Detección de Malware en Android mediante Análisis de Permisos

Oswald David Gutiérrez Cortina
Ingeniería de Sistemas
Universidad de Antioquia
Medellín, Colombia
oswaldd.gutierrez@udea.edu.co

Sebastián Bedoya Restrepo
Ingeniería de Sistemas
Universidad de Antioquia
Medellín, Colombia
sebastian.bedoya@udea.edu.co

Quelix Xiomara Pérez Alzate
Ingeniería de Sistemas
Universidad de Antioquia
Medellín, Colombia
quelix.perez@udea.edu.co

1. Introducción

El sistema operativo Android es actualmente la plataforma móvil más utilizada en el mundo, con miles de millones de dispositivos activos y una amplia variedad de aplicaciones disponibles en tiendas oficiales y de terceros. Sin embargo, esta popularidad lo convierte también en un objetivo frecuente para desarrolladores de software malicioso, que buscan explotar vulnerabilidades del sistema o engañar al usuario mediante aplicaciones aparentemente legítimas.

Una de las estrategias más comunes empleadas para la identificación de malware consiste en analizar los permisos solicitados por las aplicaciones durante su instalación o ejecución. Dado que estos permisos determinan el acceso del software a recursos sensibles del dispositivo (como contactos, cámara, ubicación o mensajes), su patrón de solicitud puede revelar comportamientos sospechosos o anómalos característicos de aplicaciones maliciosas.

El conjunto de datos NATICUSdroid (Android Permissions), disponible públicamente en el repositorio UCI Machine Learning [1], recopila información sobre los permisos extraídos de más de 29.000 aplicaciones Android, tanto benignas como maliciosas, desarrolladas entre los años 2010 y 2019. Este dataset ofrece una base sólida para la aplicación de técnicas de Machine Learning (ML) orientadas a la clasificación binaria, permitiendo explorar la eficacia de diversos modelos en la detección automática de malware.

Desarrollar una solución basada en aprendizaje de máquina para este problema tiene un alto impacto práctico, pues contribuye a mejorar los mecanismos de seguridad en los ecosistemas Android, facilitando la detección temprana de amenazas y reduciendo el riesgo de ataques dirigidos a usuarios finales. Además, este tipo de estudios permiten comprender mejor la evolución del malware y sus patrones de comportamiento a lo largo del tiempo.

2. Descripción del Problema

2.1. Contexto del problema

La detección de malware en dispositivos Android representa uno de los principales desafíos en el ámbito de la seguridad informática móvil. Las técnicas tradicionales basadas en firmas —como las utilizadas por los antivirus convencionales— han perdido eficacia ante la rápida evolución de las variantes de malware y las técnicas de ofuscación que dificultan su reconocimiento.

En este escenario, los modelos de Machine Learning se presentan como una alternativa prometedora, ya que son capaces de identificar patrones complejos en grandes volúmenes de datos y generalizar comportamientos sin depender de firmas específicas. En particular, el análisis de las combinaciones de permisos solicitados por las aplicaciones resulta muy útil, pues estos reflejan tanto el nivel de acceso requerido como las posibles intenciones del desarrollador.

El objetivo de este proyecto es diseñar, entrenar y evaluar un sistema de clasificación supervisado que determine si una aplicación Android es benigna o maliciosa a partir de los permisos declarados en su archivo de manifiesto.

2.2. Composición de la base de datos

El conjunto de datos NATICUSdroid (Android Permissions) [1] fue desarrollado con el propósito de apoyar la investigación en detección de malware mediante el análisis de permisos en aplicaciones Android. Está conformado por 29.333 instancias, cada una correspondiente a una aplicación, clasificadas como benignas o

maliciosas.

Cada registro está compuesto por 86 características, donde cada una representa un permiso específico del archivo AndroidManifest.xml. Los valores están codificados de forma binaria —1 si el permiso está presente y 0 si no lo está—, permitiendo representar cada aplicación como un vector de permisos. Esta estructura facilita tanto el análisis estadístico como la aplicación de diversos algoritmos de aprendizaje automático.

Características principales del conjunto de datos:

- **Número de instancias:** 29.333
- **Número de características:** 86
- **Tipo de características:** Binarias (presencia o ausencia de permisos)
- **Tarea asociada:** Clasificación binaria (benigna / maliciosa)
- **Datos faltantes:** No presenta
- **Fuente:** UCI Machine Learning Repository

Según los autores, el dataset no contiene valores faltantes y se recomienda realizar una división 70/30 entre los conjuntos de entrenamiento y prueba o aplicar validación cruzada de 10 pliegues para obtener una evaluación robusta del modelo.

Por su naturaleza tabular y binaria, NATICUSdroid resulta especialmente adecuado para la aplicación de modelos supervisados como árboles de decisión, bosques aleatorios (Random Forest), máquinas de soporte vectorial (SVM) y redes neuronales, permitiendo comparar el rendimiento de diferentes algoritmos en términos de precisión, sensibilidad, especificidad y otras métricas de evaluación complementarias.

2.3. Paradigma a utilizar

El paradigma de aprendizaje seleccionado corresponde a un modelo supervisado de clasificación binaria, cuyo objetivo es aprender la relación entre los permisos solicitados por una aplicación y su naturaleza (benigna o maliciosa).

