Técnicas avanzadas de análisis de datos

Actividad 2:

Análisis de datos topológicos para la detección de ransomware en la Blockchain de Bitcoin

Máster en Gestión y Análisis de Grandes Volúmenes de Datos:

Big Data

11/03/2022

Grupo 14

Pablo Manuel Márquez Rivero – pmmr98@hotmail.com

Marina Jiménez Cómez - mjimenez27002@alumnos.uemc.es

Antonio José Aroca Aguilar - ajaroca26969@alumnos.uemc.es

Cristina Varas Menadas - Cvaras26427@alumnos.uemc.es

Miguel Angel Casimiro Artés - miguelangelcasimiroartes@gmail.com

1. Introducción

La proliferación de criptodivisas como Bitcoin, las cuales permiten transacciones "*anónimas*" o "*pseudo-anónimas*" entre pares, ha permitido que los desarrolladores de ransomware ganen tracción, encriptando datos sensibles de los usuarios y solicitando pagos en Bitcoin para su rescate. Como es normal, con el auge de las criptodivisas y el aumento de los desarrolladores de ransomware, se han realizado bastante esfuerzos para analizar las transacciones de criptomonedas utilizando siempre ciertas heurísticas, como por ejemplo, "co-spending", que se basa en la idea de que las direcciones de entrada a la misma transacción deben pertenecer a la misma persona. Sin embargo, podrían conseguirse mejores resultados o detectar mayor cantidad de casos de ransomware si se aprovechan las herramientas avanzadas de la ciencia de datos, como pueden ser por ejemplo la creación de modelos de predicción.

En la presente actividad se han desarrollado diferentes modelos de clasificación/predicción utilizando las herramientas estudiadas a lo largo de la asignatura que permiten predecir si una determinada transacción de Bitcoin está relacionada o no con un ransomware. El motivo del uso de estos modelos es que el software convencional de seguridad informática requiere de un esfuerzo humano importante para identificar vulnerabilidades, a través de un proceso que permita encontrar sus características y luego desarrollar la solución sobre la herramienta. Esta es una labor que puede ser más eficiente si se aplican algoritmos de Machine Learning. En base a trabajos previos, las principales ventajas de contar con un sistema de seguridad con Machine Learning son las siguientes: detección proactiva de ataques, análisis automático de amenazas, y descubrimiento asistido de vulnerabilidades. De esta forma, en este trabajo se va a llevar a cabo un proceso consistente en buscar, entre varias combinaciones de algoritmos, el modelo de aprendizaje automático ideal para predecir un secuestro de datos (ransomware) en una transacción de Bitcoin.

En esta actividad, como es de esperar, trabajaremos con una serie de datos los cuales nos aportarán información sobre las transacciones. Para atender a la pregunta de si una transacción se relaciona con un ransomware o no, lo haremos de dos formas diferentes:

- Mediante clasificación binaria: Se puede analizar si una transacción de Bitcoin está relacionada o no con un ransomware, independientemente del tipo de ransomware. Etiquetamos como 0 al tipo de ransomware "White" (es decir que no tiene ransomware) y 1 al resto de Ransomware, independientemente del tipo.

- Mediante clasificación multiclase: Se clasifica por tipo de Ransomware (pricentonCerber, pricetonLocky, montrealAPI...) incluyendo el tipo "White" que señala que no hay Ransomware. Esta opción se ha probado en desarrollo, lo que ha

llevado a elevados tiempos de carga y peores resultados que para los modelos de clasificación binaria, por lo que en el apartado de Resultados solo se muestran los de esta última.

2. Materiales y métodos

2.1. Materiales

El material de datos utilizado consiste en un dataset que contiene las siguientes características:

- Address: dirección del destinatario de la transacción.
- Year: año de la transacción.
- Day: día del año de la transacción.
- Length: conteo de las rondas mixtas de Bitcoin.
- Weight: cuantificador del comportamiento de fusión (más direcciones de entrada que de salida).
- Count: similar a la anterior, aunque esta es un conteo con información sobre el número de transacciones.
- Looped: conteo de cuántas transacciones dividen sus monedas, moviéndolas en la red y fusionándolas en una sola dirección.
- Neighbors: número de vecinos que tuvo la transacción.
- Income: ingresos en términos de cantidad de Satoshi (unidad más pequeña de un bitcoin).
- Label: tipo de Ramsomware.

Además se ha modificado la característica “label” y se han añadido dos nuevas columnas, las cuales serán las variables a predecir en nuestros modelos:

- “Clase”: 0 representa que no hay Ransomware y 1 que hay algún tipo de Ransomware.
- “Label_encoded”: Contiene una codificación de tipo Hot Encoding de los distintos tipos de Ransomware. Con el tipo numérico, el algoritmo puede interpretar los datos.

Estos datos serán analizados desde un punto estadístico, donde estudiaremos la correlación de las variables y su distribución, y se utilizarán técnicas de filtrados de variables para ver qué características aportan información relevante al problema de clasificación planteado.

2.2. Métodos y algoritmos

- **Regresión logística binaria**

  Es un análisis predictivo. Se utiliza para describir datos y explicar la relación entre una variable binaria dependiente y una o más variables independientes. Se utiliza para predecir la probabilidad de pertenencia a la clase 1. Es ampliamente utilizado por su eficiencia y pocos recursos requeridos en comparación con otros algoritmos más complejos. Sin embargo, requiere tamaños muestrales bastante grandes en general.

- **Decision Tree (clasificación binaria)**

  Es uno de los algoritmos de aprendizaje supervisado más utilizados en Machine Learning. Son representaciones gráficas de posibles soluciones a un problema determinado basándose en ciertas condiciones. Los árboles de decisión tienen un primer nodo llamado raíz y nodos, los cuales se bifurcan y subdividen hasta llegar a las hojas, que son los nodos finales y equivalen a respuestas a la solución (en nuestro caso 0 o 1).

- **Aprendizaje conjunto**

  El algoritmo de Aprendizaje conjunto se basa en un grupo de modelos base (normalmente árboles de decisión) que trabajan de manera colectiva para mejorar la predicción final. Estos modelos pueden no funcionar bien de forma independiente debido a la alta variabilidad o al alto sesgo. Pero cuando se agregan, se convierten en un algoritmo fuerte capaz de reducir la varianza y el sesgo, y por consiguiente alcanzar un mejor desempeño. Hay dos tipos de aprendizaje conjuntos:

  - **Bagging**: entrenamiento de modelos débiles en paralelo. Se utiliza cuando existe alta varianza y bajo sesgo. También se utilizan para reducir el sobre-entrenamiento.
  - **Boosting**: entrenamiento de modelos que aprenden secuencialmente, es decir, aprenden de los errores del modelo anterior. Se utiliza cuando existe baja varianza y alto sesgo.

- **Máquinas de Vectores de Soporte**

  Se trata de un algoritmo de aprendizaje automático supervisado que se puede utilizar para problemas de clasificación o regresión. Dadas 2 o más clases de datos etiquetadas, actúa como un clasificador discriminativo, definido formalmente por un hiperplano óptimo en un espacio de alta dimensión que separa todas las clases. Los nuevos ejemplos que luego se mapean en ese mismo espacio se pueden clasificar según el lado de la brecha en que se encuentran. La correcta separación de los puntos se alcanza por el hiperplano de mayor separación a los puntos de

entrenamiento más cercanos de cualquier tipo (esta distancia es el Vector de Soporte). Como regla general son excelentes para conjuntos de datos relativamente pequeños con menos valores atípicos.

