Informe final del proyecto

Por:

María Paula Rojas Ortega

Juan Camilo Castañeda Ospina

Materia:

Introducción a la inteligencia artificial

Profesor:

Raúl Ramos Pollan

Universidad de Antioquia

Facultad de Ingeniería

Medellín

2023

Índice

1. Introducción

Se ha observado un aumento considerable en el uso de software maliciosos o malware debido al crecimiento de la oferta de servicios digitales. Una vez que un ordenador está infectado por malware, los delincuentes pueden perjudicar a consumidores y empresas de muchas formas, tales como corromper datos, robar o secuestrar información, y otros ciberdelitos mediados por este tipo de software. La industria del malware sigue siendo un mercado bien organizado y financiado dedicado a eludir las medidas de seguridad tradicionales. Estos softwares pueden afectar un gran número de sistemas en poco tiempo, por lo que su detección debe darse lo más rápido posible, sin embargo, usan diferentes métodos de ocultación y evasión haciendo más difícil su detección. Por lo anterior se hace necesario mejorar las técnicas de detección para que esta se dé más eficazmente (Aslan, Ö. A., & Samet, R., 2020; Aboaoja, F. A. et.al., 2022).

El objetivo es predecir la probabilidad de que una máquina Windows sea infectada por varias familias de malware, basándose en diferentes propiedades de dicha máquina. La métrica de evaluación principal para el modelo será el área bajo la curva ROC (Receiver operating characteristic) entre la probabilidad predicha y la observada ("Receiver operating characteristic", 2023). Se espera que el modelo tenga un porcentaje de acierto de al menos un 95%, ya que el error en la detección de malwares en un mayor grado puede representar millonarias pérdidas en sistemas y datos almacenados. Con esta información se desearía obtener mejores análisis de la detección del malware y sus posibles relaciones con el hardware y software particular de cada ordenador, lo que permitiría crear soluciones personalizadas para los procesadores y equipos más vulnerables.

2. Dataset

El dataset a utilizar proviene de una competencia de kaggle, en la cual se proporcionan datos de máquinas con ciertas características que tuvieron un reporte de amenaza de Windows Defender y si fueron infectadas o no. El dataset está compuesto por dos robustos conjuntos de archivos CSV (por sus siglas en inglés, comma-separated values) para calibrar el algoritmo (train.csv) y probar el modelo (test.csv).

Cada fila representa una sola máquina y las columnas tienen información sobre las características de esta. El dataset original contiene 7.853.253 entradas para el test.csv y 8.921.483 entradas para el test.csv y un total de 82 columnas, más una en train.csv, "*HasDetection",* que es con la que se calibra el modelo, pues tiene información de sí la máquina fue infectada o no. Sin embargo, para simplificar el análisis se decidió reducir el número de filas a 100.000, provenientes del archivo train.csv, ya que el test.csv no contiene el

resultado de la variable objetivo(“*HasDetection”*), por lo que no era posible obtener la métrica de evaluación.

Entre las características más relevantes de las máquinas se encuentran:

- **MachineIdentifier** – ID de la máquina.
- **EngineVersion** – Versión del motor. e.g. 1.1.12603.0
- **AppVersion** – Versión de la aplicación. e.g. 4.9.10586.0
- **IsBeta** – Si la versión es beta o no. e.g. false
- **AVProductStatesIdentifier** - ID para la configuración específica del software antivirus.
- **CountryIdentifier** - ID del país donde se encuentra la máquina.
- **CityIdentifier** - ID de la ciudad donde se encuentra la máquina.
- **GeoNameIdentifier** - ID de la región geográfica en la que se encuentra una máquina
- **Platform** - Nombre de sistema operativo.
- **Processor** - Arquitectura del sistema operativo.
- **OsBuild** - Compilación del sistema operativo.
- **OsPlatformSubRelease** - Devuelve la sub-release de la plataforma OS
- **OsBuildLab** - Laboratorio de compilación que generó el OS. Example: 9600.17630.amd64fre.winblue_r7.150109-2022
- **SkuEdition** – Edición de antivirus.
- **SmartScreen** - Indica si el SmartScreen está habilitado y de qué forma.
- **UacLuaenable** - Este atributo informa de si el tipo de usuario "administrador en modo de aprobación de administrador" está deshabilitado o habilitado.
- **Census_MDC2FormFactor** - Factor de forma. La lógica utilizada para definir el factor de forma se basa en estándares empresariales e industriales y se alinea con la forma en que la gente piensa en su dispositivo. (Ejemplos: smartphone, tableta pequeña, todo en uno, convertible...)

- **Census_ProcessorCoreCount** - Número de núcleos lógicos del procesador.

- **Census_PrimaryDiskTotalCapacity** - Cantidad de espacio en disco en el disco primario de la máquina en MB

- **Census_PrimaryDiskTypeName** - Nombre del tipo de disco - HDD or SSD

- **Census_SystemVolumeTotalCapacity** - El tamaño de la partición en la que está instalado el volumen de Sistema en MB.

- **Census_TotalPhysicalRAM** - RAM física en MB.

- **Census_InternalPrimaryDiagonalDisplaySizeInInches** - Talla de la pantalla ppal.

- **Census_PowerPlatformRoleName** - Indica el perfil de gestión de energía preferido. Este valor ayuda a identificar el factor de forma básico del dispositivo.

- **Census_OSVersion** - Versión numérica del SO Example - 10.0.10130.0

- **Census_OSArchitecture** - Arquitectura del SO. Example - amd64

- **Census_OSBranch** - Rama del SO extraída del OsVersionFull. Example - OsBranch = fbl_partner_eeap where OsVersion = 6.4.9813.0.amd64fre.fbl_partner_eeap.140810-0005

- **Census_OSBuildNumber** - Número de compilación del sistema operativo extraído de OsVersionFull. Example - OsBuildNumber = 10512 or 10240

- **Census_OSEdition** - Nombre de la edición OS (currently Windows only).

- **Census_OSInstallTypeName** -Descripción amigable de qué instalación se utilizó en la máquina. Ejemplo: Refresh.