El enfoque se basa en el uso de algoritmos de aprendizaje supervisado que reciben como entrada el conjunto de permisos (atributos) y producen como salida una etiqueta que indica el tipo de aplicación. Se buscará evaluar diversos clasificadores con el fin de identificar el modelo que ofrezca el mejor equilibrio entre exactitud, recall y robustez ante falsos positivos.

De manera adicional, se considerará el uso de métodos de reducción de dimensionalidad (como PCA o selección de características basada en importancia) para analizar la contribución de los diferentes permisos al proceso de detección, optimizando así la interpretabilidad y eficiencia del modelo.

3. Estado del arte

En el artículo de Prasad et al. [2], utilizan un paradigma de aprendizaje supervisado, donde las aplicaciones Android se clasifican como benignas o maliciosas a partir de características extraídas del código fuente y los permisos solicitados. Los autores emplean un modelo de ensamble que combina las técnicas Random Forest, Árboles de Decisión y Bagging, con un método propio de selección óptima de características (AndroMD Optimal Feature Selection, AOFS) para mejorar la precisión y reducir la redundancia de datos. La validación se realiza mediante experimentos con tres grandes conjuntos de datos creados automáticamente (KeyCount, ZeroOne y MNF) y pruebas en entornos reales con aplicaciones legítimas y maliciosas desarrolladas por los investigadores. Las métricas usadas para evaluar el desempeño fueron precisión (accuracy), tasa de verdaderos positivos (TPR), tasa de falsos positivos (FPR) y medida F (F1-score). Los resultados muestran una precisión de hasta 99,88 % en pruebas internas y 91,66 % en entornos reales, logrando incluso detectar malware desconocido (zero-day) que otros sistemas como VirusTotal no identificaban.

Por su parte, Sahin et al. [3] también emplea un paradigma de aprendizaje supervisado, donde el sistema aprende a clasificar aplicaciones Android como benignas o maliciosas a partir de un conjunto etiquetado de datos. Utilizan diversas técnicas de aprendizaje automático, entre ellas Random Forest, SVM, Naive Bayes, KNN y redes neuronales, aplicadas tras reducir la dimensionalidad de las características mediante PCA (Análisis de Componentes Principales) y LDA (Análisis Discriminante Lineal) para optimizar el rendimiento y disminuir el tiempo de ejecución. La metodología de validación usada consiste en pruebas experimentales con diferentes conjuntos de datos de aplicaciones benignas y maliciosas, comparando el desempeño de los modelos con y sin reducción de dimensiones. Las métricas de evaluación incluyeron accuracy (exactitud), True Positive Rate (TPR), False Positive Rate (FPR) y F1-score. En los resultados citados, se reportan desempeños sobresalientes: Random Forest alcanzó hasta 97 % de exactitud con el modelo NATICUSdroid, mientras que otros estudios previos con diferentes técnicas de Machine Learning lograron entre 93 % y 98 % de precisión, confirmando la eficacia de la reducción de dimensiones para mejorar la detección de malware en Android.

El trabajo de Najibi & Bidgoly [4] emplea un paradigma de aprendizaje supervisado para detección binaria

(benign/malware) y clasificación multi-clase de malware Android. Los autores implementan tres arquitecturas de deep learning: DNN, 1D-CNN y BiLSTM, entrenadas sobre características de flujo de red del dataset CICAndMal2017. La validación se realizó mediante holdout estratificado (80 %-20 %) y evaluación en datos no vistos, separando temporalmente muestras pre-2017 (entrenamiento) de muestras 2017 (prueba) para simular generalización a malware emergente. Las métricas empleadas fueron Accuracy, Precision, Recall, F1-Score y AUC (área bajo la curva ROC, que mide la capacidad discriminativa del modelo en todos los umbrales de clasificación). Adicionalmente, utilizaron LIME y SHAP como técnicas de explicabilidad post-hoc para interpretar decisiones del modelo. Los resultados en el dataset completo mostraron que BiLSTM alcanzó el mejor desempeño en detección (F1=99.01 %, AUC=99.85 %) y 1D-CNN en clasificación (F1=94.05 %, AUC=99.41 %). En datos no vistos, BiLSTM mantuvo superioridad con F1=70.90 % (detección) y F1=62.84 % (clasificación). Tras reentrenamiento usando únicamente características robustas identificadas mediante análisis de explicabilidad, el modelo de detección preservó F1=71 % y el de clasificación F1=57 %, demostrando que la selección de características resistentes a evasión mejora la robustez con degradación mínima del rendimiento.

El trabajo de Vu Minh & Do Xuan [5] emplea un paradigma de aprendizaje supervisado para clasificación binaria de malware Android. Los autores proponen un método híbrido que combina: (1) construcción de perfiles de comportamiento mediante grafos de llamadas a funciones (FCG) enriquecidos con características estructurales (in-degree, out-degree, closeness, Katz, coeficiente de clustering) y características semánticas (embeddings de nombres de paquetes, clases y funciones usando Skip-gram/Word2vec), y (2) extracción de características mediante la red neuronal de grafos GraphSAGE, seguida de clasificación con Random Forest (RF). La validación se realizó mediante división estratificada 80 %-20 % sobre un dataset de 40,819 aplicaciones APK (50 % benignas, 50 % malware) recolectadas entre 2018-2023 desde Androzoo y Virusshare, verificadas con VirusTotal. Las métricas empleadas fueron Accuracy, Precision, Recall y F1-Score (media armónica entre precisión y recall, definida como $F_1 = 2 \times (\text{Precision} \times \text{Recall})/(\text{Precision} + \text{Recall})$). Los resultados mostraron que GraphSAGE con 2 capas + RF con 100 árboles alcanzó el mejor desempeño: Accuracy=99.03 %, Precision=98.87 %, Recall=99.19 %, F1=99.03 %. El modelo superó a métodos comparativos como DexRay (96.82 %), MalScan (97.58 %), y MalDetGCN (96.82 %), así como a otras GNNs (GCN: 98.76 %, GAT: 98.74 %). La ablación demostró que eliminar las características propuestas reduce el F1 en $\sim$14 %, validando su contribución crítica al rendimiento del modelo.

