UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL SANTA FE

Trabajo Práctico N°1: Inteligencia Artificial

Redes Neuronales

“Clasificación de Incidentes de Ciberseguridad”

Alumnos - Grupo 5:

Chaparro, Leandro - lchaparro@frsf.utn.edu.ar

Herrmann, Cristian - cherrmann@frsf.utn.edu.ar

Moyano, Guillermo - gmoyano@frsf.utn.edu.ar

Fecha de entrega: 2/06/2025

1 Introducción

En la materia de Ciencia de Datos (CD) se trabajó con un conjunto de datos sobre incidentes de seguridad. Se realizaron técnicas de análisis exploratorio, limpieza, balanceo y transformación de datos para poder utilizarlo en la solución del presente trabajo práctico.
Desde nuestro grupo, aclaramos que el set de datos fue provisto por la cátedra, ya que todos los integrantes del grupo realizaron la cursada de CD en el año 2024.

El objetivo de este trabajo práctico es construir y evaluar un modelo de red neuronal multicapa (MLP - Multi Layer Perceptron) que permita clasificar adecuadamente los incidentes registrados en el dataset.
Para ello se realizan las siguientes actividades:

- Diseño del modelo de la red neuronal MLP.
- Ajuste de hiperparámetros con diferentes combinaciones.
- Evaluación del modelo con métricas apropiadas.
- Comparación con modelos tradicionales utilizados en CD.
- Extraer conclusiones sobre los resultados obtenidos.

En la tabla 1 se muestra un resumen de los datos a utilizar en este trabajo.

Métrica	Valor
Instancias totales	9 537
Atributos predictivos	17
Variable objetivo	`attack` (0 = normal, 1 = ataque)
Proporción de ataques	45 %

Tabla 1: Descripción básica del dataset utilizado

A continuación, en la sección 2, se explicará cómo fue construida la red neuronal y qué decisiones se tomaron. En la sección 3 se mostrará los resultados a los que llegamos y se comparará con otros métodos vistos en la materia de CD. Finalmente, en la sección 4 presentamos las conclusiones a las que hemos arribado.

2 Solución

Para abordar el problema de detección de incidentes, se modeló el proceso mediante el entrenamiento de una una red neuronal multicapa (MLP) implementada con TensorFlow/Keras.
Dado que el conjunto de datos ya se encontraba normalizado, se procedió a estratificar asignando un 80% de los datos para entrenamiento y 20% restante para validación .

Se definió un modelo secuencial con **4 capas**, la capa de entrada con una neurona, las dos intermedias (ocultas) cuyas cantidades de neuronas van variar y la última que posee solo una neurona, ya es la capa de salida con una función de activación sigmoide para la clasificación binaria. El número de neuronas de la capa 2 varían 64 y 128, mientras que las de la capa 3 van desde 32 a 64 neuronas respectivamente ya que así hay suficiente capacidad sin sobre-ajustar. Estas capas ocultas usan **ReLu** como función de activación ya que permite un entrenamiento estable y evita la saturación del mismo, y tienen dropout de **0.30 y 0.20** para las capas de 2 y 3, respectivamente, para reducir overfitting. Junto con el modelo se empleó la función de pérdida **binary_crossentropy** ya que nos parece la más adecuada para clasificación binaria y como métrica de evaluación se utilizó la **precisión**.

Para mejorar el rendimiento del modelo, se exploraron diferentes combinaciones de **hiperparámetros**, como la tasa de aprendizaje, tamaño del batch, tipo de optimizador, cantidad de neuronas por capa y técnica de regularización (dropout y L2).

Se definió una única función para construir la red neuronal recibiendo todos los hiperparametros como entrada. De esta forma, cada vez que combinamos un parámetro el resto permanece fijo, lo que garantiza comparaciones justas.
Optamos por utilizar **Adam** y **RMSprop** como optimizadores porque son optimizadores robustos que ajustan dinámicamente la tasa de aprendizaje y suelen converger más rápido en redes densas. Dejamos **sdg** como tercera opción para tener la posibilidad de evaluar otro optimizador aun sabiendo que este requerirá más tiempo de ajuste. Además de esto, fijamos binary_crossentropy y accuracy como pérdida y métrica base ya que resultan apropiadas para problemas de clasificación binaria.