- **Census_OSWUAutoUpdateOptionsName:** Nombre descriptivo de la configuración de actualización automática de WindowsUpdate en la máquina.

- **Census_GenuineStateName:** Nombre del estado de la licencia.

- **Census_ActivationChannel:** Clave de licencia por menor o clave de licencia por volumen para una máquina.

- **Census_IsTouchEnabled:** Indica si es un dispositivo táctil.

- **Wdft_IsGamer:** Indica si el dispositivo es usado por gamers o no.

3. Métrica

La métrica de evaluación principal para el modelo será el área bajo la curva ROC (Receiver operating characteristic) entre la probabilidad predicha y la observada. Esta curva se genera graficando la tasa de verdaderos positivos (TPR) frente a la tasa de falsos positivos (FPR) con distintos umbrales. La tasa de verdaderos positivos también se conoce como sensibilidad, memoria o probabilidad de detección, y la tasa de falsos positivos también se conoce como probabilidad de falsa alarma (“Receiver operating characteristic”, 2023).

Los cuales se calculan mediante la siguiente expresión:

sensitivity, recall, hit rate, or true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

fall-out or false positive rate (FPR)

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

(“Receiver operating characteristic”,2023).

De igual forma, se espera que el modelo aumente la inversión en antivirus y herramientas de detección de malware, y de soluciones personalizadas, tales como aplicaciones con un diseño más enfocado a la prevención de brechas que podrían permitir la entrada de malware al sistema.

4. Variable objetivo

Como se mencionó en el apartado “Dataset”, la variable objetivo es “HasDetection”, la cual indica si un malware ha sido detectado en la máquina correspondiente.

5. Exploración de datos

5.1. Descripción del dataset

Este incluye el tamaño del dataframe, los tipos de datos que contiene cada columna se encontraron 53 de tipo numérico (i*nt64 o float64*) y 30 de tipo objeto(*str*).

5.2. Descripción de datos faltantes

Debido a la gran cantidad de columnas se realizó a través de una gráfica (*Figura 1*), para por medio de esta establecer las columnas con mayor cantidad de datos faltantes.

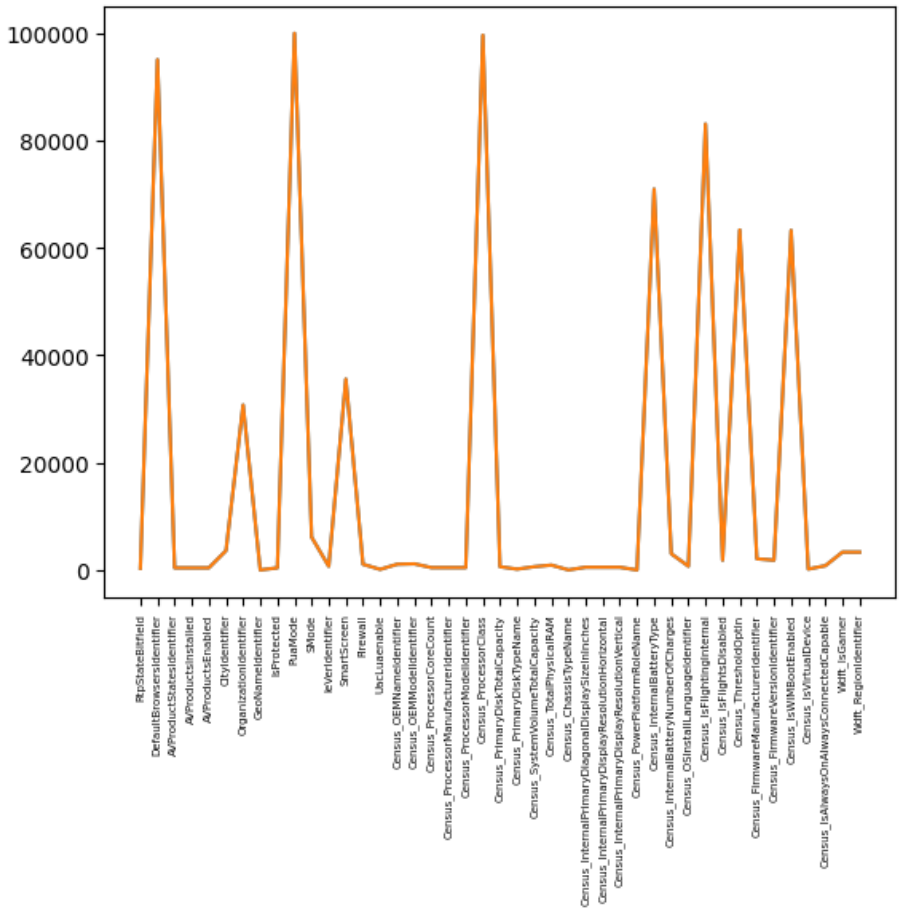

Figura 1. Datos faltantes del dataset

5.3. Distribución de la variable objetivo

Debido a que la naturaleza de la variable es binaria, la distribución de esta se expresa en términos del porcentaje de datos correspondientes a ambos valores. En cuanto al valor 1(se detecta un malware) posee un 50,18% de los datos, con respecto al 0(no se detecta un malware) corresponde a un 42,83% (*Figura 2*), por lo que se puede afirmar que tienen un porcentaje similar (teniendo en cuenta los datos faltantes).