- **Red Neuronal**

  Una red neuronal es un algoritmo que consiste en simular el comportamiento de un cerebro biológico mediante miles de neuronas artificiales interconectadas que se almacenan en filas llamadas capas, formando miles de conexiones. Suelen utilizarse para problemas de clasificación o regresión.

3. Resultados

3.1. Análisis estadístico y características de la muestra.

El dataset con el que vamos a crear nuestros modelos de clasificación/predicción está formado por 2362524 registros. Todas las características que tenemos en nuestro dataset son de tipo numérico, excepto las variables *address* y *label*, las cuales están en formato texto. A continuación, mostramos los 5 primeros registros de dicho dataset:

	address	year	day	length	weight	count	looped	neighbors	income	label
0	111K8kZAEnJg245r2cM6y9zgJGHZtJPy6	2017	11	18	0.008333	1	0	2	100050000	princetonCerber
1	1123pJv8jzeFQaCV4w644pzQJzVWay2zcA	2016	132	44	0.000244	1	0	1	100000000	princetonLocky
2	112536im7hy6wtKbpH1qYDWtTyMRAcA2p7	2016	246	0	1.000000	1	0	2	200000000	princetonCerber
3	1129TSjKtx65E35GiUo4AYVeyo48twbrGX	2016	238	144	0.072848	456	0	1	200000000	princetonLocky
4	112AmFATxzhuSpvtz1hfpa3Zrw3BG276pc	2016	96	144	0.084614	2821	0	1	50000000	princetonLocky

Debemos considerar de todas las variables que contiene nuestro dataset cuáles de ellas son las que realmente tienen relevancia para el problema. Para ello, realizamos un análisis de los datos y descartamos aquellas características que no aporten información o que, simplemente, son perjudiciales a la hora de crear el modelo, debido a la relación que tienen con las otras variables descriptivas. Antes de proseguir con el estudio de las características, cambiaremos los datos asociados a la variable "label", como se ha comentado en la Introducción, poniendo un 0 si el texto que aparece es "White", y un 1 en caso contrario. Esta nueva variable se llamará clase, y será la variable que queremos predecir con los modelos que creemos posteriormente.

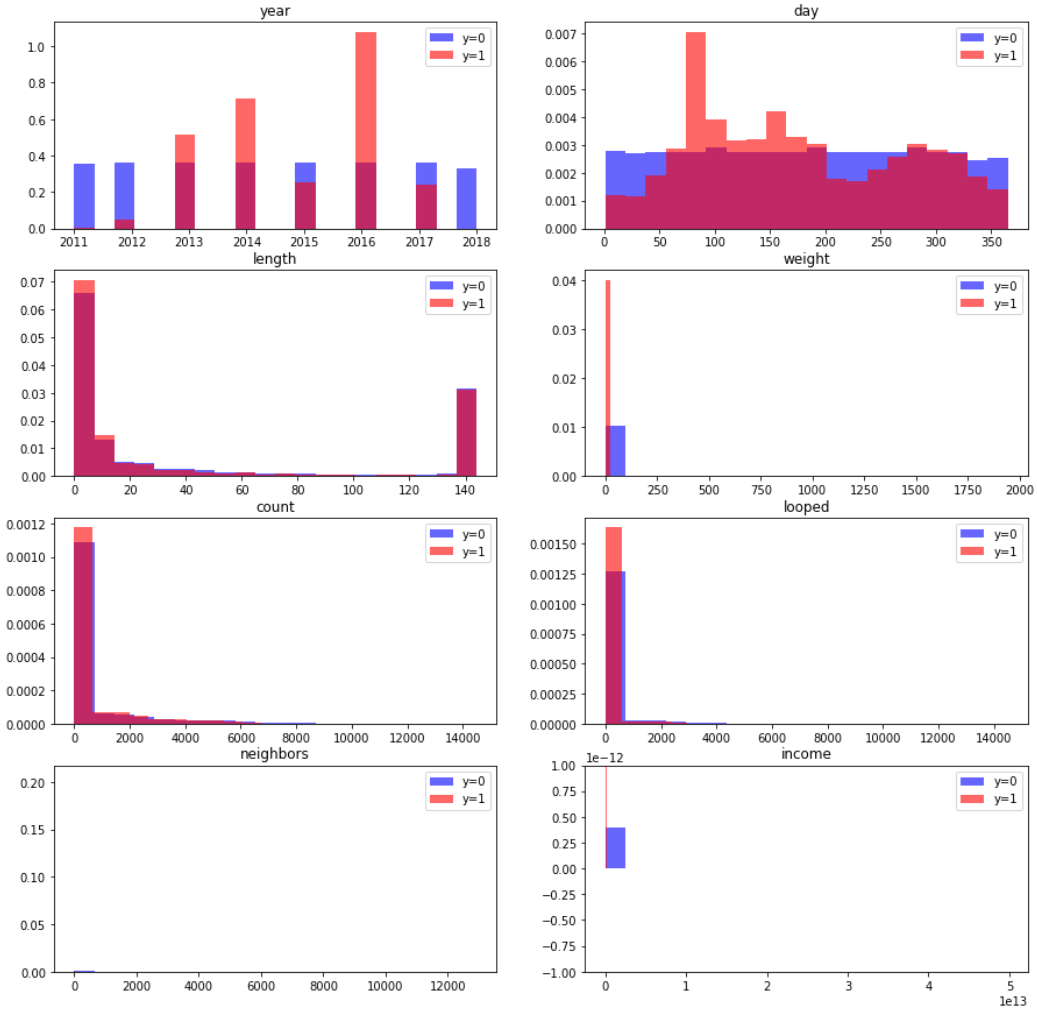
3.1.1. Variable "address"

En primer lugar, supone un problema el tipo de dato de la variable "address", y es que los algoritmos que aplicaremos solo permiten atributos de tipo numérico.

Para solventar esto se nos abren dos alternativas, utilizar un LabelEnconder() o un OneShotEncoder() de la biblioteca sklearn. El primero de estos métodos atribuye a cada objeto distinto dentro de esta columna un número entre 0 y n-1, siendo n el número total de valores distintos que puede tomar este campo. El problema de este LabelEncoder() es que, sin quererlo, estamos estableciendo un orden a un campo que no presenta ningún orden natural y esto puede inducir a error a los algoritmos. Por otro lado, el método de OneShotEncoder() genera una enorme cantidad de datos que no podemos alojar en memoria, debido al gran número de direcciones distintas que aparecen en el dataset. Por tanto, no tenemos más remedio que descartar esta variable.

3.1.2. Histogramas

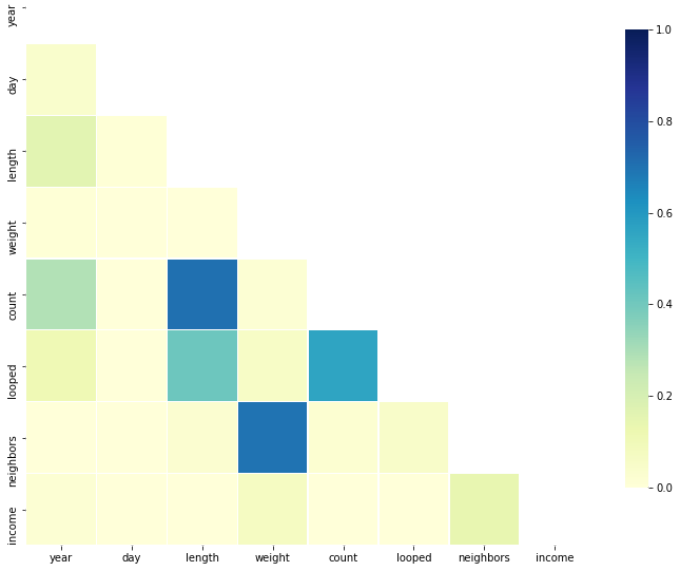
Representamos los histogramas de cada variable distinguiendo las clases. La principal ventaja de trabajar con este tipo de gráficos descriptivos en el Análisis Exploratorio de Datos (EDA) es el poder visualizar la distribución de los datos, teniendo así una idea más detallada de su comportamiento. En este sentido, el histograma se obtiene tras organizar los datos en diferentes subgrupos (bins) y realizar el conteo del número de registros en cada uno. De esta forma, podremos comprobar si existe alguna asimetría en la distribución o si puede aproximarse a una distribución Normal. A nivel de interpretación de los resultados obtenidos, comprobamos que ninguna de las variables se aproxima a esta distribución, ya que todas son asimétricas. En primer lugar, las variables de year y day tienen comportamientos dispersos, agrupando sus máximos en 2016 y entre los días 75-100, respectivamente. El resto de variables se encuentran concentradas a la izquierda, en torno a valores cercanos o iguales a 0. Por otro lado, el diferenciar las clases en el histograma nos permite comprobar si hay alguna tendencia en la concentración de registros de cada clase.