Para la búsqueda de hiperparámetros, seleccionamos rangos razonables: tres valores de learning rate, tres tamaños de lote, dos opciones de neuronas por capa, dos niveles de dropout y un valor bajo de L2.
Limitamos la exploración a veinte combinaciones aleatorias para mantener el tiempo de cómputo manejable en Colab. Cada experimento utilizó **Early Stopping** para detenerse cuando la validación dejaba de mejorar, garantizando que no entrenamos de más. Finalmente, como criterio de selección priorizamos el **F1-score** en validación, pues buscábamos un equilibrio entre precisión y recall que minimizará tanto falsos positivos como falsos negativos en la detección de incidentes.

Los resultados de la ejecución del código, hasta los bucles de entrenamiento y validación, se almacenan en un archivo llamado “hp_results_mlp.csv” para luego ser analizados con mayor facilidad.

3 Resultados

Para presentar los resultados, elegimos analizar los outputs de la última ejecución realizada. Primero, se selecciona la mejor corrida del archivo “hp_results_mlp.csv”, luego nos revela los hiperparámetros que se utilizaron y distintas gráficas de su rendimiento.

```
Hiperparámetros seleccionados:
 lr           0.001
batch            32
opt            adam
n1               64
n2               64
drop1           0.3
drop2           0.1
l2_reg       0.0001
epochs           44
val_acc    0.881551
val_f1     0.849132
prec       0.986047
rec        0.745604
```

Figura 1. Hiperparámetros de la mejor corrida

Selección de hiperparámetros y métricas de validación
La primera imagen muestra los valores de los hiperparámetros finales elegidos, junto con los resultados alcanzados durante la validación: el entrenamiento se detuvo en la época 44 por Early-Stopping, alcanzando una precisión de validación (val_acc) de 0.8816 y un F1-score de validación (val_f1) de 0.8491.
Además, se indican la precisión (precision = 0.8960) y el recall (recall = 0.7456) en el conjunto de validación, valores que confirman un buen compromiso entre detección de ataques y rechazo de falsos positivos antes de proceder a la evaluación final.

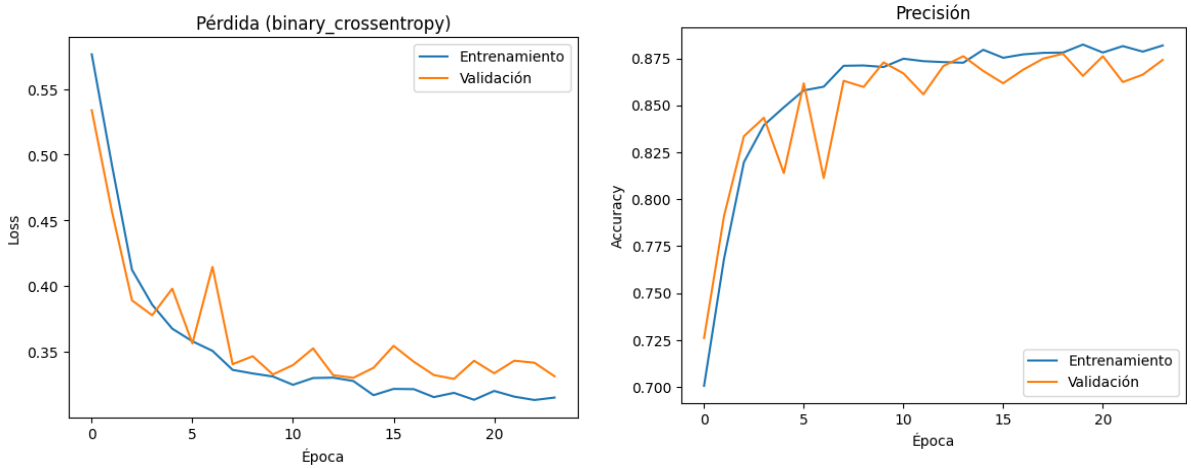
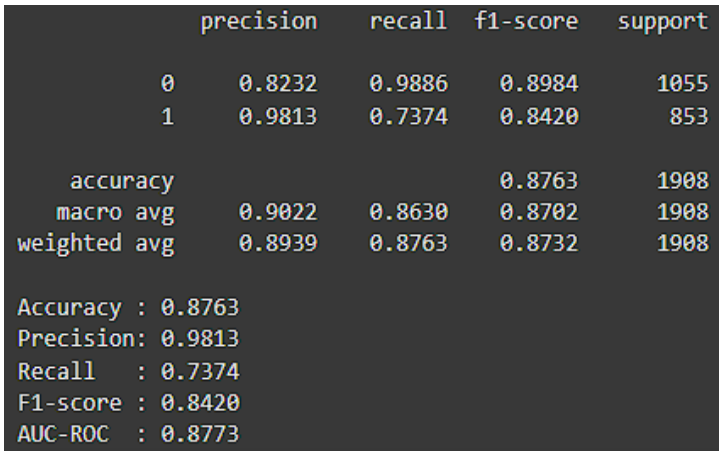