4. Entrenamiento y Evaluación de los Modelos

4.1. Configuración experimental

4.1.1. Metodología de Validación

En este trabajo se llevó a cabo una evaluación exhaustiva del desempeño de cinco modelos de clasificación dentro de un escenario de Aprendizaje Multi-Instancia, analizando su capacidad para distinguir entre aplicaciones maliciosas y benignas a partir de la información contenida en sus permisos declarados. Los modelos considerados abarcan enfoques estadísticos, métodos basados en distancias, técnicas de ensamblado y aproximaciones inspiradas en el funcionamiento neuronal, lo que permite cubrir un espectro amplio de estrategias de aprendizaje. En particular, se incluyeron Regresión Logística, K-Nearest Neighbors, Random Forest, Red Neuronal Multicapa y Máquinas de Vectores de Soporte, cada uno seleccionado por sus propiedades distintivas para la detección de patrones complejos en datos de alta dimensionalidad. Esta diversidad metodológica ofrece una base sólida para comparar el rendimiento bajo condiciones experimentales homogéneas, explorar la sensibilidad de cada modelo frente a la estructura del dataset y establecer conclusiones robustas sobre su capacidad de generalización en contextos relacionados con la seguridad en plataformas Android.

4.1.2. División de Datos

Para la partición del conjunto de datos se emplearon distintos esquemas de validación según el modelo. La Red Neuronal Multicapa se entrenó utilizando validación cruzada de 5-fold, para la Regresión Logística y el Random Forest se aplicó validación cruzada de 10-fold, mientras que KNN y SVM utilizaron una división estratificada de 70 % entrenamiento, 15 % validación y 15 % prueba.

Al utilizar estos diferentes métodos de división garantizamos una evaluación más robusta y coherente con la naturaleza de cada algoritmo. La validación cruzada permite reducir la varianza asociada a una única partición y proporciona estimaciones más estables del rendimiento, especialmente en modelos sensibles a la estructura de los datos. Por otro lado, la división tripartita en KNN y SVM facilitan un entrenamiento eficiente conservando un conjunto de prueba independiente para medir la capacidad de generalización. En conjunto, estos procedimientos aseguran que los resultados obtenidos no

dependan de una sola partición del dataset y que los modelos sean comparables bajo condiciones controladas y reproducibles.

4.1.3. Distribución de la variable de salida

El dataset NATICUSdroid utilizado en este estudio presenta una distribución balanceada de clases con 14,700 instancias de aplicaciones maliciosas (50.12 %) y 14,632 instancias de aplicaciones benignas (49.88 %), totalizando 29,332 muestras. Esta distribución prácticamente equitativa, con un ratio de desbalance de 1.00:1, garantiza que ambas categorías estén representadas de manera uniforme, lo que evita la necesidad de aplicar técnicas adicionales de corrección como oversampling o undersampling. La paridad en la presencia de cada tipo de aplicación permite que los análisis y evaluaciones derivados del dataset sean más precisos y transparentes, ya que no se ven afectados por sesgos hacia alguna de las clases. De esta forma, los patrones asociados a aplicaciones maliciosas y benignas pueden estudiarse en condiciones equilibradas y sin influencias provocadas por una desproporción en los datos.

4.1.4. Estandarización de características

Como parte del preprocesamiento realizado, se aplicó una estandarización a las características del dataset con el fin de asegurar que todas las variables —en este caso, los 86 permisos de Android— tuvieran una contribución uniforme durante las etapas de análisis. Para ello, se empleó StandardScaler de scikit-learn, que transforma cada característica para que comparta una escala comparable en lugar de mantener valores dispersos o con rangos muy diferentes.

Dado que el estudio utiliza validación cruzada estratificada, el proceso de estandarización se llevó a cabo ajustando el escalador exclusivamente con los datos de los folds de entrenamiento en cada iteración, evitando incorporar información del fold destinado a validación. Después de este ajuste, el mismo transformador se aplicó tanto a los datos de entrenamiento como a los de prueba dentro de cada ciclo.

Este procedimiento evita la fuga de información entre particiones y garantiza que la evaluación se mantenga fiel a las condiciones reales del análisis, preservando la integridad de los resultados y evitando sesgos derivados del preprocesamiento.

4.1.5. Entrenamiento

El proceso de entrenamiento se diseñó bajo un enfoque de clasificación binaria supervisada, en el cual cada instancia corresponde a una aplicación Android descrita a partir del conjunto de permisos que solicita al sistema operativo. La unidad esencial de análisis es la aplicación individual, representada como un vector de características binarias que registra la presencia o ausencia de cada permiso considerado. Esta representación estructurada permite capturar de manera directa el comportamiento declarativo de las aplicaciones y facilita la identificación de patrones distintivos entre las categorías analizadas.