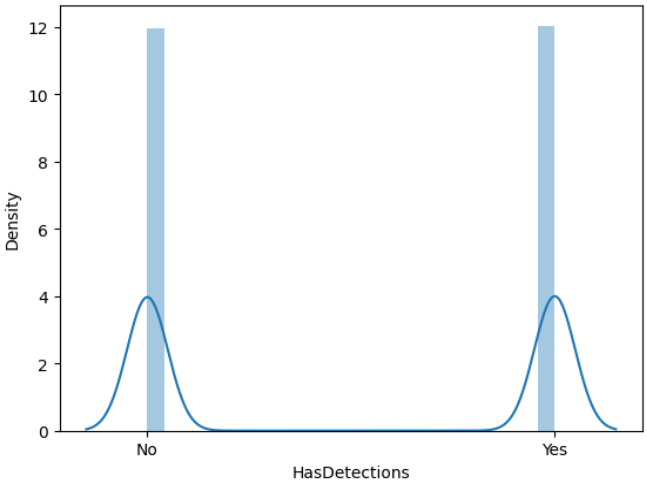

Figura 2. Distribución de la variable objetivo

5.4. Distribución de las otras variables con respecto a la variable objetivo.

Se observa que las variables “Census_ProcessorClass", "Platform", "Processor" y “Census_GenuineStateName” usadas la gráfica (*Figura 3*), para el valor “high” de la variable “Census_ProcessorClass" se observa una ligera tendencia a presentar detección de malware; en cuanto a "Platform", en el valor “windows2016” hay una ligera tendencia a no presentar detección de malware (*Figura 3, Superior*). Con respecto a la distribución de estas variables, presentan una asimetría a la izquierda muy pronunciada en "Platform", "Processor" y “Census_GenuineStateName” (*Figura 3, Inferior*).

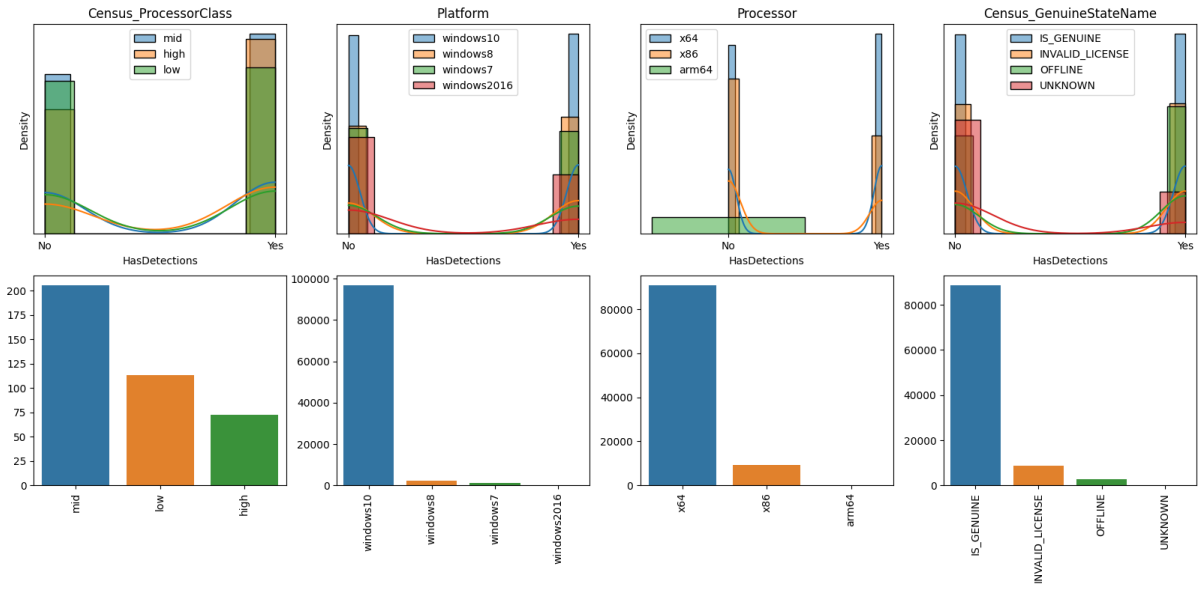

Figura 3. Superior: Distribución de la variable objetivo con respecto a otras variables. Inferior: Distribución de otras variables.

5.5. Correlación entre variables

La variable “AVProductStatesIdentifier” tiene la mayor correlación con la variable objetivo (0.116453), mientras que 'AVProductsInstalled' y 'Census_IsAlwaysOnAlwaysConnectedCapable' tienen el menor valor de correlaciones (-0.153654 y -0.062243, respectivamente) (*Figura 4*).

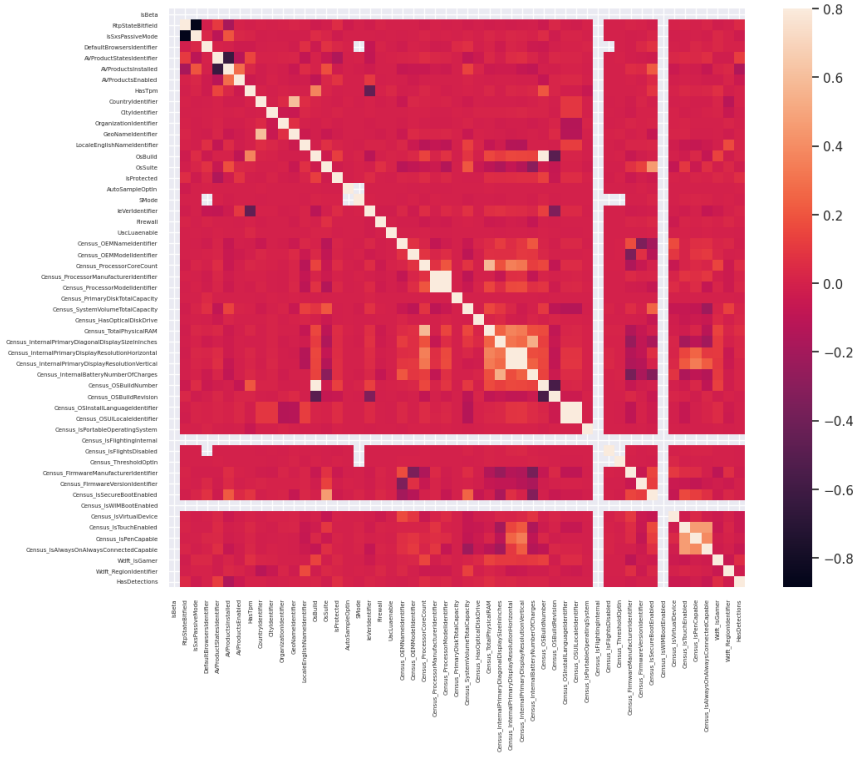

Figura 4. Correlación entre variables

6. Preprocesado

Del total de columnas se hizo una depuración bajo las siguientes condiciones:

1. No hay información sobre el contenido de la columna (Marcadas en el apartado "Data" de la competencia de Kaggle como NA, Columna no disponible o autodocumentada),
2. La cantidad de datos faltantes en la columna es mayor al 80%, y no provee información relevante con respecto a la variable objetivo.
3. Columnas que contienen un único valor para todas las filas
4. Columnas repetidas o con información similar a otras.
5. Columnas con información irrelevante

En cuanto a los datos faltantes, se encontró que en general la calidad de los datos de las 57 columnas seleccionadas, después del proceso de depuración, es muy buena, es decir, los datos faltantes representan el 1,54 %

de las entradas. Por lo que, para cumplir con el ejercicio académico, se generó el 3,5% de datos faltantes aleatoriamente.