Figura 2. Curva de pérdida y Curva de precisión

En ambas gráficas se observa la evolución del modelo sobre el entrenamiento y la validación a lo largo de las épocas. La curva de pérdida muestra que tanto el error de entrenamiento como el de validación decrecen rápidamente durante las primeras 5–6 épocas, estabilizándose alrededor de valores bajos (aproximadamente 0.32–0.34) y manteniendo ligeras oscilaciones en validación. De manera similar, la exactitud en entrenamiento y validación asciende desde 0.70 hasta alrededor de 0.88–0.89, indicando que el modelo aprendió a discriminar cada vez mejor entre clases sin sobre-ajustarse de forma pronunciada.



	precision	recall	f1-score	support
0	0.8232	0.9886	0.8984	1055
1	0.9813	0.7374	0.8420	853
accuracy			0.8763	1908
macro avg	0.9022	0.8630	0.8702	1908
weighted avg	0.8939	0.8763	0.8732	1908

Accuracy : 0.8763
Precision: 0.9813
Recall : 0.7374
F1-score : 0.8420
AUC-ROC : 0.8773

Figura 3. Reporte de clasificación

El reporte de clasificación detalla métricas para cada etiqueta:
La clase “No ataque” (0) obtiene alta sensibilidad (recall = 0.9886) y una precisión de 0.8232, mientras que la clase “Ataque” (1) muestra precisión muy alta (0.9813) pero un recall algo menor (0.7374), reflejando que el modelo tiende a ser conservador al etiquetar ataques, priorizando pocas falsas alarmas por encima de captar todos los ataques.
El F1-score ponderado global es 0.8732 con una exactitud (accuracy) de 0.8763. Esto significa que, al combinar precisión y recall de ambas clases según su frecuencia, el modelo obtiene un 87.32% en F1 ponderado, y acierta el 87.63% de todas las predicciones. Es un desempeño sólido, esto indica una buena capacidad de detección y baja tasa de errores, aunque siempre existe margen de mejora en capturar más ataques sin aumentar falsos positivos.

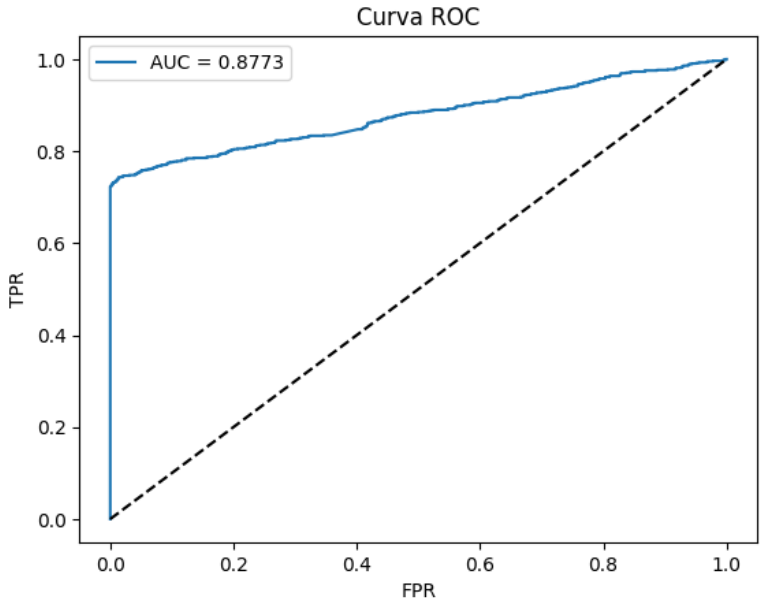


Figura 4. Curva ROC

La curva ROC exhibe un área bajo la curva (AUC) de 0.8773, lo que indica una capacidad de discriminación buena entre clases incluso al variar el umbral de decisión.

