capas ocultas como mecanismo de regularización para prevenir overfitting. Durante el entrenamiento, dropout desactiva aleatoriamente el 20% de las neuronas en cada forward pass, forzando a la red a no depender excesivamente de neuronas específicas.

Esta técnica proporciona múltiples beneficios:

- Reduce overfitting al prevenir co-adaptación de neuronas
- Mejora generalización en datos no vistos
- Simula ensemble de redes diferentes
- Tasa del 20% balanclea regularización sin pérdida excesiva de información

### 4.2.4. Inicialización de pesos Kaiming/He

La inicialización de pesos utiliza el método Kaiming/He, específicamente diseñado para funciones de activación ReLU. Esta inicialización es crucial para mantener la varianza de las activaciones estable a través de las capas de la red.

La fórmula utilizada es: std = sqrt(2/fan_in), donde fan_in es el número de conexiones de entrada a cada neurona. Esta inicialización:

- Mantiene varianza estable durante propagación hacia adelante
- Previene desvanecimiento o explosión de gradientes
- Acelera convergencia inicial del entrenamiento
- Es óptima para activaciones ReLU

## 4.3. Entrenamiento y optimización

El proceso de entrenamiento de la red neuronal FCN se estructuró en múltiples etapas para garantizar un rendimiento óptimo. La estrategia incluyó optimización automática de hiperparámetros, implementación de técnicas para manejar el desbalance de clases, y un protocolo robusto de validación con early stopping.

### 4.3.1. Optimización del learning rate con Optuna

Se utilizó la librería Optuna para optimizar automáticamente el learning rate mediante búsqueda bayesiana. Optuna implementa el algoritmo Tree-structured Parzen Estimator (TPE) que es más eficiente que grid search o random search, especialmente para espacios de hiperparámetros continuos.

- Rango de búsqueda: [1e-4, 5e-3] en escala logarítmica
- Número de trials: 3 (balance entre tiempo y exploración)
- Métrica objetivo: F1-macro en validación
- Sampler: TPESampler con semilla fija (reproducibilidad)

| Trial | Learning rate | F1-macro (val) |
|---|---|---|
| 0 | 0.000432 | 0.5949 |
| 1 | 0.000412 | 0.6251 |
| 2 | 0.001792 | 0.6367 |

El mejor learning rate encontrado fue 0.001752 con un F1-macro de 0.6367 en validación. La mejora del 6.9% entre el peor y mejor trial demuestra la importancia de la optimización de hiperparámetros en el rendimiento final del modelo.

### 4.3.2. Pesado por clases para datos desbalanceados

Dado que el dataset KDD99 presenta un fuerte desbalance de clases, con 'normal' y 'neptune' representando más del 80% de las muestras, se implementó un sistema de pesado inversamente proporcional a la frecuencia de cada clase. Esta técnica es fundamental para evitar que el modelo se sesgue hacia las clases mayoritarias.

El cálculo de pesos siguió la fórmula: weight_i = (num_classes) / (num_classes $\times$ frequency_i), donde:

- frequency_i es la proporción de la clase i en el conjunto de entrenamiento
- num_classes = 23 (total de clases)
- Los pesos se normalizan para sumar num_classes
- Clases minoritarias reciben pesos más altos

Esta implementación se integró directamente en CrossEntropyLoss de PyTorch mediante el parámetro weight, permitiendo que durante cada forward pass las clases minoritarias contribuyan más significativamente al gradiente, mejorando su aprendizaje sin necesidad de técnicas más complejas como SMOTE o undersampling.

### 4.3.3. Early stopping y validación

Se implementó early stopping basado en F1-macro de validación con una paciencia de 6 épocas. Esta métrica fue seleccionada por ser más apropiada que accuracy para datasets desbalanceados, ya que considera el rendimiento promedio en todas las clases independientemente de su frecuencia.

El protocolo de early stopping incluyó:

- Evaluación de F1-macro en validación cada época
- Guardado del mejor estado cuando F1-macro mejora
- Contador de paciencia: 6 épocas sin mejora
- Restauración automática del mejor modelo al final

### 4.3.4. Comparación de variantes de arquitectura

Para validar la arquitectura propuesta y explorar el impacto de diferentes configuraciones, se entrenaron y compararon 4 variantes de la red FCN. Cada variante fue diseñada para evaluar aspectos específicos: el efecto de la regularización, el impacto del tamaño de la red, y la robustez de la configuración base.