Posteriormente, se realizan pruebas con el fin de determinar el método de llenado de datos faltantes más adecuado para los tipos de datos en el dataset, por lo anterior se realizan pruebas independientemente para las variables categóricas y las numéricas, y así mismo se establecen como se llevará a cabo la evaluación de los métodos. Para variables numéricas, la forma de rellenado de datos que más se ajustó fue el reemplazo por media, con una media de diferencia entre los promedios de los datos de 4.828584x1^-10, entre desviación estándar de 17994630.546 y coeficiente de correlación de 0.05056. Con respecto a las variables categóricas, la forma de rellenado de datos que más se ajustó fue el reemplazo por moda, con una media de diferencia entre los promedios de los datos de 0.00026, entre desviación estándar de 0.0001965 y coeficiente de correlación de 0.000698, indicando un mejor ajuste del método a los datos con respecto a los otros métodos. Por lo que se procedió a hacer One-hot encoding.

7. Selección de modelo

Para la elección del modelo se plantearon tres estimadores: Support Vector Classifier (SVC), Random Forest Classifier y Decision Tree Classifier, se seleccionaron las dos que dieron mejores resultados a través de la metodología *cross-validation* usando la métrica del área bajo la curva ROC. Los estimadores elegidos fueron Random Forest Classifier y Decision Tree Classifier con 0.6553 y 0.5526 de score respectivamente.

8. Iteraciones de desarrollo

7.1. Random Forest

Se procesaron los datos y prepararon conjuntos de entrenamiento y prueba. Se utilizó Random Forest Classifier para entrenar el modelo con los datos de entrenamiento. Se aplicó una búsqueda de hiperparámetros con *cross-validation* utilizando la función Grid Search CV para encontrar los mejores hiperparámetros del modelo. Esto permitió ajustar el modelo de manera óptima y obtener un mejor rendimiento en nuestra tarea de clasificación. Se encontró que los mejores parámetros para el modelo Random Forest Classifier son 'max_depth': 30, 'n_estimators': 100. Estos parámetros óptimos indican que utilizamos 100 árboles en el bosque y una profundidad máxima de 30 para cada árbol.

Se utilizó una matriz de confusión para evaluar el rendimiento de nuestro modelo de clasificación. La matriz de confusión obtenida tiene dimensiones

2x2 y muestra los recuentos de las diferentes combinaciones de etiquetas reales y etiquetas predichas por el modelo.

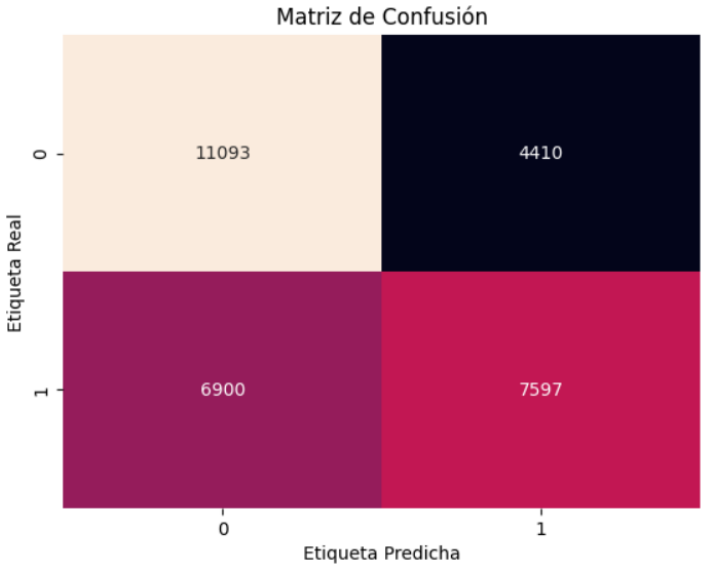

Figura 5. Matriz de confusión Random Forest

Con base en estos recuentos, se observa que el modelo tiene un mayor número de aciertos en la predicción de la clase 0 en comparación con la clase 1 (*Figura 5*). Sin embargo, el modelo también tiene un número considerable de falsos positivos y falsos negativos (*Figura 5*). Esto indica que el modelo puede tener dificultades para distinguir correctamente entre las dos clases y puede tener un sesgo hacia la clase 0.

Después de entrenar el modelo con los datos de entrenamiento, se evaluó su desempeño utilizando la curva ROC (Receiver Operating Characteristic) y el área bajo la curva (AUC-ROC).

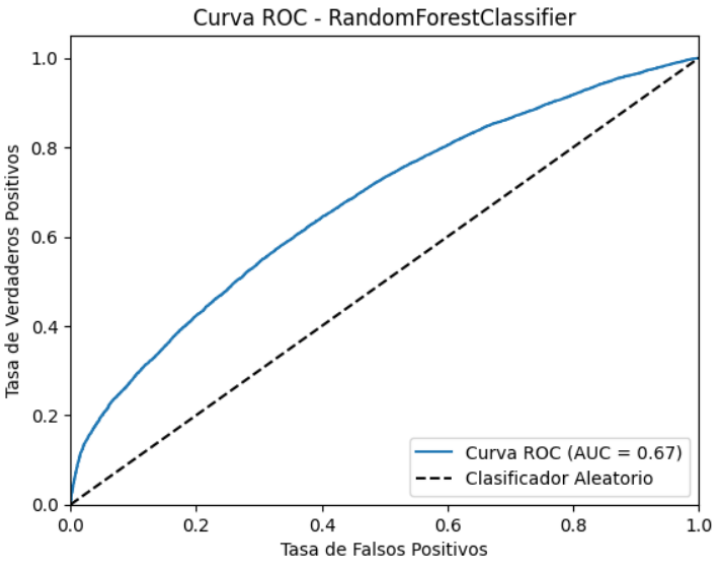

Figura 6. Curva ROC Random Forest

Se obtuvo una curva ROC con un AUC-ROC de 0.67(*Figura 6*), lo cual indica un rendimiento moderado en la clasificación. Este análisis ayuda a evaluar la precisión del modelo y brinda una medida cuantitativa de su rendimiento. Se continúan explorando otras posibles mejoras y enfoques para seguir mejorando la precisión de las predicciones.