3.1.3. Matriz de correlación

Con esta matriz podemos observar qué variables presentan mayor colinealidad. Es útil cuando se pretende analizar qué características no aportan información relevante.

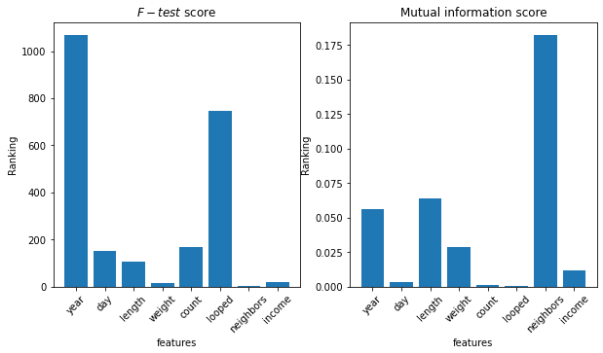
La mayor colinealidad vale 0.7 y se obtiene entre las variables “count” y “length” y entre “weight” y “neighbors”. Lo ideal sería descartar, de cada par, una de las dos características.



3.1.4 F-test y Mutual Information

Calculando las puntuaciones del F-test y Mutual Information obtenemos la dependencia lineal y no lineal que tienen cada una de las características con la variable objetivo. Es decir, con la etiqueta binaria que hemos asignado y que queremos predecir. Según la gráfica, las variables más importantes son “year”, “looped” y “neighbors”.

Con este resultado, elegimos la variable “neighbors” y descartamos “weight” debido a la gran relevancia no lineal que muestra “neighbors” con la variable “Clase”.



3.1.5 Método Embedded para filtrado

Por último, con el algoritmo LASSO mediremos la importancia de cada variable. Para ello, separamos los registros en datos para entrenamiento y para test. La división se hace de tal forma que solo se toma el 20% para testeo.

En el siguiente paso, mediante validación cruzada, se calcula el valor de α óptimo. En nuestro caso, obtenemos $\alpha = 4.52 * 10^{-6}$. Al entrenar LASSO con este parámetro, se obtienen los siguientes coeficientes:

```
MSE Modelo Lasso (train): 0.014

MSE Modelo Lasso (test) : 0.0138

RMSE Modelo Lasso (train): 0.118

RMSE Modelo Lasso (test) : 0.118

year: 0.002947121391430012

day: -0.0008356818659241389

length: 0.0001446621872715825

weight: 0.0007508159153221959

count: -0.0008515920369221731

looped: -0.002045335630045483

neighbors: -0.00048301289091627706

income: -0.00024255641935500008
```

Debido a que el coeficiente asociado a “length” es menor en valor absoluto que el asociado a “count”, eliminamos esa variable. Además, lo mismo ocurriría con las variables “weight” y “neighbors”, y eliminaremos la variable “weight” para evitar la correlación de estas dos variables.

En resumen, el dataset con el que trabajaremos consta de los siguientes campos: year, day, count, looped, neighbors, Income y Clase.

3.1.6. Conjunto de entrenamiento y validación

De la colección de datos de la que partimos escogemos aleatoriamente un 20% para testear los métodos desarrollados.

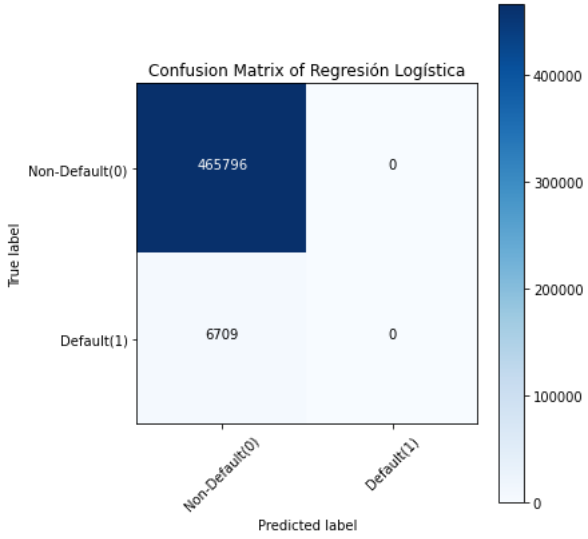
Al hacer esta división, encontramos que el dataset destinado a entrenamiento presenta clases desbalanceadas. En concreto, hay 1863183 registros de clase 0 contra únicamente 26836 registros de la clase 1. Para poder mejorar el entrenamiento de los diferentes algoritmos se aplica la técnica de Random Undersampling al conjunto de entrenamiento. Con esta técnica podemos muestrear la clase mayoritaria y por tanto reducir su número de elementos para que quede la proporción entre clases que se desee. En este caso, se ha balanceado de forma que el 33% del conjunto de entrenamiento se corresponde con transacciones con ransomware y el 66% a transacciones limpias.

Dicho balance se ha realizado únicamente sobre el dataset de entrenamiento, ya que consideramos que es mejor no hacerlo sobre el conjunto de prueba. Esto se debe a que el objetivo de este subconjunto es precisamente comprobar la bondad de nuestro modelo, por lo que debe asemejarse lo máximo posible a la realidad y, en general, se pretende clasificar datos no balanceados.

3.2. Resultados de los algoritmos.

3.2.1. Regresión logística binaria

Se aplica regresión logística con un parámetro $C = 10^{-10}$ que resulta como valor óptimo tras realizar validación cruzada. La matriz de confusión indica que todos los registros se clasifican como no fraudulentos.

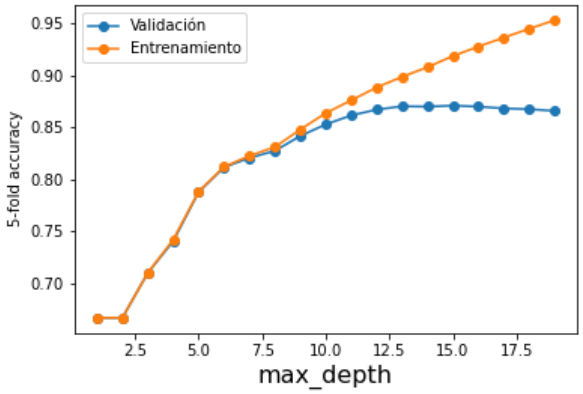


Para intentar mejorar este resultado se ha probado a introducir pesos a las clases y realizar bagging con la regresión logística como algoritmo base, pero el resultado sigue siendo el mismo.