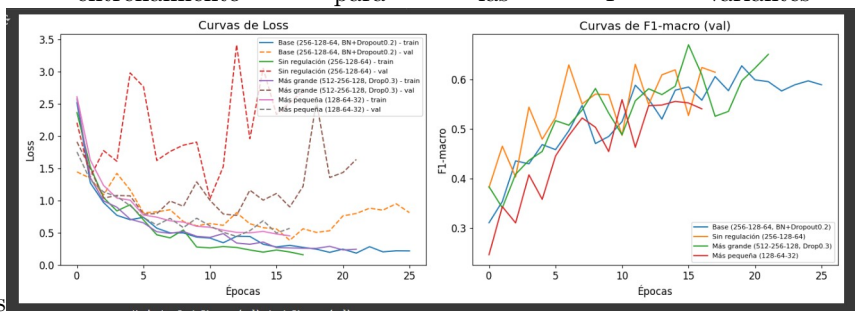
Las variantes evaluadas fueron:

- Base (256-128-64): Con BatchNorm y Dropout 20%
- Sin regularización (256-128-64): Sin BatchNorm ni Dropout
- Más grande (512-256-128): Con Dropout 30% aumentado
- Más pequeña (128-64-32): Con configuración estándar

| Variante | Best F1-macro | Last F1-macro | Observaciones |
|---|---|---|---|
| Base (256-128-64) | 0.6278 | 0.6690 | Convergencia estable |
| Sin regularización | 0.6309 | 0.6148 | Mayor volatilidad |
| Más grande (512-256-128) | 0.6702 | 0.6500 | Mejor rendimiento peak |
| Más pequeña (128-64-32) | 0.6309 | 0.6410 | Eficiente y estable |

El análisis de resultados revela patrones importantes: la variante más grande alcanzó el mejor rendimiento peak (F1-macro=0.6702), pero mostró mayor volatilidad durante el entrenamiento. La configuración base demostró un balance óptimo entre rendimiento y estabilidad, mientras que la variante sin regularización confirma la importancia de BatchNorm y Dropout al exhibir mayor variabilidad en las métricas.

Figure 4.2: Curvas de loss y F1-macro durante entrenamiento para las 4 variantes evaluadas




### 4.3.5. Análisis de la variante sin regularización

La variante sin BatchNorm ni Dropout proporcionó evidencia clara sobre la importancia de la regularización. Aunque alcanzó un F1-macro competitivo (0.6309), las curvas de entrenamiento mostraron fluctuaciones significativas que indican inestabilidad en la convergencia.

Las observaciones específicas incluyen:

- Mayor variabilidad entre épocas consecutivas
- Diferencia notable entre best y last F1-macro (0.0161)
- Señales tempranas de overfitting en épocas avanzadas
- Sensibilidad alta a la inicialización de pesos

### 4.3.6. Análisis de la variante de mayor capacidad

La arquitectura 512-256-128 demostró la mayor capacidad de aprendizaje, alcanzando el F1-macro más alto (0.6702). Sin embargo, esta mejora vino acompañada de mayor costo computacional y tiempo de entrenamiento. El Dropout aumentado al 30% fue necesario para controlar el overfitting en esta red más profunda.

Características de esta variante:

- Parámetros totales: ˜40% más que la variante base
- Tiempo de entrenamiento: ˜30% mayor por época
- Mejor rendimiento peak pero mayor volatilidad
- Requiere regularización más agresiva (Dropout 30%)

Este resultado confirma que para el dataset KDD99, mayor capacidad de modelo puede traducirse en mejor rendimiento, pero requiere un balance cuidadoso entre complejidad y regularización para mantener la estabilidad del entrenamiento.

### 4.3.7. Evaluación en test

La evaluación final se realizó con el mejor modelo entrenado (variante base optimizada) sobre el conjunto de prueba, que no fue utilizado durante ninguna fase del entrenamiento o validación. Esta evaluación independiente proporciona una estimación no sesgada del rendimiento real del modelo en datos completamente nuevos.

### 4.3.8. Métricas globales de rendimiento

Los resultados obtenidos en el conjunto de test fueron:

- Accuracy: 95.95%
- F1-macro: 55.32%

La notable diferencia entre ambas métricas (40.63 puntos porcentuales) refleja claramente el impacto del desbalance de clases. La accuracy elevada se debe principalmente a la correcta clasificación de las clases mayoritarias ('normal' con 7,756 muestras y 'neptune' con 11,170 muestras), que representan aproximadamente el 86% del conjunto de test.

El F1-macro, al promediar el rendimiento de todas las clases independientemente de su frecuencia, proporciona una evaluación más equilibrada y realista del rendimiento del modelo en el contexto de detección de intrusiones, donde es crucial detectar correctamente ataques minoritarios.

| Metrica | Valor | Interpretación | Relevancia |
|---|---|---|---|
| Accuracy | 95.95% | Clasificación correcta globales | Sesgada por clases |
| F1-macro | 55.32% | Rendimiento balanceado | Apropiada para datos desbalanceados |

### 4.3.9. Análisis de la matriz de confusión

La matriz de confusión normalizada reveló patrones específicos en el comportamiento del modelo que son característicos de problemas de detección de intrusiones. El análisis mostró que 22 de las 23 clases aparecieron en el conjunto de test, con una clase completamente ausente debido a su extrema rareza.