7.1.1. Curva de aprendizaje

La implementación de la curva de aprendizaje nos permite visualizar la evolución en el rendimiento del modelo a medida que se le proporciona más información durante el proceso de entrenamiento.

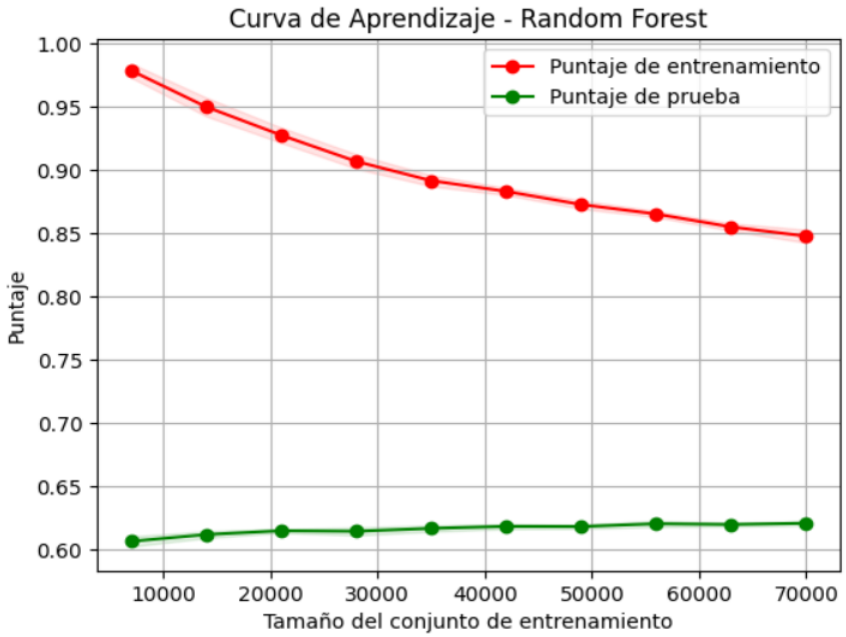

Figura 7. Curva de aprendizaje

La curva de aprendizaje que se presenta en este caso particular muestra un claro problema de Overfitting. A lo largo del entrenamiento, la precisión en el conjunto de entrenamiento comienza en un alto nivel del 97% y disminuye gradualmente hasta el 85% (*Figura 7*), lo cual indica que el modelo está memorizando los datos de entrenamiento en lugar de aprender patrones generalizables.

Por otro lado, la curva de precisión en el conjunto de prueba muestra un rendimiento bajo desde el principio, comenzando en un modesto 61% y alcanzando solo un 62.5% (*Figura 7*). Esta brecha entre las curvas de entrenamiento y prueba es una indicación clara de que el modelo no puede generalizar adecuadamente a nuevos datos.

Para abordar este problema, es recomendable tomar medidas para reducir la complejidad del modelo y aumentar la cantidad y diversidad de los datos de entrenamiento.

7.2. Random Forest + PCA

Los resultados obtenidos utilizando el Random ForestClassifier estándar fueron prometedores, con un AUC-ROC de 0.67. Esto indica que el modelo tiene una capacidad razonable para distinguir entre las clases positivas y negativas en nuestro problema de clasificación.

Se decide explorar cómo el rendimiento del modelo podría mejorarse mediante el uso de PCA. PCA es una técnica de reducción de dimensionalidad que busca capturar la mayor variabilidad posible de los datos en un número menor de componentes principales. Se aplica PCA a nuestro conjunto de datos y luego se entrena el modelo Random ForestClassifier utilizando los componentes principales obtenidos. Se realizó una validación cruzada para encontrar los mejores hiperparámetros del modelo, que resultaron ser {'pca__n_components': 4, 'rf__max_depth': 10, 'rf__n_estimators': 200}.

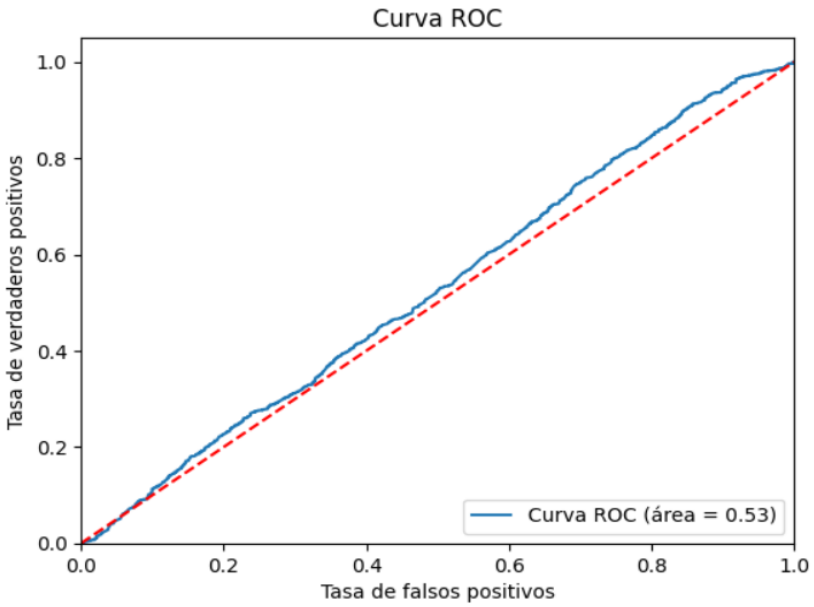

Figura 8. Curva ROC Random Forest + PCA

Los resultados obtenidos al utilizar Random ForestClassifier con PCA mostraron un AUC-ROC de 0.5357 (*Figura 8*). Aunque el rendimiento disminuyó en comparación con el modelo estándar, esto puede atribuirse a la reducción de dimensionalidad realizada por PCA, lo que pudo haber llevado a la pérdida de información importante en el proceso.