3.2.2. Decision Tree

Otra aproximación al problema que pretendemos resolver puede ser mediante árboles de decisión binarios, considerando 0 cuando no hay ningún tipo de ransomware y 1 en caso contrario, tal y como se ha explicado anteriormente.

Para dicho árbol de decisión se ha utilizado la implementación que tiene la biblioteca scikit-learn mediante la función DecisionTreeClassifier(). Además hemos utilizado GridSearchCV() como función de validación cruzada para la optimización de sus hiperparámetros, en este caso, la profundidad máxima del árbol.

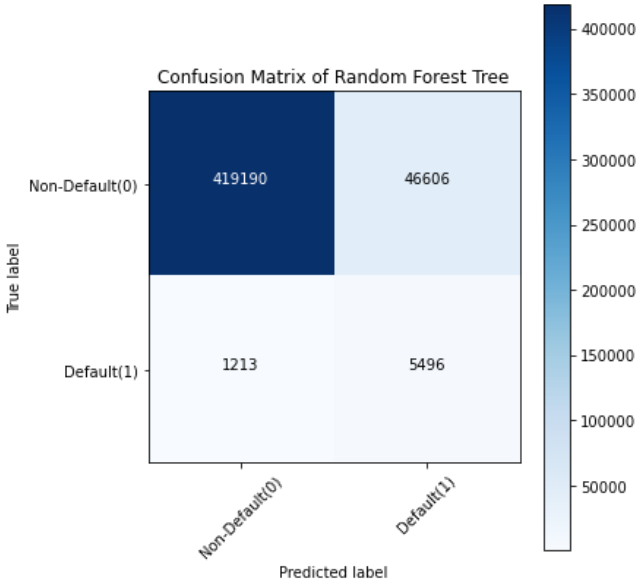


En la imagen anterior se puede apreciar como, sobre la profundidad 9, es cuando el accuracy sobre el conjunto de entrenamiento empieza a crecer a mayor escala que sobre el conjunto de validación. Sin embargo, no es hasta que tenemos árboles de profundidad 15 cuando llegamos a un accuracy máximo en el conjunto de validación, y a partir de dicha profundidad el valor de esta métrica crece únicamente sobre el entrenamiento, disminuyendo en la validación. Esto es un indicio claro de que se produce overfitting con árboles de decisión de profundidad mayor a 15, y por tanto, este será el valor que utilicemos a la hora de entrenar nuestro árbol.

Tras dicho entrenamiento obtenemos un accuracy de:

- 0.9160 sobre el entrenamiento
- 0.8987 sobre el conjunto de testeo

Esta métrica, siendo importante, no debe ser la única a considerar teniendo en cuenta la cantidad de datos de cada clase que están presentes en el proceso de entrenamiento. Es por ello que se obtiene la matriz de confusión.



Esta nos permite conocer mucha más información acerca de la bondad de nuestro algoritmo. En esta matriz, lo primero que se observa es la gran cantidad de 0s que hay en relación a 1s. Esto es debido, como ya mencionamos anteriormente, a que solamente se balanceó el subconjunto train, manteniendo el de testeo con todos los datos, y por tanto, con las proporciones de elementos de cada clase intactas, para comprobar cómo de bueno es un modelo sobre las condiciones reales. Dicho esto, podemos calcular con facilidad distintas métricas que nos van a permitir obtener mucha más información acerca de nuestro modelo.

- Accuracy: 0.8987
- Recall: 0.8191
- Specificity: 0.8999
- Precision: 0.1054
- F1-score: 0.1867

Tanto el accuracy, que puede resultar engañoso al no estar balanceados los datos, como el recall y el specificity, que sí son realmente importantes en estos casos, obtienen una puntuación que supera el 80%. Que estas dos últimas métricas tengan un valor elevado es precisamente lo que estamos buscando puesto que nos indican el porcentaje de elementos clasificados correctamente de cada clase, es decir, de todos los casos positivos, cuántos fueron clasificados correctamente por el algoritmo (Recall), y de todos los casos negativos, cuántos fueron clasificados como negativos (Specificity). Sin embargo, podemos ver que obtenemos valores pésimos, inferiores al 20% en precision y f1-score. Esto es algo que puede suceder cuando trabajamos con conjuntos desbalanceados donde la clase mayoritaria es el 0. A pesar de tener mayor especificidad que exhaustividad, al haber tal desproporción de clases como hay en los datos sin balancear, hay muchos más falsos

positivos que verdaderos positivos, lo que provoca esa baja precisión y, consecuentemente, un valor bajo en la puntuación f1.

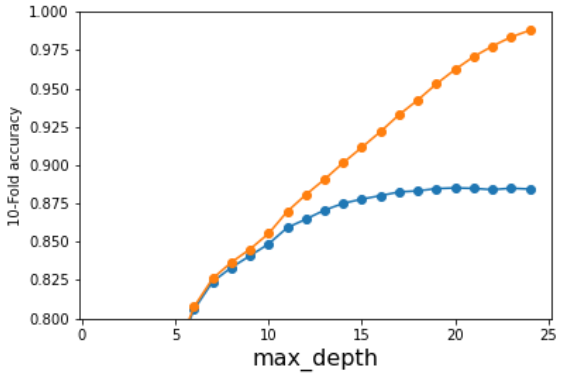
De todas formas, analizando el tipo de problema, nosotros lo que queremos es que toda transacción afectada por ransomware sea detectada, aunque se encuentren muchos falsos positivos. Lo que no podemos permitir de ninguna manera son los falsos negativos, puesto que se están colando transacciones afectadas sin que sean detectadas. Idealmente debería buscarse un 1 en cada una de estas métricas, pero siendo realistas nos interesa más un alto specificity, y recall sobre todo, aunque esto conlleve una baja precisión.

Observando una representación gráfica del árbol para detectar dónde se han producido las equivocaciones, apreciamos una gran cantidad de hojas con gini 0, incluso a altas profundidades. Sin embargo, también hay otras hojas con valores gini elevados, llegando incluso al 0.5 en algunos casos, el valor más alto posible. Lo que significa que llegados a esas hojas no sabemos a qué clase pertenecen los datos.

3.2.3. Random Forest

Siempre que podemos aplicar árboles de decisión a un problema, una alternativa interesante radica en los bosques aleatorios o random forests. Al igual que en el caso anterior, la biblioteca scikit learn tiene una implementación de este tipo de algoritmos, RandomForestClassifier().

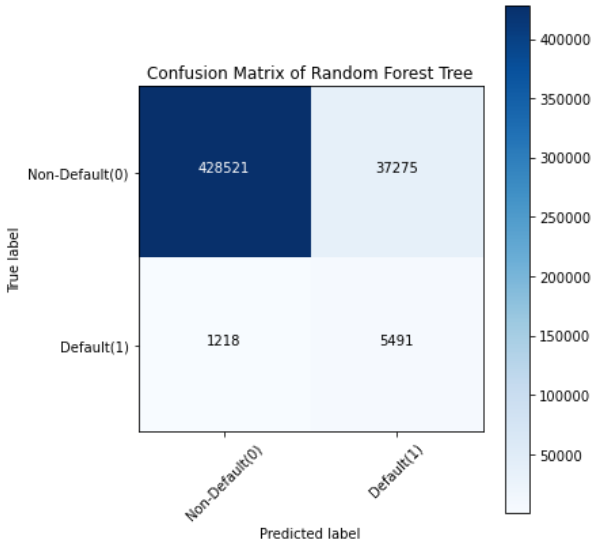
En este caso se han implementado dos aproximaciones distintas al problema siguiendo este algoritmo. En la primera tan solo hemos utilizado la validación cruzada para optimizar la profundidad máxima que van a tener los distintos árboles que compondrán este bosque aleatorio, ya que esta no tiene por qué coincidir con la profundidad óptima de un único árbol. De hecho, no ha sido así. En la siguiente imagen podemos ver como en este caso la profundidad óptima de los árboles ha aumentado hasta situarse en un valor de 20.