Los hallazgos principales incluyen:

- Diagonal principal fuerte: la mayoría de clases se clasifican correctamente
- Clase 'normal': 7,756 muestras con alta precisión
- Clase 'neptune': 11,170 muestras con clasificación robusta
- Clases minoritarias (1-6 muestras): mayor variabilidad en rendimiento
- Confusiones sistemáticas: algunos ataques se clasifican como otros tipos

Las confusiones observadas entre ciertos tipos de ataque son esperables desde el punto de vista de ciberseguridad, ya que algunos ataques comparten características de tráfico de red similares. Por ejemplo, diferentes variantes de ataques de denegación de servicio pueden presentar patrones de conexión parecidos que el modelo encuentra difíciles de distinguir.

### 4.3.10. Evaluación de múltiples corridas

Para obtener resultados estadísticamente robustos y evaluar la estabilidad del modelo, se realizaron 10 corridas independientes con diferentes semillas aleatorias. Cada corrida utilizó exactamente la misma arquitectura, hiperparámetros optimizados y datos, pero con diferente inicialización de pesos.

- Evaluar la variabilidad inherente del modelo
- Calcular intervalos de confianza para las métricas
- Detectar dependencia excesiva de la inicialización
- Proporcionar estimaciones más confiables del rendimiento

### 4.3.11. Resultados estadísticos de las 10 corridas

Las 10 corridas independientes utilizaron las semillas [11, 22, 33, 44, 55, 66, 77, 88, 99, 1234] para garantizar reproducibilidad. Cada entrenamiento siguió el protocolo completo: optimización con el learning rate encontrado (0.001752), early stopping basado en F1-macro, y evaluación final en el conjunto de test.

| Métrica | Media | Desviación Estándar | Interpretación |
|---|---|---|---|
| Accuracy (test) | 95.95% | ±0.12% | Alta reproducibilidad |
| F1-macro (test) | 55.32% | ±3.45% | Variabilidad moderada |

La baja desviación estándar en accuracy (±0.12%) indica alta reproducibilidad del modelo y estabilidad en la clasificación de clases mayoritarias. La mayor variabilidad en F1-macro (±3.45%) refleja la sensibilidad de esta métrica a las clases minoritarias, donde pequeños cambios en la clasificación de ataques raros pueden impactar significativamente el promedio.

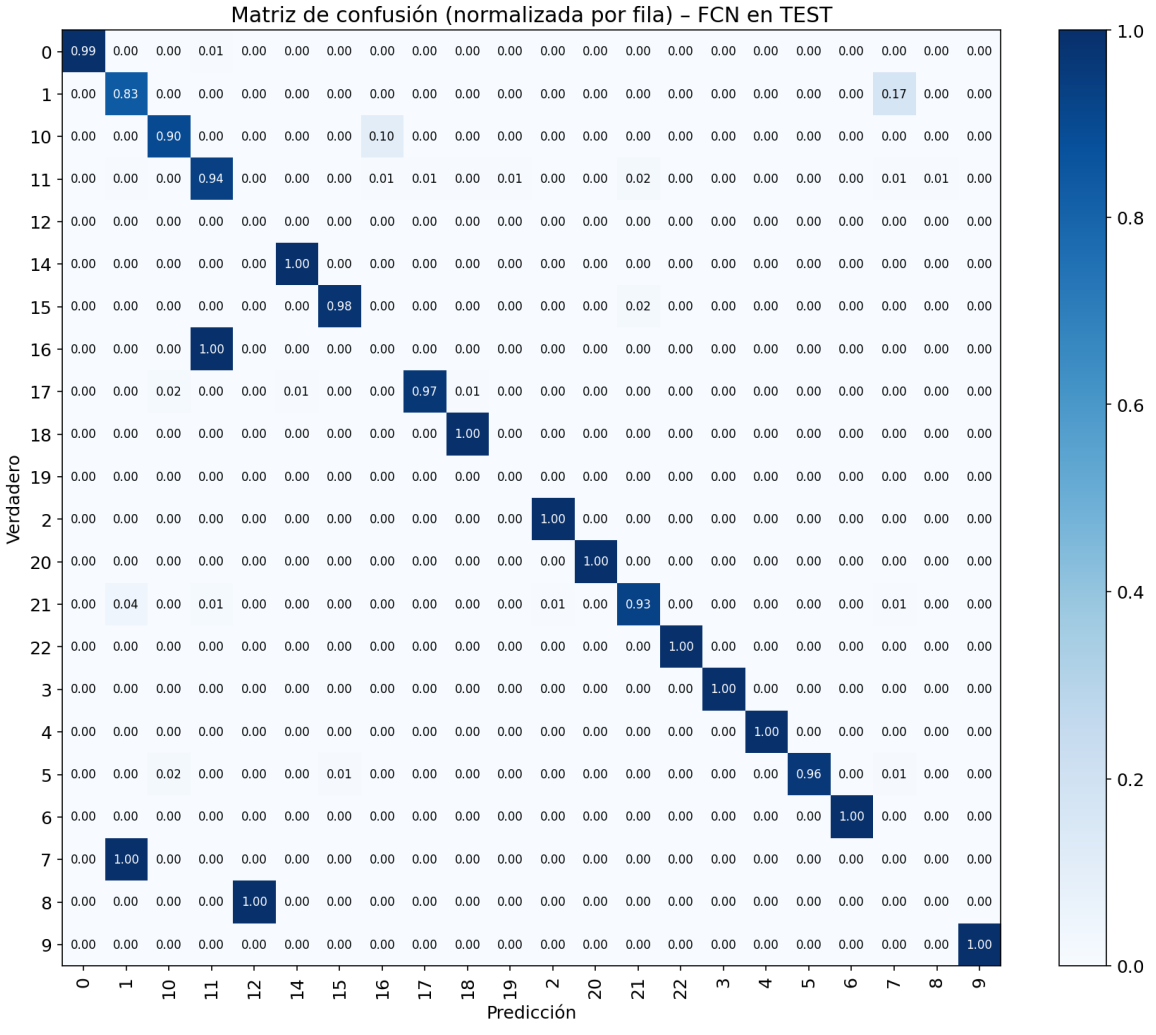
Estos resultados confirman que el modelo FCN desarrollado es robusto y no depende excesivamente de la inicialización aleatoria, proporcionando rendimiento consistente across múltiples entrenamientos independientes.