Para preparar los datos antes de su uso en los distintos esquemas de aprendizaje, se implementó un proceso de estandarización destinado a normalizar las características del conjunto global. La aplicación rigurosa de este preprocesamiento contribuyó a mejorar la estabilidad numérica durante el entrenamiento, favorecer la convergencia de los métodos basados en optimización y asegurar condiciones equitativas para el análisis comparativo, reduciendo sesgos que podrían afectar la interpretación y la validez de los resultados experimentales.

4.1.6. Configuración de Hiperparámetros

Random Forest

- `n_estimators`: Define el número de árboles de decisión que componen el bosque. Un mayor número incrementa la capacidad del modelo para capturar patrones complejos mediante el ensamble de múltiples predictores débiles, aunque con un costo computacional proporcional.

- `max_depth`: Controla la profundidad máxima que puede alcanzar cada árbol individual. Árboles más profundos pueden memorizar el conjunto de entrenamiento.

- `min_samples_split`: Establece el número mínimo de muestras requeridas para dividir un nodo interno. Valores más altos actúan como mecanismo de regularización, evitando divisiones excesivamente específicas que capturen ruido en lugar de patrones generalizables.

- `min_samples_leaf`: Define el número mínimo de muestras que debe contener un nodo hoja.

- `max_features`: Determina el número de características consideradas al buscar la mejor división en cada nodo. Esta limitación introduce diversidad entre árboles al forzar diferentes perspectivas de los datos.

- `bootstrap`: Indica si se utiliza muestreo con reemplazo para construir cada árbol. Implementa el método fundamental de bagging que permite entrenar cada estimador en un subconjunto diferente de datos.

- `class_weight`: Ajusta automáticamente los pesos de las clases de forma inversamente proporcional a sus frecuencias para compensar cualquier desbalance en la distribución de aplicaciones benignas versus maliciosas.

Cuadro 1: Hiperparámetros evaluados para Random Forest

Hiperparámetro	Valores evaluados
`n_estimators`	200, 300, 400, 500
`max_depth`	20, 25, 30, None
`min_samples_split`	2, 5, 10
`min_samples_leaf`	1, 2, 4
`max_features`	'sqrt', 'log2'
`bootstrap`	True
`class_weight`	'balanced'

Regresión Logística

- `C`: Controla el balance entre ajuste a los datos de entrenamiento y simplicidad del modelo.
- `penalty`: Tipo de norma utilizada en la penalización. La regularización L2 (norma euclidiana) tiende a distribuir los pesos entre características correlacionadas en lugar de seleccionar arbitrariamente una.
- `solver`: Algoritmo de optimización empleado (Limited-memory Broyden–Fletcher–Goldfarb–Shanno). Este método garantiza convergencia estable con regularización L2.
- `max_iter`: Número máximo de iteraciones permitidas para la convergencia del algoritmo. Este límite previene ciclos infinitos en casos donde la convergencia es lenta, aunque típicamente el algoritmo converge mucho antes.
- `random_state`: Semilla aleatoria para garantizar reproducibilidad exacta de resultados en ejecuciones posteriores. Controla la inicialización de componentes estocásticos del proceso de optimización.
- `class_weight`: Ponderación de clases. Asume distribución balanceada en el conjunto de datos, tratando todas las muestras con igual importancia durante el entrenamiento.

K-Nearest Neighbors (KNN)

- `n_neighbors`: Define el número de vecinos más cercanos considerados para la clasificación. Un valor mayor suaviza las fronteras de decisión y aumenta la robustez ante ruido, mientras que valores menores permiten capturar patrones locales más específicos.

Cuadro 2: Hiperparámetros evaluados para Regresión Logística

Hiperparámetro	Valor
`C`	1.0
`penalty`	'l2'
`solver`	'lbfgs'
`max_iter`	1000
`random_state`	42
`class_weight`	None

- `weights`: Determina cómo se ponderan las contribuciones de los vecinos. La estrategia 'uniform' trata todos los vecinos por igual, mientras que 'distance' pondera inversamente proporcional a la distancia, dando mayor importancia a vecinos más cercanos.
- `scaler`: Preprocesamiento mediante StandardScaler para normalizar las características. Esencial en KNN dado que el algoritmo es sensible a la escala de las variables al calcular distancias euclidianas.
- `cv`: Validación cruzada estratificada con 5 particiones para evaluar diferentes combinaciones de hiperparámetros manteniendo la proporción de clases en cada fold.
- `scoring`: Métrica F1-score utilizada para seleccionar la mejor configuración, priorizando el balance entre precisión y recall en el proceso de búsqueda de hiperparámetros.

Cuadro 3: Hiperparámetros evaluados para K-Nearest Neighbors

Hiperparámetro	Valores evaluados
`n_neighbors`	3, 5, 7, 9
`weights`	'uniform', 'distance'
`scaler`	StandardScaler()
`cv`	StratifiedKFold (n=5)
`scoring`	F1-score
Mejores hiperparámetros:	
`n_neighbors`	7
`weights`	'distance'

Support Vector Machine (SVM)

- `C`: Parámetro de penalización que controla el compromiso entre maximizar el margen de separación y minimizar el error de clasificación. Valores menores incrementan la regularización permitiendo más errores de clasificación a cambio de un margen más amplio.