7.2.1. Curva de aprendizaje

La siguiente etapa de nuestro análisis consistió en graficar la curva de aprendizaje. Esta visualización nos permitió evaluar el rendimiento del modelo Random ForestClassifier con PCA con la cantidad de 7000 datos.

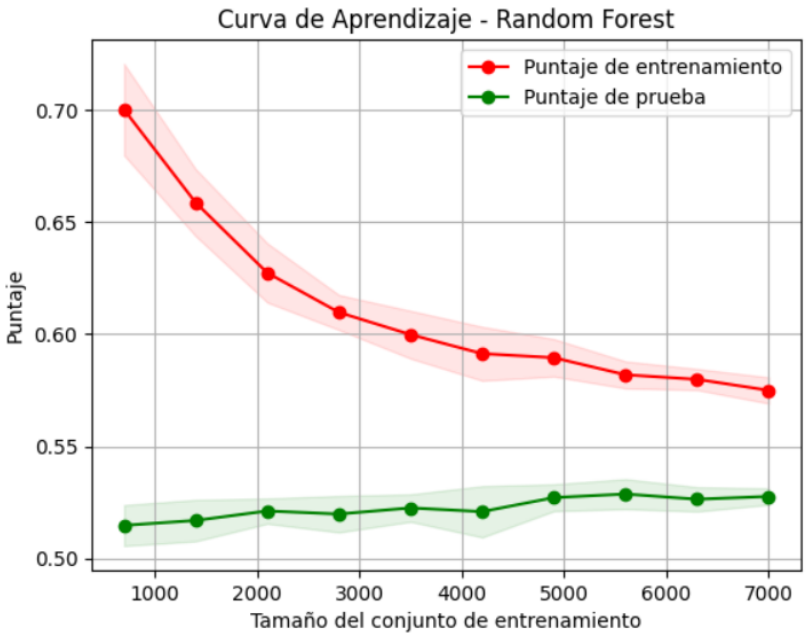

Figura 9. Curva de aprendizaje Random Forest + PCA

La curva de aprendizaje mostró que el puntaje de entrenamiento inicialmente se situaba alrededor del 70% y disminuye gradualmente a medida que aumentaba el tamaño del conjunto de entrenamiento, alcanzando aproximadamente el 57.5% (*Figura 9*). Por otro lado, el puntaje de prueba comenzaba alrededor del 52% y mostraba un ligero incremento hasta llegar al 53% (*Figura 9*)

Estos resultados indican que nuestro modelo no logra un alto rendimiento en términos de puntaje de prueba, lo cual sugiere la presencia de un desajuste en el modelo. A medida que se incrementa el tamaño del conjunto de entrenamiento, el puntaje de entrenamiento disminuye, lo que indica que el modelo tiene dificultades para ajustarse correctamente a los datos. Sin embargo, el puntaje de prueba se mantiene relativamente constante y no muestra una mejora significativa a medida que aumenta el tamaño del conjunto de entrenamiento. Esto sugiere que el modelo no está generalizando bien.

7.3. Decision Tree Classifier

Los datos fueron procesados y se crearon conjuntos de entrenamiento y prueba. Se empleó un clasificador de Decision Tree Classifier para entrenar el modelo utilizando los datos de entrenamiento. Se llevó a cabo una búsqueda de hiperparámetros mediante validación cruzada utilizando la función Grid Search CV para encontrar los hiperparámetros óptimos del modelo. Esto permitió ajustar el modelo de manera eficiente y lograr un mejor desempeño en nuestra tarea de clasificación. Se determinó que los parámetros óptimos para el clasificador de Decision Tree Classifier son 'criterion': 'entropy', 'max_depth': 10 y 'min_samples_split': 5.

Empleamos una matriz de confusión para analizar el desempeño del modelo de clasificación.

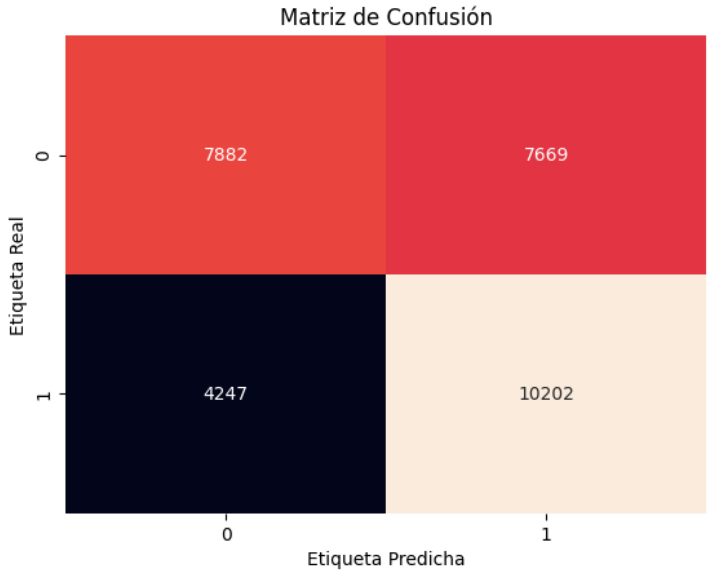

Figura 10. Matriz de confusión Decision Tree Classifier

Al analizar esta matriz, se puede observar que el modelo presenta una mayor cantidad de aciertos al predecir la clase 1 en comparación con la clase 0 (*Figura 10*). No obstante, también se identifican un número significativo de falsos positivos y falsos negativos (*Figura 10*). Esto sugiere que el modelo puede enfrentar dificultades al distinguir correctamente entre las dos clases y podría mostrar cierto sesgo hacia la clase 1.