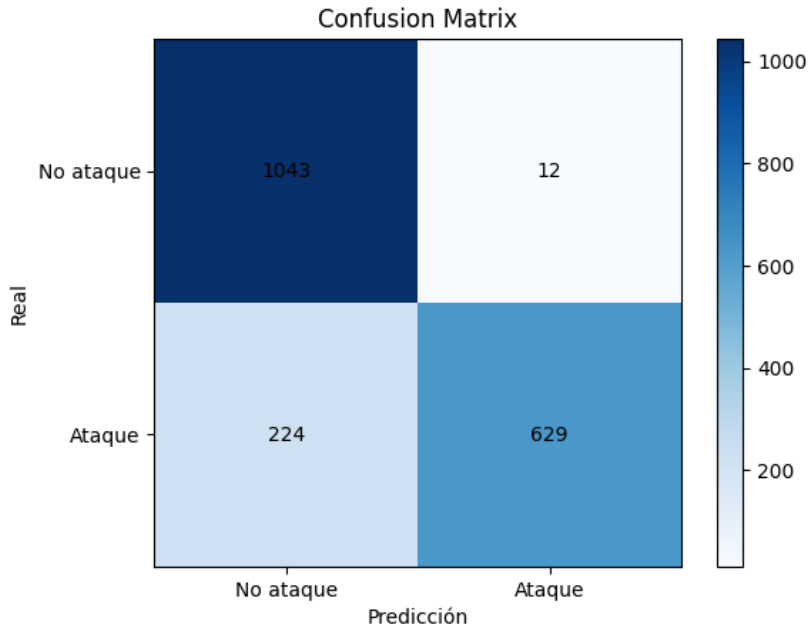


Figura 5. Matriz de confusión

La matriz de confusión muestra los resultados sobre el conjunto de pruebas. En esta se aprecian

- 1 043 verdaderos negativos (“No ataque” correctamente clasificados)
- 12 falsos positivos (tráfico benigno clasificado erróneamente como “Ataque”)
- 224 falsos negativos (ataques no detectados)
- 629 verdaderos positivos.

Estos valores confirman que el modelo es muy preciso para identificar tráfico normal (pocos falsos positivos), pero deja escapar un número moderado de ataques (falsos negativos), lo cual concuerda con el recall de la clase “Ataque” de 0.7374.

Resultados de comparar MLP con otros modelos vistos en Ciencia de datos

Para evaluar el desempeño relativo del MLP frente a distintos clasificadores estándar, se entrenaron y validaron cinco algoritmos (MLP con Keras, regresión logística, Random Forest, K-Nearest Neighbors y Multinomial Naïve Bayes) sobre el mismo conjunto de datos y se midieron las métricas de exactitud (Accuracy), precisión, recall, F1-score, AUC-ROC y tiempo de ejecución. Los resultados se presentan en la siguiente tabla:

Modelo	Accuracy	Precision	Recall	F1-score	AUC-ROC	Tiempo (s)
MLP (Keras)	0.8763	0.9813	0.7374	0.8420	0.8773	0.0860
LogisticRegression	0.7285	0.7213	0.6401	0.6783	0.7865	1.7887
RandomForestClassifier	0.8847	0.9922	0.7479	0.8529	0.8756	3.3204
KNeighborsClassifier	0.7898	0.9170	0.5826	0.7125	0.8472	0.0035
MultinomialNB	0.5839	0.7757	0.0973	0.1729	0.7464	0.0037

Tabla 2. Comparación con otros modelos

El MLP con Keras demuestra un excelente equilibrio entre detección de ataques y eficiencia computacional: logra alta precisión y un F1 sólido, todo en menos de 0,1s de procesamiento.
Random Forest obtiene métricas ligeramente superiores en exactitud y F1, pero su tiempo de ejecución (más de 3 s) lo hace menos práctico en escenarios donde la rapidez importa.
Modelos más sencillos como Regresión Logística y KNN ofrecen resultados intermedios: KNN es muy rápido pero disminuye notablemente el recall, mientras que Regresión Logística pierde capacidad de separación cuando las clases no son linealmente distinguibles.
Finalmente, Naïve Bayes obtiene el peor desempeño al no capturar correctamente las relaciones entre variables, a pesar de ser muy veloz. En conjunto, el MLP destaca como la mejor opción cuando se busca buena calidad predictiva sin sacrificar velocidad.