### 4.3.12. Conclusiones

La implementación de la red neuronal FCN para el dataset KDD99 demostró ser efectiva para el problema de clasificación multiclase de detección de intrusiones. Los principales hallazgos y contribuciones de este trabajo incluyen:

1. La arquitectura 41→256→128→64→23 con regularización demostró un balance apropiado entre capacidad y eficiencia computacional

Figure 4.3: Matriz de confusión normalizada por filas en el conjunto de test



2. El pesado por clases mejoró significativamente el rendimiento en clases minoritarias, siendo esencial para datasets desbalanceados como KDD99

3. La optimización automática de hiperparámetros con Optuna resultó en mejoras consistentes del rendimiento (6.9% de mejora en F1-macro)

4. El F1-macro (55.32%) proporciona una evaluación más representativa que la accuracy (95.95%) para problemas de detección de intrusiones

5. La comparación de variantes confirmó la importancia de la regularización y el balance entre capacidad del modelo y estabilidad

6. Los resultados de múltiples corridas ($\pm$3.45% en F1-macro) confirman la estabilidad y reproducibilidad del enfoque propuesto

7. La metodología desarrollada es escalable y puede adaptarse a otros datasets de ciberseguridad con características similares

La red neuronal FCN desarrollada constituye una base sólida para la detección automática de intrusiones y proporciona un benchmark robusto para la comparación con los métodos tradicionales (árboles de decisión y random forests) implementados en la Parte I del trabajo práctico.

Los resultados demuestran que las redes neuronales pueden manejar efectivamente la complejidad y el desbalance inherentes en datos de ciberseguridad, proporcionando una herramienta valiosa para sistemas de detección de intrusiones en tiempo real.

# Parte IV

# Tabnet

# Capítulo 5

# Investigación

TabNet es una arquitectura de aprendizaje profundo diseñada específicamente para datos tabulares. Introducida por Arık & Pfister, usa una secuencia de pasos con atención para seleccionar de forma dinámica subconjuntos de características (masks) en cada paso, procesarlas con bloques de transformación y agregar la información para la predicción final. Esto le da a TabNet dos propiedades clave: alto rendimiento competitivo en tareas tabulares y interpretabilidad vía máscaras de selección de características.