Tras el entrenamiento del modelo con los datos de entrenamiento, se procedió a evaluar su rendimiento mediante el análisis de la curva ROC (Receiver Operating Characteristic) y el cálculo del área bajo la curva (AUC-ROC) (*Figura 11*).

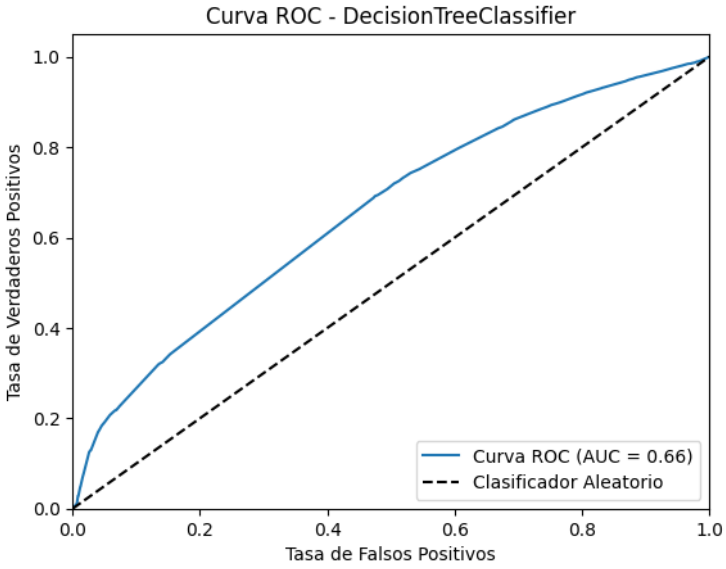

Figura 11. Curva ROC Decision Tree Classifier

Luego de realizar el análisis de la curva ROC, se obtuvo una curva que muestra un área bajo la curva (AUC-ROC) de 0.666, lo que indica un desempeño moderado en la tarea de clasificación. Esto significa que el modelo tiene cierta capacidad para distinguir entre las clases, pero existe margen de mejora para lograr una clasificación más precisa y confiable.

7.3.1. Curva de aprendizaje

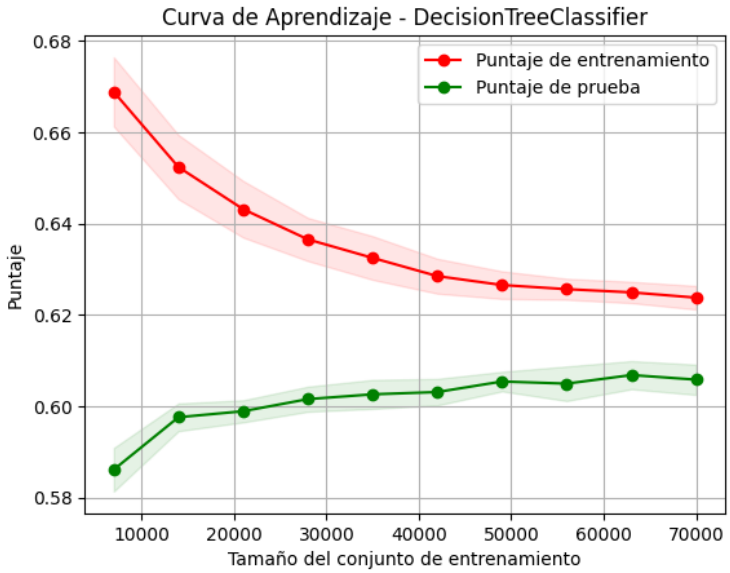

Figura 12. Curva de aprendizaje Decision Tree Classifier

La curva de aprendizaje analizada muestra que el modelo experimenta un ligero sobre ajuste, pero también muestra una mejora gradual en su rendimiento a medida que se le presenta más información de prueba (*Figura 12*). Abordar el sobre ajuste requerirá ajustar la complejidad del modelo, recopilar más datos o utilizar técnicas como la regularización y el

ensamblado. Estas acciones pueden ayudar al modelo a lograr una mejor generalización y a evitar que se adapte en exceso a los datos de entrenamiento.

7.4. Decision Tree Classifier + NMF

Ya que el clasificador NMF no admite valores negativos, se eliminaron dos filas que contenían este tipo de valores para poder ejecutar el clasificador. Se crearon conjuntos de entrenamiento y prueba, sin las filas con valores negativos. Se empleó un pipeline con clasificador NMF y Decision Tree Classifier para entrenar el modelo. Se llevó a cabo una búsqueda de hiperparámetros mediante validación cruzada utilizando la función Grid Search CV para encontrar los hiperparámetros óptimos del modelo, sin embargo, debido a la gran cantidad de datos y la limitación para iteraciones del NMF, se hizo con 10.000 filas. Esto permitió ajustar el modelo de manera eficiente y lograr un mejor desempeño en nuestra tarea de clasificación. Se determinó que los parámetros óptimos para el pipeline son, para el NMF, n_components: 2, y para el Decission Tree, max_depth': 5.

Tras el entrenamiento del modelo con los datos de entrenamiento, se procedió a evaluar su rendimiento mediante el análisis de la curva ROC (Receiver Operating Characteristic) y el cálculo del área bajo la curva (AUC-ROC) (*Figura 13*).