También ha aumentado el accuracy del ajuste durante la validación cruzada, llegando en este caso al 88.5%. Sin embargo, este accuracy no es el que nos interesa, ni siquiera el

propio accuracy es la métrica que más nos interesa, como ya vimos anteriormente. Una vez hallada la profundidad máxima de los árboles que constituirán el algoritmo random forest, tenemos que entrenar nuestro modelo con dicha profundidad. Además, se ha establecido que el número de árboles sea 50 en esta primera aproximación, y que el tamaño de los subconjuntos de características que se deben tener en cuenta sea escogido mediante el criterio "*sqrt*" , es decir, la raíz cuadrada del número de características.

Tras dicho entrenamiento, el modelo resultante nos ha proporcionado un accuracy de 0.9603 sobre el subconjunto train y 0.9185 sobre el de testeo, obteniendo además una matriz de confusión como la que podemos ver en la siguiente imagen.

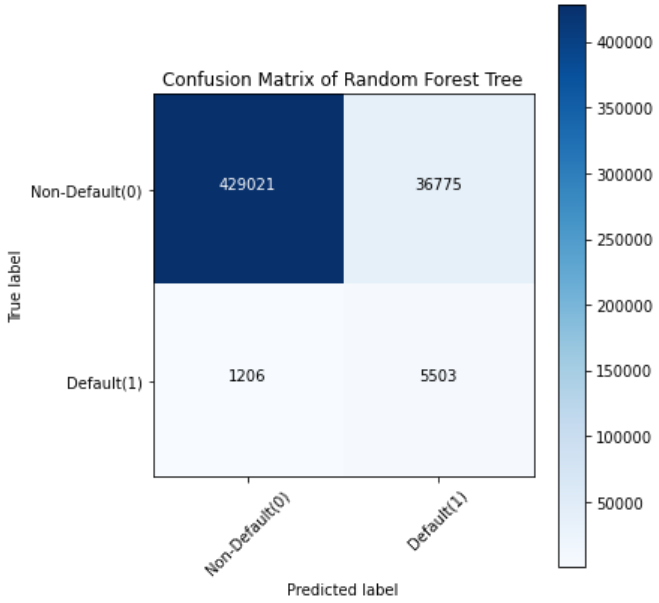


Tras calcular las principales métricas para medir la bondad de los algoritmos de clasificación que son fáciles de hallar a partir de esta matriz, obtenemos los siguientes resultados:

- Accuracy: 0.9185
- Recall: 0.8184
- Specificity: 0.9199
- Precision: 0.1283
- F1-score: 0.2218

Esta implementación de random forest ha resultado que tiene un recall ligeramente más bajo, tiene menor exhaustividad en un 0.07%. Sin embargo, sí que tiene un valor superior en el resto de métricas. Aun así, tan solo hemos optimizado un hiperparámetro, habiendo establecido valores arbitrarios para los otros que hemos comentado. Por ello, la segunda alternativa que hemos realizado mediante random forest consiste, precisamente, en una validación cruzada en la que se ha optimizado no sólo el valor de la profundidad máxima de los árboles, sino también el número de árboles y el valor de "*max_features*".

Tras dicho proceso de validación, ha resultado que el mejor Random Forest encontrado es un modelo con un total de 200 árboles con una profundidad máxima de 21 cada uno de ellos. Además el tamaño óptimo de los subconjuntos de características viene determinado por la técnica "*auto*". Esta técnica establece este valor del mismo modo que lo hacía "*sqrt*", por lo que en este sentido no hay diferencia. Simplemente hemos aumentado el número de estimadores en un 300%. Esto ha permitido que, tras un proceso de entrenamiento, el accuracy del entrenamiento haya pasado a 0.9710 sobre el training y 0.9196 sobre el test. Como vemos, no hay una gran diferencia en lo que en exactitud se refiere.



Sin embargo, esta diferencia sí que es más apreciable si miramos cada una de las distintas métricas que hemos ido obteniendo con ayuda de la matriz de confusión, exactitud, exhaustividad, especificidad, precisión y puntuación f1.

- Accuracy: 0.9196
- Recall: 0.8202
- Specificity: 0.9210
- Precision: 0.1301
- F1-score: 0.2245

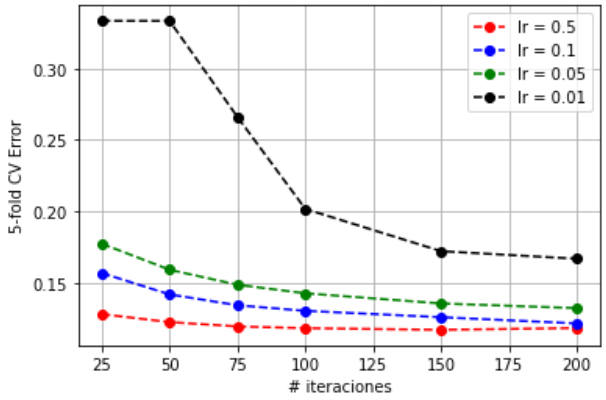
Este nuevo bosque aleatorio supera al anterior en todas ellas, e incluso tiene un recall superior a los árboles de decisión.

3.2.3. Boosted Trees

Otro algoritmo de aprendizaje conjunto es el GradientBoostingClassifier(), también de la biblioteca scikit learn. En este caso, al igual que con random forest, se han realizado

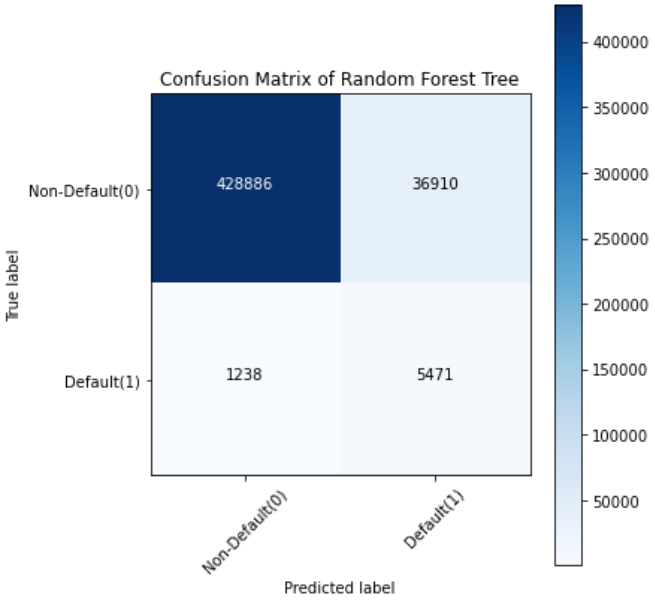
dos aproximaciones distintas para la resolución del problema. En la primera se ha establecido arbitrariamente la profundidad máxima de los árboles que constituyen el modelo. En este caso, el valor escogido ha sido una profundidad baja, 5 solamente, debido a la tardanza de este tipo de algoritmos en el proceso de validación cruzada cuando se trabaja con árboles de gran profundidad, o incluso cuando también se quiere establecer el valor óptimo de este hiperparámetro junto con los demás.

Con esta profundidad inicial, se ha obtenido que el número óptimo de árboles era 150, con un learning rate óptimo, o tasa de aprendizaje, de 0.5, tal y como podemos ver en la siguiente gráfica del error producido durante el proceso.