- `Kernel`: Kernel de función de base radial (Radial Basis Function) que transforma el espacio de características original a un espacio de mayor dimensionalidad donde la separación lineal es posible. Este kernel es efectivo para relaciones no lineales complejas entre permisos.
- `max_iter`: Límite de iteraciones para el algoritmo de optimización SMO (Sequential Minimal Optimization). Este valor garantiza suficiente tiempo de convergencia para encontrar el hiperplano óptimo, especialmente en configuraciones con baja regularización.
- `tol`: Criterio de convergencia que define el cambio mínimo en la función objetivo para considerar que el algoritmo ha convergido. Una tolerancia más estricta asegura mayor precisión en la solución, aunque requiere más iteraciones.
- `random_state`: Semilla aleatoria para garantizar reproducibilidad en la inicialización del algoritmo y en las particiones de validación cruzada. Este control es fundamental para comparaciones consistentes entre diferentes valores de C.
- `StratifiedKFold`: Configuración de validación cruzada estratificada. Este esquema preserva las proporciones de clases en cada fold, asegurando evaluación robusta del desempeño para cada valor de C sin sesgo por distribución desbalanceada.

Cuadro 4: Hiperparámetros evaluados para SVM

Hiperparámetro	**Valores evaluados**
`C`	0.01, 0.1, 1, 10, 100
`Kernel`	RBF
`max_iter`	5000
`tol`	1e-8
`random_state`	Valor fijo
`StratifiedKFold splits`	4

Red Neuronal

- `Arquitectura de capas`: La red implementa una estructura secuencial de cuatro capas densas con dimensiones 256, 128, 64 y 2 neuronas respectivamente.
- `Funciones de activación`: Se utiliza ReLU (Rectified Linear Unit) como función de activación no lineal en las tres primeras capas ocultas. Esta elección introduce capacidad de modelado no lineal esencial para capturar relaciones complejas entre permisos, mientras mantiene eficiencia computacional y mitiga el problema de gradientes desvanecientes.
- `Dropout`: Se aplica regularización mediante dropout con tasas de 0.4 (40 %) y 0.3 (30 %) en las dos primeras capas ocultas. Esta técnica desactiva aleatoriamente neuronas durante el entrenamiento, forzando redundancia en las representaciones aprendidas y reduciendo la dependencia en características específicas, lo cual previene el sobreajuste.
- `Learning rate`: Tasa de aprendizaje establecida en 1e-3 (0.001) para el optimizador Adam. Este valor controla la magnitud de las actualizaciones de pesos en cada iteración, balanceando velocidad de convergencia con estabilidad del entrenamiento.
- `Weight decay`: Penalización L2 con factor 1e-4 (0.0001) aplicada directamente en el optimizador Adam. Esta regularización adicional penaliza pesos de gran magnitud, promoviendo soluciones más generalizables y reduciendo la complejidad efectiva del modelo.
- `Batch size`: Tamaño de lote fijado en 64 muestras. Este hiperparámetro determina cuántas instancias se procesan simultáneamente antes de actualizar los pesos, afectando la estabilidad del gradiente estimado y la eficiencia del uso de memoria.
- `Epochs`: Número máximo de épocas establecido en 80 iteraciones completas sobre el conjunto de entrenamiento. Sin embargo, el entrenamiento está sujeto a terminación anticipada mediante early stopping.
- `Early stopping patience`: Mecanismo de parada anticipada con paciencia de 8 épocas. El entrenamiento se detiene si la pérdida en validación no mejora durante 8 épocas consecutivas, preservando los pesos correspondientes al mejor desempeño y evitando iteraciones innecesarias que podrían degradar la generalización.

Cuadro 5: Hiperparámetros evaluados para Red Neuronal

Hiperparámetro	**Valor**
`Arquitectura`	256-128-64-2
`Activación`	ReLU
`Dropout`	0.4, 0.3
`Learning rate`	0.001
`Weight decay`	0.0001
`Batch size`	64
`Epochs`	80
`Early stopping patience`	8

4.1.7. Métricas de rendimiento

Accuracy (Exactitud): Proporción de predicciones correctas sobre el total de instancias evaluadas. Mide el desempeño global del modelo sin distinguir entre tipos de error.

Precision (Precisión): De todas las aplicaciones clasificadas como malware, cuántas son realmente maliciosas. Métrica crítica para minimizar falsas alarmas que generarían desconfianza del usuario.

Recall (Sensibilidad): De todas las aplicaciones maliciosas reales, cuántas fueron detectadas correctamente. Métrica fundamental en seguridad, donde no detectar malware tiene consecuencias críticas.

F1-Score: Media armónica entre precisión y recall. Proporciona un balance único que penaliza disparidades extremas entre ambas métricas, siendo especialmente útil cuando se requiere equilibrio entre detección y confiabilidad.

ROC-AUC: Área bajo la curva ROC que evalúa la capacidad del modelo para discriminar entre clases en todos los umbrales de decisión posibles. Valores cercanos a 1 indican excelente separabilidad entre aplicaciones benignas y maliciosas.

4.2. Resultados y Análisis

La evaluación de los cinco modelos de Machine Learning reveló un rendimiento consistentemente alto en la tarea de clasificación binaria de aplicaciones Android. La Tabla 6 presenta las métricas obtenidas en el conjunto de TEST, que reflejan la capacidad de generalización de cada arquitectura ante datos no vistos durante el entrenamiento.