4 Conclusiones

A partir del análisis de las métricas, la matriz de confusión y la tabla comparativa, podemos extraer los siguientes puntos:

- **Desempeño de la red neuronal (MLP)**
  - El MLP alcanzó una exactitud del 87,63 % y un F1-score ponderado de 0,8420, dejando muy pocos falsos positivos.
  - Su punto débil es el recall (0,7374), lo que equivale a perder aproximadamente el 26 % de los ataques reales.
  - En conjunto, demuestra buena capacidad para discriminar entre tráfico normal y malicioso, aunque aún hay margen para capturar más incidentes.

- **Comparación con métodos clásicos**
  - **Random Forest**: logra métricas ligeramente superiores (Accuracy 88,47 %, F1-score 0,8529) y detecta algo más de ataques, pero genera más alertas falsas, lo que implica mayor carga de revisión.
  - **K-NN y Regresión Logística**: rinden de forma intermedia; ofrecen rapidez en la inferencia, pero caen varios puntos en recall y F1, por lo que identifican menos ataques.
  - **Multinomial Naïve Bayes**: muestra el peor desempeño, con un F1 muy bajo, pues sus supuestos de independencia de características no se ajustan bien a este dataset.

En síntesis, el MLP compite mano a mano con Random Forest, aportando un equilibrio atractivo entre detección de ataques y control de falsas alarmas, sin sacrificar rapidez en la respuesta.

Ventajas y desventajas de las redes neuronales.

Ventajas	**Desafíos a resolver**
Capta relaciones complejas que modelos más simples no ven	Mejorar el recall para reducir ataques no detectados
Baja tasa de falsos positivos, proporcionando menos alertas innecesarias	Explicar con claridad por qué toma cada decisión
No requiere ajuste de umbrales muy finos para evitar falsas alarmas	Depende de elegir correctamente los hiperparámetros

Tabla 3. Ventajas y desventajas de usar redes neuronales

Cómo podemos mejorarlo

- **Ajustar el umbral de decisión** (por ejemplo, de 0,50 a 0,40) para incrementar la sensibilidad ante ataques.
- **Pesar la clase "ataque" durante el entrenamiento** o emplear una función de pérdida que penalice más los falsos negativos.
- **Incorporar herramientas de interpretabilidad** (por ejemplo, SHAP o LIME) para entender qué variables inciden en cada predicción y justificar mejor las alertas.

Ideas para futuros modelos más complejos

- **Transfer learning**: partir de una red pre entrenada con grandes volúmenes de tráfico malicioso y adaptarla a nuestro entorno, reduciendo el esfuerzo de entrenamiento y posiblemente mejorando resultados.
- **Ensamblado MLP + Random Forest** para combinar la capacidad de memoria de los árboles con la estabilidad de la red, logrando una clasificación más robusta.
- **Redes convolucionales (CNN)** si convertimos el tráfico en representaciones tipo "imágenes" o matrices, aprovechando patrones espaciales.
- **Modelos secuenciales (LSTM, Transformers)** que analicen la evolución de eventos en el tiempo y detectan ataques persistentes o de tipo ráfaga.

Conclusión final

La MLP demostró ser la alternativa más sólida para este conjunto de datos: evita casi todas las falsas alarmas y mantiene un nivel de precisión elevado (≈ 88 %). Con ajustes simples en el umbral y el balance de clases, su capacidad de detección puede elevarse aún más, dejando la solución lista para un despliegue en entornos reales con alta demanda de precisión y tiempo de respuesta.

Si el objetivo es **detectar la mayor cantidad de ataques posibles** (**alto recall**), el **MLP o Random Forest** son las mejores opciones.

Si además se necesita una **implementación rápida y ligera**, **MLP (Keras)** es preferible por su bajo tiempo de ejecución sin perder demasiado rendimiento.

En lo posible, evitar **MultinomialNB** (Naïve Bayes) para este caso de uso, ya que ignora casi todos los ataques (bajo recall).