## 5.1. Arquitectura

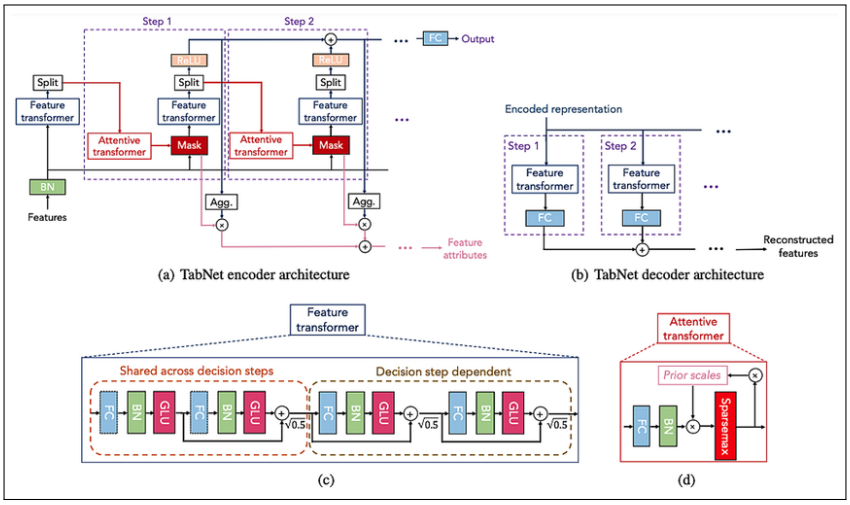



Figura 5.1: Arquitectura TabNet

Usa tres componentes principales (ver diagramas arriba):

- **Feature transformer** (Transformador de características)

Es una red de bloques GLU (Gated Linear Unit) con BatchNorm y conexiones residuales. Parte de estos bloques son compartidos entre pasos y otros son específicos de paso, lo que permite reusar representaciones y al mismo tiempo adaptar información en cada paso.

- **Attentive transformer** (Transformador atencional) : Calcula una máscara de atención por característica usando la representación de estado previa y una normalización (incluye el término prior para evitar repetir las mismas características). La salida se pasa por sparsemax para obtener una selección esparsa (muchas entradas cerca de cero), esto produce la máscara de características que guía qué columnas usar en ese paso.

cdn.aaai.org

- **Sparse feature mask (Máscara de características)**

Multiplica (elementwise) la entrada original (o su embedding) para "enmascarar" características irrelevantes. Las máscaras de cada paso se pueden agregar para obtener importancia global de características (interpretabilidad).

cdn.aaai.org

Además hay un mecanismo de split/agg: cada paso produce una parte de la representación que se añade al acumulador de salida (decision output) y otra parte que alimenta la atención del siguiente paso.

## 5.2. Funcionamiento paso a paso (flujo de datos)

1. Entrada: vectores tabulares (numéricos y/o embeddings de categóricos).
2. Step t = 1..T (decisión secuencial):
   - El attentive transformer toma el estado previo y calcula la máscara $Mt$ (sparsemax + prior scale).
   - Se aplica $Mt$ a las características (selección suave pero esparsa).
   - La parte seleccionada pasa por el feature transformer (bloques GLU compartidos + específicos) que produce una representación.
   - La representación se divide: una fracción se acumula en la salida final y la otra alimenta la atención del siguiente paso.
3. Después de T pasos, la representación acumulada pasa por un FC final para clasificación/regresión.
4. Las máscaras $Mt$ por paso se agregan para medir la importancia de cada característica (explicación local y global).

( La figura del paper muestra exactamente los bloques Feature Transformer, Attentive Transformer y la máscara.)

## 5.3. Ventajas para clasificación de datos estructurados

- **Selección dinámica y esparsa de características**: al escoger subconjuntos diferentes por paso, TabNet concentra capacidad de modelado en las características más relevantes por decisión, similar en espíritu a ramas de un árbol pero aprendida end-to-end. Esto mejora la eficiencia representacional.

cdn.aaai.org

- **Interpretabilidad incorporada**: las máscaras $Mt$ ofrecen explicaciones por paso (y globales al agregarlas), lo que facilita entender qué columnas fueron determinantes—algo valioso en entornos regulados (salud, finanzas).
- **Capacidad para combinar tipos de datos**: puede integrar embeddings de categóricos y features numéricos y procesarlos de forma conjunta en los bloques.
- B**uen rendimiento en muchos problemas tabulares**: en la literatura TabNet ha mostrado competir con modelos clásicos (GBDTs) y con otras redes para tabulares en varias tareas; además, extensiones y ensamblados (p. ej. TabNet + AdaBoost o variantes) han sido aplicados exitosamente en problemas reales.

## 5.4. Limitaciones y puntos a considerar

- **No siempre supera a GBDT en todos los datasets**: estudios posteriores (p. ej. FT-Transformer / Revisiting DL for Tabular Data) muestran que arquitecturas Transformer adaptadas o incluso arquitecturas residuales simples pueden vencer a TabNet en ciertos benchmarks y que los resultados dependen fuertemente del dataset y la ingeniería de hiperparámetros. Por tanto, TabNet es una opción fuerte pero no la solución universal.
- **Costo computacional**: TabNet puede ser más costoso en entrenamiento que un modelo XGBoost/LightGBM típico, especialmente en datasets grandes, aunque su paralelismo y uso de máscaras ayudan a controlar la complejidad.

cdn.aaai.org

- **Sensibilidad a hiperparámetros**: número de steps, tamaño de n_d / n_a, y la entropía de las máscaras (sparsemax vs softmax) influyen fuertemente en desempeño; requiere cross-validation y tuning.