Los resultados muestran que todos los modelos superaron el 96 % de exactitud, evidenciando que los patrones de permisos de Android constituyen características discriminativas efectivas para la detección de malware. Random Forest y Red Neuronal Multicapa alcanzaron el mejor desempeño con accuracy superiores a 0.97, mientras que Regresión Logística y SVM obtuvieron resultados equivalentes de 0.9601. KNN mostró un desempeño intermedio con 0.9650 de exactitud.

En términos de precisión, la Red Neuronal Multicapa destacó con 0.9840, indicando que 98.4 % de las aplicaciones clasificadas como malware fueron correctas, minimizando así las falsas alarmas. Random Forest obtuvo la segunda mejor precisión con 0.9776. Por otro lado, el recall más alto correspondió a KNN (0.9680), seguido por SVM (0.9655) y Regresión Logística (0.9654), demostrando capacidad ligeramente superior para detectar aplicaciones maliciosas reales. El F1-Score, como métrica de balance, posicionó a Random Forest como el modelo más equilibrado con 0.9719.

4.2.1. Análisis de Overfitting

La comparación entre las métricas de TRAIN y TEST permite evaluar la capacidad de generalización de cada modelo. La Tabla 7 muestra las métricas en el conjunto de entrenamiento, mientras que la Tabla 8 presenta el gap de accuracy entre ambos conjuntos, revelando el grado de sobreajuste presente en cada arquitectura.

Regresión Logística exhibió el menor gap con apenas 0.08 %, indicando generalización casi perfecta gracias a la regularización L2 con C=1.0. SVM mostró un gap negativo (−0,14 %), fenómeno ocasionalmente observado en validación cruzada cuando el conjunto de test resulta marginalmente más fácil de clasificar. KNN presentó un gap de 0.70 %, situándose entre los modelos con mejor generalización. Random Forest y Red Neuronal Multicapa presentaron gaps moderados de 0.85 % y 0.77 % respectivamente, valores aceptables que no comprometen la generalización.

4.2.2. Efecto de los Hiperparámetros

Random Forest

El proceso de Grid Search evaluó 192 combinaciones de hiperparámetros mediante validación cruzada estratificada de 10 folds. La configuración óptima identificó n_estimators=500 como el número de árboles que maximizó el F1-Score sin incremento significativo del costo computacional. La profundidad máxima óptima fue max_depth=None, permitiendo a los árboles crecer hasta capturar la complejidad inherente de los datos sin restricciones artificiales.

Los parámetros de regularización min_samples_split=2 y min_samples_leaf=1 permitieron divisiones detalladas, equilibradas por el ensamble de 500 árboles que naturalmente previene el sobreajuste mediante agregación. La estrategia max_features='sqrt' introdujo diversidad entre árboles al considerar únicamente la raíz cuadrada del total de características ($\sqrt{86} \approx 9$ permisos) en cada división, fortaleciendo la capacidad de generalización del bosque. El uso de class_weight='balanced' compensó automáticamente cualquier desbalance residual en la distribución de clases.

Regresión Logística

La configuración de hiperparámetros privilegió la interpretabilidad y estabilidad. El parámetro de regularización C=1.0 proporcionó un balance óptimo entre ajuste y generalización, evidenciado por el gap prácticamente nulo entre TRAIN y TEST. La penalización L2 distribuyó la importancia entre permisos correlacionados en lugar de realizar selección agresiva de características, resultando en un modelo robusto ante variaciones en los datos.

Cuadro 6: Métricas de desempeño en conjunto de prueba (TEST)

Modelo	Accuracy	Precision	Recall	F1-Score
Random Forest	0.9720	0.9776	0.9663	0.9719
Regresión Logística	0.9601	0.9555	0.9654	0.9604
K-Nearest Neighbors	0.9650	0.9620	0.9680	0.9650
Red Neuronal Multicapa	0.9723	0.9840	0.9596	0.9716
SVM	0.9601	0.9554	0.9655	0.9604

Cuadro 7: Métricas de desempeño en conjunto de entrenamiento (TRAIN)

Modelo	Accuracy	Precision	Recall	F1-Score
Random Forest	0.9805	0.9874	0.9736	0.9804
Regresión Logística	0.9609	0.9565	0.9659	0.9612
K-Nearest Neighbors	0.9720	0.9690	0.9750	0.9720
Red Neuronal Multicapa	0.9800	0.9900	0.9700	0.9800
SVM	0.9587	0.9534	0.9649	0.9591

El solver 'lbfgs' convergió consistentemente en menos de 1000 iteraciones, confirmando la adecuación del learning rate implícito. La ausencia de ponderación de clases (class_weight=None) fue apropiada dada la distribución equilibrada del dataset, evitando sesgos artificiales en las predicciones.

K-Nearest Neighbors

El proceso de búsqueda de hiperparámetros evaluó 8 combinaciones mediante validación cruzada estratificada de 5 folds, utilizando F1-score como métrica de optimización. La configuración óptima identificó n_neighbors=7 como el número de vecinos que maximizó el balance entre sesgo y varianza del modelo. Este valor intermedio evita el sobreajuste asociado a vecindarios muy pequeños y la pérdida de información local característica de vecindarios excesivamente grandes.

La estrategia de ponderación weights='distance' resultó superior a 'uniform', permitiendo que vecinos más cercanos en el espacio de características tengan mayor influencia en la decisión de clasificación. Este esquema es particularmente efectivo en regiones donde la densidad de muestras varía, otorgando mayor relevancia a instancias genuinamente similares al patrón bajo análisis.