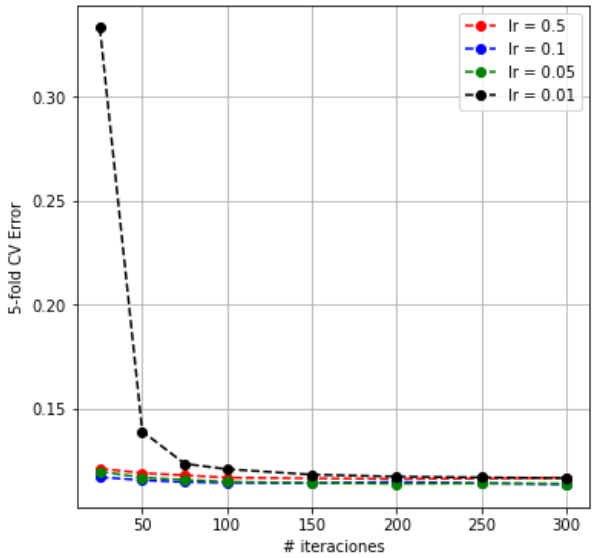


Tras establecer estos como los valores para dichos parámetros en la realización de un proceso de entrenamiento del modelo, hemos obtenido valores de accuracy bastante buenos, 0.9161 para el train y 0.9192 para el test. Además se ha obtenido una matriz de confusión como la que veremos a continuación, que nos ha permitido calcular los siguientes resultados:

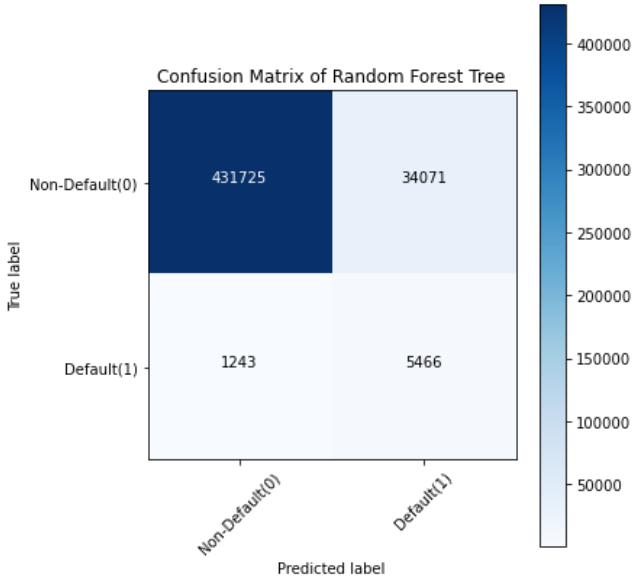
- Accuracy: 0.9192
- Recall: 0.8154
- Specificity: 0.9207
- Precision: 0.1290
- F1-score: 0.2227



Resultados muy similares a los del último random forest, aunque ligeramente inferiores en todas las métricas. Esto nos llevó a plantearnos nuestra segunda aproximación por medio de boosted trees, añadir un mayor número de árboles y, si bien no consideramos viable optimizar la profundidad máxima mediante GridSearchCV() por el tiempo que esto requeriría, sí que pretendíamos determinar una cifra que no sea meramente arbitraria. Para esto último decidimos realizar validación cruzada de un único árbol de decisión con el conjunto de datos que se va a usar para el modelo boosted trees, optimizando la profundidad máxima de dicho árbol. Una vez obtenido este valor, establecerlo como la profundidad de cada uno de los árboles del nuevo modelo de boosting.



Esta nueva validación cruzada del modelo GradientBoostingClassifier() con una profundidad máxima de 15, la misma que la óptima del árbol de decisión entrenado con sus mismos datos, nos ha indicado que el número de árboles en este caso debe ser 300, con una tasa de aprendizaje de 0.1, tal y como vemos en la imagen anterior. En esta nueva implementación de los boosted trees la tasa de aprendizaje ha disminuido, mientras que el número de iteradores ha aumentado con respecto a la anterior. Pero esto no es lo único que ha aumentado, también lo ha hecho el accuracy obtenido tanto en el conjunto de entrenamiento, con un valor de 0.9978 en este caso, como en el conjunto de testeo, donde la exactitud obtenida ha sido de 0.9252.



Por otro lado, la matriz de confusión obtenida se corresponde con la imagen anterior y los valores para el resto de métricas asociadas a esta matriz son los siguientes:

- Accuracy: 0.9252
- Recall: 0.8147
- Specificity: 0.9268
- Precision: 0.1382
- F1-score: 0.2363

Apreciamos como aunque la exhaustividad es ligeramente inferior a la del modelo boosted trees anterior, la especificidad es superior en mayor medida. Esto hace que tanto la precisión, como, consecuentemente, la puntuación f1 y la exactitud de este modelo sean superiores a las del boosting realizado con anterioridad.

3.2.4. Máquinas de Vectores de Soporte

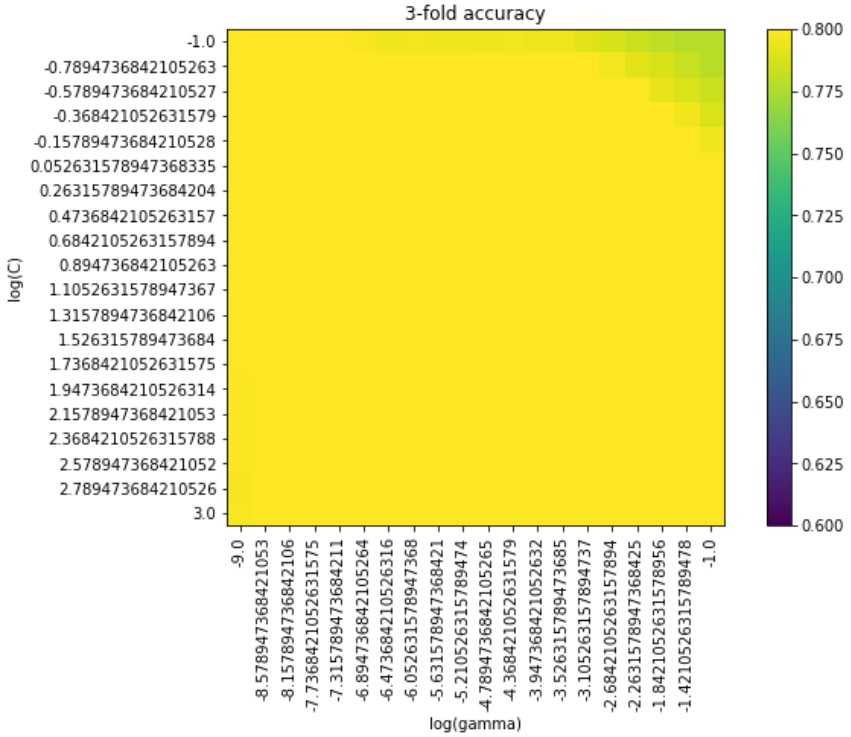
En este caso, se crea un modelo de máquinas de vectores soporte (*Support Vector Machine*) para clasificación, las cuales se basan en la construcción de un hiperplano en un espacio de alta dimensión, donde la separación de los puntos alcanzada por el hiperplano de mayor separación a los puntos de entrenamiento corresponderá al Vector de Soporte. Para el propósito de esta actividad, el algoritmo en cuestión es interesante ya que sus principales ventajas se basan en el trabajo con pocos datos de entrenamiento y en espacios de alta dimensión. Sin embargo, sus desventajas principales son el elevado coste computacional y el sobre-entrenamiento. En este sentido, no se ha podido crear el modelo para todo el dataset de entrenamiento, por lo que se ha seleccionado de forma aleatoria una submuestra de 6000 registros (4672 de clase 0, 1328 de clase 1), la cual ha conllevado a los resultados que se presentan a continuación.