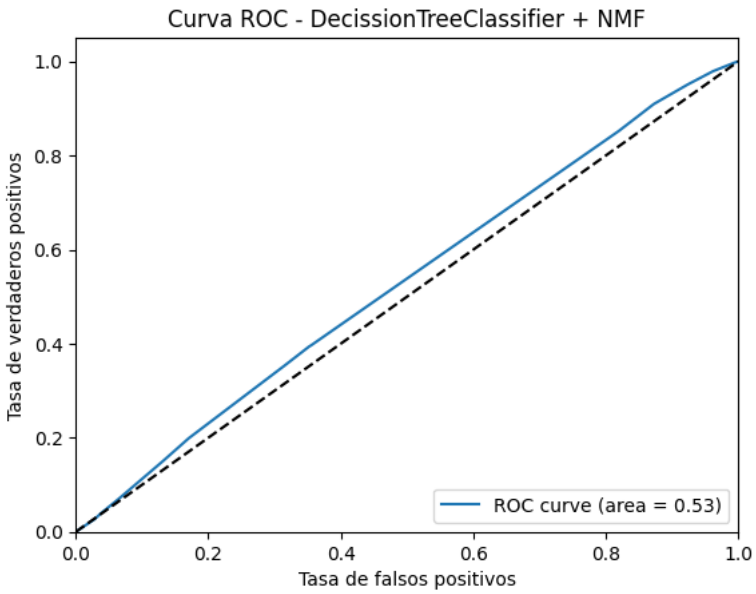

Figura 13. Curva de ROC Decision Tree Classifier + NMF

Este valor del área bajo la curva ROC (*Figura 13*) indica que el desempeño del modelo en la tarea de clasificación se puede considerar como justo, lo que implica que aún hay margen de mejora sustancial. El resultado sugiere que el modelo actual tiene una capacidad limitada para diferenciar de manera

precisa y confiable entre las distintas clases del problema en cuestión. En consecuencia, se hace necesario implementar mejoras significativas para lograr una clasificación más acertada y confiable en futuros análisis.

7.4.1. Curva de aprendizaje

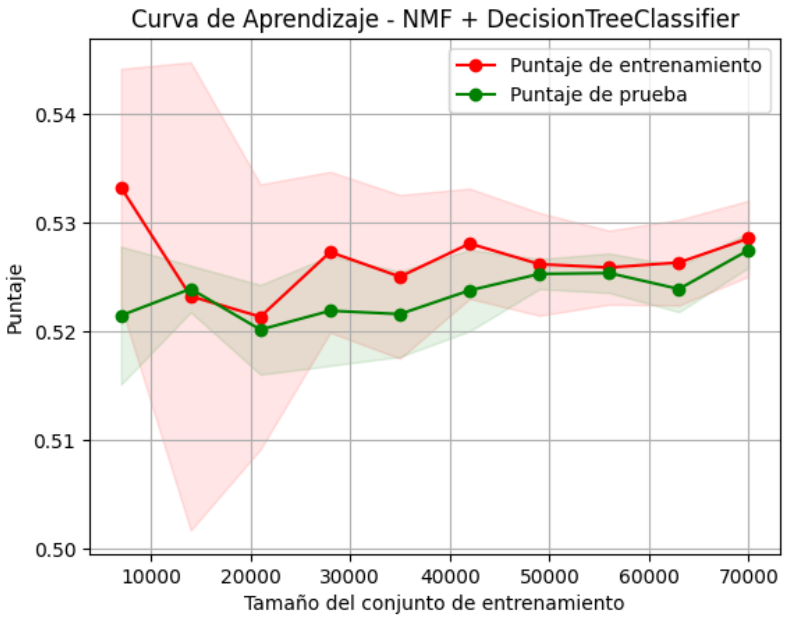

Figura 14. Curva de ROC Decision Tree Classifier + NMF

Se puede observar que el modelo presenta underfitting, lo que indica un alto sesgo (high bias), es decir, el modelo es demasiado simple o no tiene la capacidad suficiente para capturar la complejidad de los datos. Como resultado, tanto el conjunto de entrenamiento como el conjunto de prueba tienen puntajes de rendimiento relativamente bajos.

Las causas más comunes del underfitting incluyen un modelo demasiado simple y datos que están naturalmente mezclados, lo que dificulta la tarea de encontrar una relación precisa entre las características y las etiquetas objetivo. Por lo tanto, se deben considerar soluciones como aumentar la complejidad del modelo y obtener más características relevantes para mejorar el rendimiento predictivo.

9. Retos y consideraciones de despliegue

En un inicio se consideró utilizar el dataset completo para el ejercicio (train.csv), pero debido a la gran cantidad de datos no fue posible procesarlos, se rebasaba la capacidad de RAM disponible, y en cada paso se redujo la cantidad de estos hasta llegar 100.000 filas.

Por lo anteriormente mencionado, si se requiere aplicar estos modelos en el dataset completo (o en una cantidad mayor de datos) se necesita una RAM

superior a 12.7 GB (límite disponible de RAM en Google Colab), una capacidad de almacenamiento superior a 37.58 GB aproximadamente, considerando que el dataset original posee 4.3 GB y que el proceso de one-hot encoding hace más pesado el dataset. Así mismo, hay partes del modelo que tardan 2 horas en ejecutar (con 100.000 filas) por lo que hay que considerar la disponibilidad de RAM, para desplegarse en producción.

Este código se puede añadir como herramienta adicional al antivirus Microsoft Windows Defender, para analizar las posibles características del computador que puedan generar brechas por donde se posibilite la entrada de un malware.

10. Conclusiones

- La elección entre los modelos supervisados y la unión entre modelos supervisados y no supervisados depende del equilibrio adecuado entre precisión y la eficiencia computacional en cada contexto específico del problema. En este caso, los modelos supervisados generaron mejores resultados en cuanto a la métrica establecida.
- La mayoría de los modelos presentan overfitting por lo que es necesario disminuir la complejidad de estos o aumentar la cantidad de datos.
- La disponibilidad de RAM limita la reproducibilidad de los modelos y su tiempo de ejecución.

11. Bibliografía

Aslan, Ö. A., & Samet, R. (2020). A comprehensive review on malware detection approaches. IEEE Access, 8, 6249-6271.

Aboaoja, F. A., Zainal, A., Ghaleb, F. A., Al-rimy, B. A. S., Eisa, T. A. E., & Elnour, A. A. H. (2022). Malware detection issues, challenges, and future directions: A survey. Applied Sciences, 12(17), 8482.

Microsoft Malware Prediction | Kaggle. (2023). Retrieved 2 March 2023, from https://www.kaggle.com/competitions/microsoft-malware-prediction/overview

Receiver operating characteristic. (2023). Retrieved March 2, 2023, from https://en.wikipedia.org/wiki/Receiver_operating_characteristic