La división tripartita del dataset (70 % entrenamiento, 15 % validación, 15 % prueba) permitió tanto la optimización de hiperparámetros como la evaluación final en un conjunto completamente independiente. La normalización mediante StandardScaler fue fundamental para garantizar que todas las características contribuyeran equitativamente al cálculo de distancias, evitando que permisos con mayor frecuencia dominaran artificialmente las métricas de similitud.

Red Neuronal Multicapa

La arquitectura de cuatro capas (256-128-64-2) demostró capacidad suficiente para modelar las relaciones no lineales entre permisos sin incurrir en sobreajuste significativo. Las tasas de dropout de 0.4 y 0.3 en las primeras capas ocultas actuaron como regularizadores efectivos, evidenciado por el gap de solo 0.77 % entre TRAIN y TEST.

El learning rate de 1e-3 combinado con el optimizador Adam permitió convergencia estable, mientras que el weight decay de 1e-4 complementó la regularización por dropout. El mecanismo de early stopping con paciencia de 8 épocas detuvo el entrenamiento automáticamente al detectar estancamiento en la pérdida de validación, preservando los pesos correspondientes al mejor desempeño y evitando iteraciones que degradarían la generalización. El batch size de 64 muestras proporcionó estimaciones de gradiente suficientemente estables para actualización eficiente de pesos.

Support Vector Machine

La exploración de diferentes valores de C (0.01, 0.1, 1, 10, 100) mediante validación cruzada de 4 folds permitió identificar el parámetro de regularización óptimo. Valores bajos de C incrementaron la regularización privilegiando márgenes amplios, mientras que valores altos permitieron mayor ajuste a los datos de entrenamiento. La configuración óptima balanceó estos extremos, resultando en el recall más alto (0.9655) entre todos los modelos.

El kernel RBF transformó el espacio de características original a dimensionalidad superior donde la separación lineal fue posible, capturando interacciones complejas entre permisos. El límite de 5000 iteraciones y tolerancia de 1e-8 garantizaron convergencia precisa del algoritmo SMO, especialmente relevante en configuraciones con baja regularización que requieren mayor refinamien-

Cuadro 8: Brecha de generalización (GAP) en Accuracy

Modelo	GAP (Absoluto)	GAP (%)
Regresión Logística	0.0008	0.08 %
SVM	−0,0014	0.14 %
K-Nearest Neighbors	0.0070	0.70 %
Red Neuronal Multicapa	0.0077	0.77 %
Random Forest	0.0085	0.85 %

to del hiperplano óptimo.

4.2.3. Modelo Óptimo Seleccionado

Considerando el balance entre todas las métricas evaluadas, Random Forest emergió como el modelo óptimo para la tarea de detección de malware en Android. Su F1-Score de 0.9719 superó marginalmente a la Red Neuronal (0.9716) y a KNN (0.9650), mientras mantuvo mejor equilibrio entre precisión (0.9776) y recall (0.9663). Adicionalmente, la interpretabilidad inherente de Random Forest mediante importancia de características facilita el análisis forense de aplicaciones sospechosas, aspecto crítico en contextos de seguridad donde la explicabilidad de las decisiones del modelo es fundamental para la confianza de los usuarios y analistas de seguridad.

5. Interpretación del Análisis de Correlación

5.1. Significado de las Correlaciones

El coeficiente de correlación de Pearson cuantifica el grado de asociación lineal entre cada permiso y la variable objetivo (Result). Una correlación positiva alta indica que el permiso aparece con mayor frecuencia en aplicaciones maliciosas (clase 1), mientras que una correlación negativa alta señala predominancia en aplicaciones benignas (clase 0). Es fundamental destacar que correlación no implica causalidad; estos valores indican únicamente la capacidad discriminativa individual de cada característica para separar clases.

5.1.1. Correlación Positiva Fuerte ($r > 0{,}30$)

Los permisos mostrados en la Tabla 9 exhiben asociación significativa con aplicaciones maliciosas.

Este patrón revela que el malware en Android solicita sistemáticamente permisos sensibles relacionados con rastreo de ubicación, acceso a información del dispositivo, persistencia mediante ejecución automática, y capacidad de overlay sobre otras aplicaciones. Estos permisos constituyen el núcleo discriminativo del modelo y son esenciales para la detección efectiva.

Cuadro 9: Permisos con correlación positiva fuerte

Permiso	Correlación
`android.permission.READ_PHONE_STATE`	0.7243
`android.permission.RECEIVE_BOOT_COMPLETED`	0.4988
`com.android.launcher.permission.INSTALL_SHORTCUT`	0.4512
`android.permission.ACCESS_COARSE_LOCATION`	0.4312
`android.permission.ACCESS_FINE_LOCATION`	0.4084
`android.permission.GET_TASKS`	0.3889
`android.permission.SYSTEM_ALERT_WINDOW`	0.3285

5.1.2. Correlación Negativa Fuerte ($r < -0{,}18$)

Los permisos mostrados en la Tabla 10 caracterizan aplicaciones legítimas.