En primer lugar, para la búsqueda de parámetros libres, utilizamos los vectores que nos permiten trabajar con el parámetro C (penalización de errores de clasificación) y gamma (). Además, las máquinas de vectores soporte también destacan por su versatilidad, ya que presentan diferentes kernels a elegir (lineal, sigmoidal, polinomial y Función Radial Básica). En nuestro caso, nos quedamos con este último para el modelo, creándolo a través de la función *SVC*. Posteriormente, la función *GridSearchCV* crea un diccionario que describe los parámetros que se pueden probar en un modelo para entrenarlo, a la cual le pasaremos el grid con los parámetros creados y el número de folds (validación cruzada). Esta función requiere mucho tiempo y es costosa desde el punto de vista computacional, pero suele llevar a la mejor combinación de parámetros. Por último, ajustamos los datos de entrenamiento con la función *fit*.

```
GridSearchCV(cv=3, estimator=SVC(),
             param_grid={'C': array([1.00000000e-01, 1.62377674e-01, 2.63665090e-01, 4.28133240e-01,
       6.95192796e-01, 1.12883789e+00, 1.83298071e+00, 2.97635144e+00,
       4.83293024e+00, 7.84759970e+00, 1.27427499e+01, 2.06913808e+01,
       3.35981829e+01, 5.45559478e+01, 8.85866790e+01, 1.43844989e+02,
       2.33572147e+02, 3.79269019e+02, 6.15848211e+02, 1.00000000e+03]),
                          'gamma': array([1.00000000e-09, 2.63665090e-09, 6.95192796e-09, 1.83298071e-08,
       4.83293024e-08, 1.27427499e-07, 3.35981829e-07, 8.85866790e-07,
       2.33572147e-06, 6.15848211e-06, 1.62377674e-05, 4.28133240e-05,
       1.12883789e-04, 2.97635144e-04, 7.84759970e-04, 2.06913808e-03,
       5.45559478e-03, 1.43844989e-02, 3.79269019e-02, 1.00000000e-01])},
             scoring='accuracy')
```

A través de la validación cruzada, obtenemos el siguiente gráfico, el cual devuelve los parámetros óptimos (C y gamma) para obtener los mejores datos de precisión (*Best Mean Cross-Validation Score*).

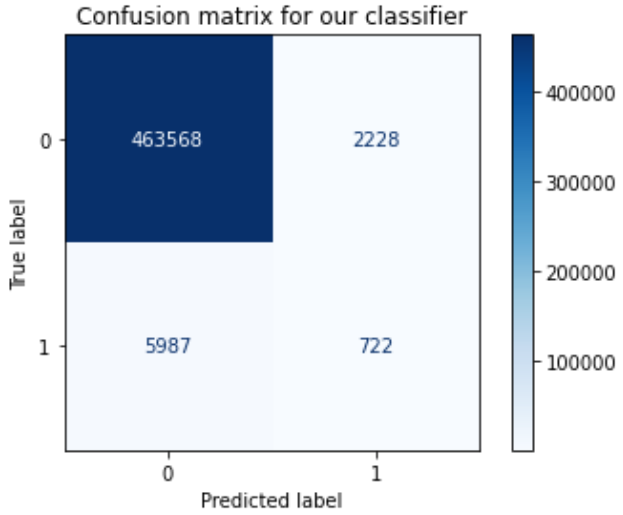
```
best mean cross-validation score: 0.825
best parameters: {'C': 12.742749857031335, 'gamma': 0.00206913808111479}
```



Por último, utilizamos esos parámetros óptimos y creamos la SVM definitiva, evaluando los resultados a través de la precisión del dataset de train y test:

```
Accuracy (TEST): 0.960
Accuracy (TRAIN): 0.759
```

Por otro lado, la matriz de confusión obtenida se corresponde con la imagen anterior y los valores para el resto de métricas asociadas a esta matriz son los siguientes:



- Accuracy: 0.9826
- Recall: 0.1076
- Specificity: 0.9952
- Precision: 0.2447
- F1-score: 0.1494

Como se puede comprobar, la mayoría de las medidas arrojan un modelo bastante preciso (excepto la especificidad, todas están por encima del 98%), aunque esto podría estar influido por el elevado número de registros de clase 0 a predecir. Si nos fijamos en lo que nos interesa predecir (clase 1), precisamente es la especificidad la métrica que mejor evalúa los resultados, alcanzando está un valor muy pobre (por debajo de 25%).

3.2.5. Red Neuronal

A continuación se presentan los resultados obtenidos tras el entrenamiento neuronal para realizar la clasificación binaria. Como ya se ha explicado, se considera 0 cuando no hay Ransomware y 1 cuando sí lo hay.

Para codificar la red neuronal y construir el modelo, se ha utilizado la librería Keras de TensorFlow. Su arquitectura se compone de los siguientes elementos:

- Modelo secuencial
- Capa de entrada de dimensión 6 (por introducir 6 características), de densidad 256 y función de activación Relu.
- Capa intermedia de densidad 256 y función de activación Relu.
- Capa de salida de dimensión 1 (la salida es binaria) y función de activación Sigmoid.
- Compilación con la función de pérdida Binary Cross Entropy.

Sobre esta arquitectura se han realizado 3 pruebas variando la muestra de test, el tamaño de los batches y las etapas de entrenamiento.

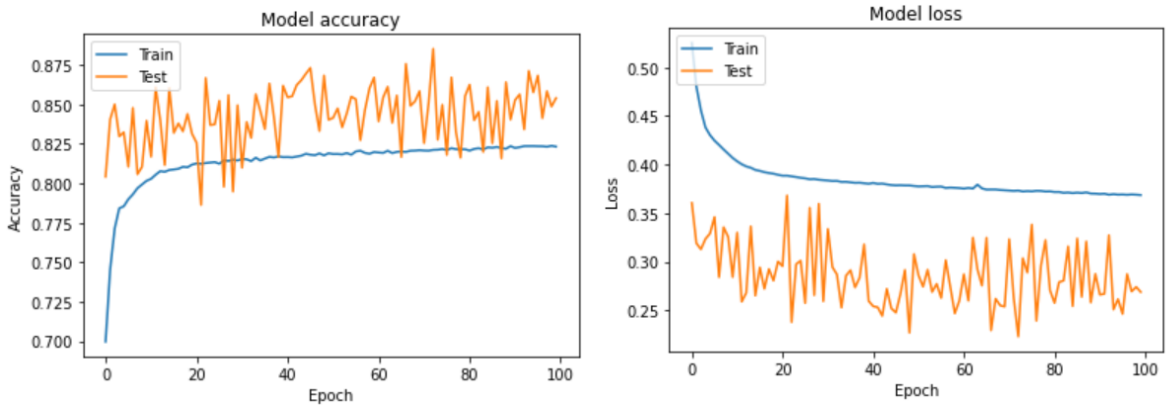
<u>Prueba 1: Muestra de train balanceada y test desbalanceada</u>

Para la primera prueba se ha entrenado el modelo con los datos balanceados pero se ha probado el modelo con la muestra de test con los datos no balanceados. Es decir, que se realiza la prueba sobre una muestra aleatoria. Se han entrenado batches de tamaño 8051 durante 100 épocas, obteniéndose los siguientes resultados:

```
14766/14766 [==============================] - 20s 1ms/step - loss: 0.2688 - accuracy: 0.8541
[0.26883891224861145, 0.8540713787078857]
```