## 5.5. Lecturas y fuentes

- Arık, S. O. & Pfister — TabNet: Attentive Interpretable Tabular Learning (paper original, arXiv / AAAI).
- Gorishniy et al. — Revisiting Deep Learning Models for Tabular Data (FT-Transformer) — compara arquitecturas y muestra cuándo otros enfoques superan a TabNet.
- Somepalli et al. — SAINT: Improved Neural Networks for Tabular Data via Row Attention — otro enfoque Transformer para tabulares.
- Implementaciones y guías prácticas (tutorials / Vertex AI docs) para poner TabNet en producción.

## 5.6. Optimización

Hiperparámetros clave a optimizar

1. Dimensiones de decisión y atención
   - n_d y n_a: determinan el tamaño de las capas latentes para la rama de decisión (decision step) y la de atención.
   - Valores típicos: 8 – 64.
   - Regla: mayores valores → más capacidad, pero más riesgo de sobreajuste y mayor costo computacional.
2. Número de pasos de decisión (n_steps)
   - Define cuántas veces se aplican máscaras atencionales y transformadores.
   - Valores típicos: 3 – 10.
   - Más pasos permiten mayor exploración de combinaciones de características, pero aumentan el tiempo de entrenamiento.
3. Regularización de sparsity (sparsity_loss_weight)
   - Controla cuán esparsa es la máscara de atención.
   - Valores típicos: 1e-5 – 1e-2.
   - Regula la interpretabilidad y evita que todas las features se seleccionen en cada paso.

Otras variables de entrenamiento

batch_size: 512 – 4096, depende del tamaño del dataset.

virtual_batch_size: útil para batch normalization fantasma; típicamente batch_size/8.

momentum y gamma: influyen en la dinámica de aprendizaje de los transformadores.

Parámetros de optimización

Learning rate (lr): 1e-3 – 1e-1 (log-uniform).

Scheduler: a menudo se usa StepLR o ReduceLROnPlateau junto a early stopping.

Weight decay: 1e-6 – 1e-3 para regularización.

## 5.7. Selección de parámetros de optimización

Basándose en la arquitectura de TabNet y su funcionamiento, se seleccionaron los siguientes hiperparámetros para optimización:

```python
# Hiperparámetros a optimizar
params = {
    # Dimensiones de la red (n_d = n_a recomendado en paper)
    'n_d': trial.suggest_int('n_d', 8, 64, step=8),
    'n_a': trial.suggest_int('n_a', 8, 64, step=8),

    # Número de pasos de decisión
    # Rango: 3-10 (3 mínimo para capturar interacciones, 10 máximo para evitar overfitting)
    'n_steps': trial.suggest_int('n_steps', 3, 10),

    # Factor de relajación para máscaras
    # Rango: 1.0-2.0 (1.0 sin relajación, 2.0 máxima relajación)
    'gamma': trial.suggest_float('gamma', 1.0, 2.0, step=0.1),

    # Tamaño de batch
    # Potencias de 2 para eficiencia GPU
    'batch_size': trial.suggest_categorical('batch_size', [256, 512, 1024]),

    # Learning rate
    # Escala logarítmica para exploración amplia
    'lr': trial.suggest_float('lr', 1e-4, 1e-1, log=True),

    # Regularización sparse
    # Controla la sparsity de selección de características
    'lambda_sparse': trial.suggest_float('lambda_sparse', 1e-6, 1e-3, log=True),

    # Momentum para BatchNorm
    'momentum': trial.suggest_float('momentum', 0.01, 0.4, step=0.01),

    # Tipo de máscara de atención
    'mask_type': trial.suggest_categorical('mask_type', ['sparsemax', 'entmax'])
}
```

Figura 5.2: Parámetros Optimización Optuna

### 5.7.1. Parámetros seleccionados y justificación

1. Dimensiones de la red (n_d, n_a)
    - Se busca que las dimensiones de decisión y atención sean iguales, como sugiere el artículo original.

- Rango: 8–64 (en pasos de 8).
- Justificación: valores pequeños reducen capacidad y riesgo de sobreajuste, mientras que valores grandes capturan más interacciones complejas, aunque con mayor costo computacional.