Cuadro 10: Permisos con correlación negativa fuerte

Permiso	Correlación
`com.google.android.c2dm.permission.RECEIVE`	−0,4861
`com.android.vending.BILLING`	−0,2548
`android.permission.READ_EXTERNAL_STORAGE`	−0,2209
`com.google...permission.READ_GSERVICES`	−0,1964

Esta asociación negativa indica que aplicaciones benignas utilizan preferentemente servicios oficiales de Google (notificaciones push, servicios de Google), sistemas de monetización transparentes (compras in-app), y acceso estándar a almacenamiento. El malware evita estos permisos para no integrarse con infraestructuras oficiales que podrían facilitar su identificación.

5.1.3. Candidatos a Eliminación

Aplicando el criterio de correlación absoluta menor a 0.05, se identifican permisos con capacidad discriminativa prácticamente nula. Estos permisos no diferencian significativamente entre clases y podrían considerarse candidatos a eliminación.

Sin embargo, como se argumentó previamente, la eliminación no se recomienda debido a: (1) posibles interacciones no lineales no capturadas por correlación simple, (2) dimensionalidad manejable del dataset, (3) rendimiento excelente con el conjunto completo, y (4) re-

gularización automática implementada por los modelos seleccionados.

Los permisos verdaderamente informativos satisfacen el criterio $|r| > 0{,}20$, abarcando aproximadamente 25-30 características del total de 86. No obstante, los permisos restantes proporcionan información contextual complementaria que mejora la robustez del modelo ante patrones sofisticados de malware.

5.2. Análisis de Reducción Dimensional

5.2.1. Extracción de Características Lineales (PCA)

Se aplicó Principal Component Analysis para reducir las 86 características originales de permisos Android. El criterio adoptado fue retener el 95 % de la varianza acumulada, un estándar establecido en la literatura que balancea óptimamente la preservación de información crítica con la eliminación de redundancia.

Los resultados muestran que PCA redujo el conjunto a 56 componentes, logrando una compresión del 34.88 % mientras explicaba el 95.02 % de la varianza original. El impacto en el desempeño fue mínimo: el F1-Score disminuyó apenas 0.49 % (de 0.9719 a 0.9670), el Accuracy cayó 0.49 %, y el ROC-AUC solo 0.20 %. La métrica más afectada fue Recall con $-0{,}81$ %, aunque mantiene un excelente valor de 0.9585.

5.2.2. Extracción de Características No Lineales (UMAP)

UMAP se empleó como técnica no lineal capaz de preservar estructura local y global. El criterio seleccionado fue raíz cuadrada del número de características. Esta función sublineal minimiza overfitting y es ampliamente utilizada en análisis exploratorio de alta dimensión. Se configuró con n_neighbors=15 para capturar estructura local y min_dist=0.1 para permitir agrupación compacta.

UMAP alcanzó una reducción extrema del 89.53 %, comprimiendo de 86 a solo 9 componentes (ratio 9.6:1). La degradación fue moderada considerando la magnitud de reducción: F1-Score disminuyó 1.46 % (de 0.9719 a 0.9577), Accuracy 1.47 %, y ROC-AUC solo 0.49 %. Precision mostró la mayor pérdida con 1.80 %, indicando ligero aumento en falsos positivos, aunque 0.9600 sigue siendo excelente. Estos resultados demuestran que UMAP captura efectivamente relaciones no lineales relevantes que PCA podría no detectar completamente.

5.2.3. Discusión Comparativa

Ambos métodos prueban ser efectivos pero con perfiles distintos. PCA ofrece reducción moderada (35 %) con pérdida mínima (¡0.5 %), manteniendo interpretabilidad mediante combinaciones lineales de permisos. UMAP logra compresión extrema (90 %) con pérdida controlada (1.5 %), ideal para recursos limitados, aunque sacrifica interpretabilidad.

5.3. Conclusiones

Este trabajo presentó un estudio exhaustivo sobre la detección de malware en aplicaciones Android mediante análisis de permisos declarados en el manifiesto. Este estudio demuestra que el análisis de permisos mediante Machine Learning constituye una aproximación efectiva, eficiente e interpretable para la detección de malware Android, alcanzando niveles de precisión comparables al estado del arte mientras mantiene simplicidad metodológica y viabilidad de despliegue en sistemas reales de protección.

Referencias

[1] Mathur, A., Podila, L. M., Kulkarni, K., Niyaz, Q., & Javaid, A. Y. (2021). NATICUSdroid: A malware detection framework for Android using native and custom permissions. *Journal of Information Security and Applications*, 58, 102696. https://doi.org/10.1016/j.jisa.2020.102696

[2] Prasad, A., Chandra, S., Alenazy, W. M., Ali, G., Shah, S., & ElAffendi, M. (2025). AndroMD: An Android malware detection framework based on source code analysis and permission scanning. *Results in Engineering*, 28, 107050. https://doi.org/10.1016/j.rineng.2025.107050

[3] Şahin, D. Ö., Kural, O. E., Akleylek, S., & Kılıç, E. (2021). Permission-based Android malware analysis by using dimension reduction with PCA and LDA. *Journal of Information Security and Applications*, 63, 102995. https://doi.org/10.1016/j.jisa.2021.102995

[4] Najibi, M., & Bidgoly, A. J. (2025). Towards a robust Android malware detection model using explainable deep learning. *Journal of Information Security and Applications*, 93, 104191. https://doi.org/10.1016/j.jisa.2025.104191

[5] Vu Minh, M., & Do Xuan, C. (2024). A novel approach for Android malware detection based on intelligent computing. *Computers, Materials & Continua*, 81(3), 4371–4396. https://doi.org/10.32604/cmc.2024.058168