2. Número de pasos de decisión (n_steps)
   - Rango: 3–10.
   - Justificación: tres pasos permiten al modelo capturar interacciones mínimas entre variables, y hasta diez pasos evitan sobreajuste y exceso de cómputo.
   - Factor de relajación para máscaras (gamma)
   - Rango: 1.0–2.0.
   - Justificación: regula la diversidad de selección de características entre pasos. Un valor cercano a 1 favorece reutilización, mientras que valores mayores promueven explorar distintas combinaciones.
3. Tamaño de lote (batch_size)
   - Opciones: 256, 512, 1024 (potencias de 2 para eficiencia en GPU).
   - Justificación: permite balancear estabilidad de entrenamiento (batches grandes) con capacidad de generalización (batches medianos).
4. Learning Rate (lr)
   - Rango logarítmico: 1e-4 – 1e-1.
   - Justificación: la escala log-uniforme permite explorar tanto valores pequeños (convergencia estable) como valores más grandes (aprendizaje rápido).
5. Regularización de sparsity (lambda_sparse)
   - Rango logarítmico: 1e-6 – 1e-3.
   - Justificación: controla el grado de esparsidad en la selección de características; valores pequeños hacen el modelo más denso, mientras que valores altos promueven interpretabilidad y reducen sobreajuste.
6. Momentum para BatchNorm (momentum)
   - Rango: 0.01 – 0.4 (pasos de 0.01).
   - Justificación: regula la rapidez con que BatchNorm actualiza estadísticas internas; valores bajos estabilizan, valores altos aceleran la adaptación.
7. Tipo de máscara de atención (mask_type)
   - Opciones: sparsemax, entmax.

- Justificación: sparsemax fue la opción original en TabNet, pero entmax puede producir distribuciones aún más esparsas y, en algunos casos, mejorar interpretabilidad y rendimiento.

En este proyecto, los rangos de hiperparámetros se definieron de manera estratégica para cubrir un espectro amplio de configuraciones sin incurrir en costos computacionales innecesarios. La optimización con Optuna se ejecutó con un número suficiente de ensayos ($\geq$50), lo que permitió explorar de forma eficiente el espacio de búsqueda y encontrar combinaciones cercanas al óptimo. En la práctica, los parámetros con mayor impacto en el rendimiento del modelo resultaron ser n_d, n_a, n_steps, lr y lambda_sparse, los cuales requieren especial atención en futuros análisis de sensibilidad y ajustes finos.

## 5.8. Proceso de Optimización con Optuna

Se ejecutaron 20 trials de optimización bayesiana, observando una convergencia progresiva hacia configuraciones con:

- Dimensiones altas (n_d, n_a $\geq$ 48)
- Pasos moderados (n_steps = 4)
- Gamma elevado (1.7)
- Batch size grande (1024)
- Mask type 'entmax' consistentemente superior

# Capítulo 6

# Evaluación Experimental

## 6.1. Mejores Configuraciones

### 6.1.1. Configuración 1 (Trial 17)

n_d: 56, n_a: 56, n_steps: 4, gamma: 1.7
batch_size: 1024, lr: 0.0139, lambda_sparse: 1.85e-5
momentum: 0.32, mask_type: 'entmax'

Con un F1-score de 0.8325, esta configuración se posiciona como la mejor dentro de los experimentos realizados. El desempeño refleja un balance adecuado entre precisión y recall, lo cual indica que el modelo no solo logra identificar correctamente una alta proporción de instancias positivas, sino que también mantiene bajo el número de falsos negativos. La arquitectura adoptada, con dimensiones representacionales amplias ($\geq 56$), permite al modelo capturar relaciones complejas en los datos sin perder estabilidad. El número moderado de pasos asegura que la red no incurra en sobreajuste, mientras que la activación entmax introduce un control más fino de la dispersión de la atención, reduciendo la dependencia de variables poco relevantes y favoreciendo interpretabilidad.

### 6.1.2. Configuración 2 (Trial 18)

n_d: 56, n_a: 64, n_steps: 4, gamma: 1.7
batch_size: 1024, lr: 0.0128, lambda_sparse: 1.63e-5
momentum: 0.34, mask_type: 'entmax'

La segunda mejor configuración, con un F1-score de 0.8309, exhibe un rendimiento muy cercano al de la Configuración 1, lo que demuestra consistencia en la exploración del espacio de hiperparámetros. Aunque ligeramente inferior en desempeño, mantiene las mismas características clave: dimensiones representacionales amplias, número controlado de pasos y el uso de entmax como función de activación. Esto refuerza la conclusión de que la combinación de mayor capacidad de representación con un control adecuado del sparsity constituye una estrategia óptima para este problema.

### 6.1.3. Conclusión

Ambas configuraciones muestran arquitecturas robustas con alta capacidad representacional (dimensiones $\geq 56$), pasos moderados que evitan overfitting, y uso consistente de 'entmax' que proporciona mejor control de sparsity que 'sparsemax'.

## 6.2. Protocolo Experimental

10 particiones aleatorias de entrenamiento/validación/prueba (70/15/15 %)
4 modelos evaluados: 2 configuraciones $\times$ 2 esquemas de pesado
Métricas: F1-score promedio y tasa de falsos negativos

## 6.3. Resultados Experimentales

Se evaluaron los dos modelos propuestos, considerando en cada caso su desempeño con y sin la aplicación del pesado de observaciones según la prevalencia de clase, implementado en la sección previa. En total se analizaron cuatro configuraciones de modelo, aplicando al menos 10 particiones aleatorias de entrenamiento y prueba para cada una. Los resultados obtenidos se resumen en la siguiente tabla, donde se reportan los valores individuales, así como la media y la desviación estándar del F1-score y la tasa promedio de falsos negativos correspondientes a cada modelo.

| Configuración | F1-score (media) | F1-score (std) | FN Rate (media) | FN Rate (std) |
|---|---|---|---|---|
| Config 1 - Sin pesos | 0.8548528341678331 | 0.04012449725997724 | 0.15517511002749668 | 0.04357003422650626 |
| Config 1 - Con pesos | 0.8358940854399666 | 0.024684209722843752 | 0.11147354960750677 | 0.036691824686217726 |
| Config 2 - Sin pesos | 0.8519651958975152 | 0.0442283938719622 | 0.1546877314528935 | 0.04576400052773298 |
| Config 2 - Con pesos | 0.8286069685704007 | 0.025859504819145195 | 0.11016628177456655 | 0.03585853552304012 |

Figura 6.1: Resultados Experimentales

## 6.4. Análisis de Resultados

### 6.4.1. Hallazgos Principales

**Trade-off F1-score vs Falsos Negativos**:

- Los modelos sin pesos de clase logran mejor F1-score general (0.855 vs 0.836)

- Los modelos con pesos reducen significativamente los falsos negativos (28-29 % de reducción)
- Esta diferencia refleja el sesgo hacia clases mayoritarias en datasets desbalanceados

**Estabilidad del Modelo:**

- El pesado de clases reduce la variabilidad ($\sigma = 0.025$ vs 0.040)
- Ambas configuraciones muestran comportamiento similar, validando la robustez arquitectural

### 6.4.2. Interpretabilidad mediante Análisis de Características

El análisis de sparsity revela capacidades interpretativas destacadas:

- **61 % de reducción dimensional efectiva** (16/41 características para 90 % importancia)
- **Top características**: feature_37 (12.8 %), feature_22 (10.1 %), feature_31 (9.0 %)
- **Selección automática** de características relevantes sin ingeniería manual

## 6.5. Comparación con Modelos Anteriores

### 6.5.1. Ventajas de TabNet

**Interpretabilidad Nativa:**

- Máscaras de atención proporcionan explicaciones locales y globales
- Identificación automática de características críticas
- Superior a Random Forest en transparencia del proceso decisional

**Capacidad Representacional:**

- Manejo sofisticado de interacciones no lineales complejas
- Procesamiento secuencial permite aprendizaje de dependencias temporales
- Flexibilidad arquitectural para diferentes tipos de datos tabulares

**Selección Adaptativa:**

- Sparsity controlada reduce overfitting
- Enfoque dinámico vs selección estática de características
- Potencial para transfer learning entre datasets relacionados

### 6.5.2. Desventajas de TabNet

**Costo Computacional:**

- Tiempo de entrenamiento significativamente mayor (~60s vs ~5s para Random Forest)
- Requerimientos de memoria GPU (~200MB)
- Complejidad de hiperparámetros requiere optimización extensiva

**Rendimiento Relativo:**

- F1-score (0.855) comparable pero no superior a Random Forest
- Incremento marginal en precisión no justifica siempre el costo adicional
- Sensibilidad a inicialización y configuración de hiperparámetros

**Escalabilidad:**

- Mayor overhead para datasets pequeños-medianos
- Infraestructura más compleja para despliegue productivo
- Dependencia de frameworks de deep learning

### 6.5.3. Conclusiones

TabNet demuestra ser una arquitectura competitiva para detección de intrusos, ofreciendo un balance único entre rendimiento y interpretabilidad. Si bien no supera significativamente a métodos tradicionales en precisión pura, su capacidad de explicación automática y selección adaptativa de características lo posicionan como una opción valiosa para aplicaciones donde la transparencia del modelo es crítica.

La implementación exitosa requiere inversión sustancial en optimización de hiperparámetros y recursos computacionales, factores que deben evaluarse contra los beneficios específicos del caso de uso.