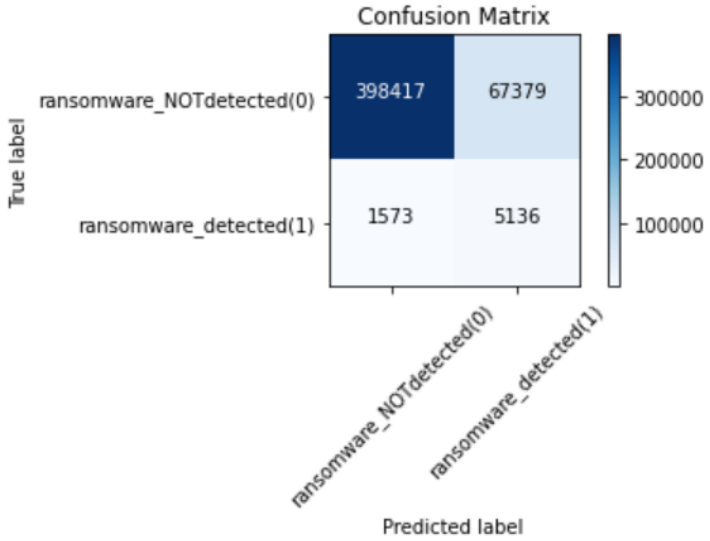
- Accuracy: 0.8540
- Loss: 0.2688
- Recall: 0.8165
- Specificity: 0.8349
- Precision: 0.0708

Vemos que tenemos una baja precisión y una alta exhaustividad (Recall), por lo que el modelo detecta bien la clase pero también incluye muestras de otras clases.



Resulta curioso como la gráfica de test presenta mejores resultados en cuanto al Accuracy y al Loss que la gráfica de Train. Parece que la gráfica de test tiene bastante ruido. Como ya se ha comentado anteriormente, el entrenamiento se realiza sobre una muestra de datos balanceada. Si posteriormente este modelo se prueba con un subconjunto no balanceado, es posible que los resultados obtenidos no sean tan fieles al aprendizaje.

Aquí vemos la matriz de confusión, la cual demuestra que hay un alto grado de accuracy por predecir un alto grado de verdaderos positivos y verdaderos negativos.



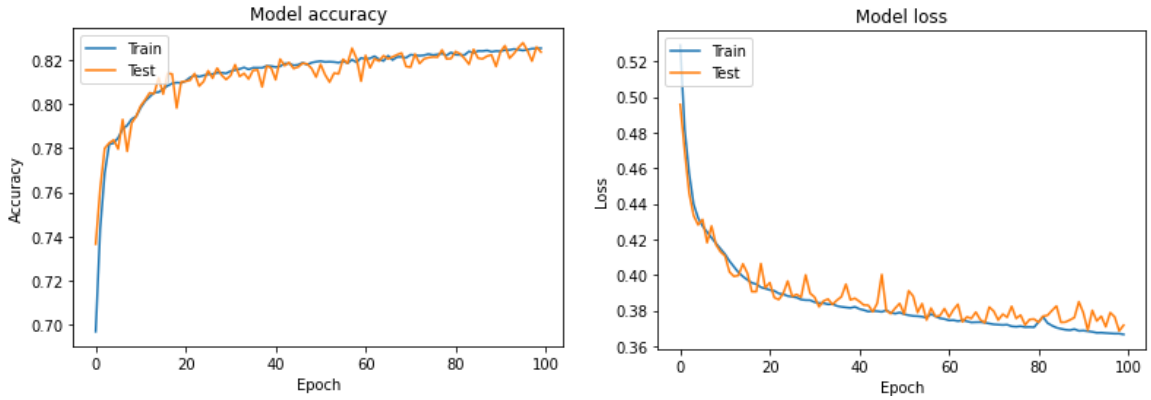
Prueba 2: Muestra de train y test balanceada

En esta segunda prueba se ha tomando la base de datos de test (20% del total de la muestra proporcionada previamente balanceada) y se han entrenado batches de tamaño 8051 durante 100 épocas. Dado que en esta prueba los datos de test se encuentran balanceados, hay menos muestra. Se han obtenido los siguientes resultados:

```
629/629 [==============================] - 1s 1ms/step - loss: 0.3717 - accuracy: 0.8237
[0.3716980218887329, 0.8236696720123291]
```

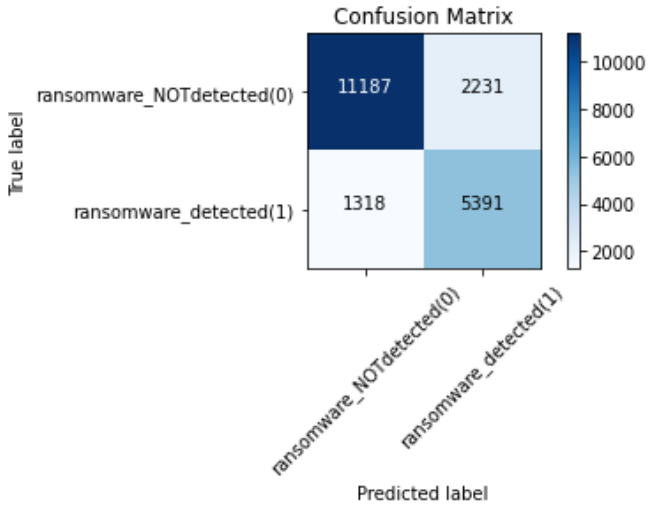
- Accuracy: 0.8237
- Loss: 0.3717
- Recall: 0.8035
- Specificity: 0.8337
- Precision: 0.7072

Vemos que tenemos una precisión y una exhaustividad bastante alta, por lo que se podría considerar que el modelo maneja bastante bien la clase.



Como se puede observar, las pruebas sobre el modelo de test muestran un buen grado de accuracy. La curva de test se ajusta bastante bien a la de train, especialmente en la gráfica de accuracy.

En la siguiente gráfica se muestra la matriz de confusión. Como se puede ver, los números más altos se encuentran en la diagonal, y esto lo que representa es que hay un número considerable de verdaderos positivos y verdaderos negativos.



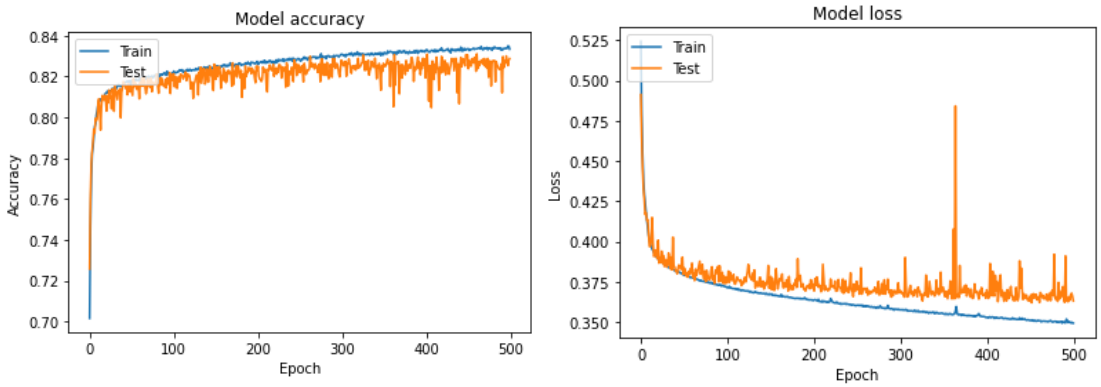
Prueba 3: prueba de train y test balanceada y aumento de las etapas de entrenamiento

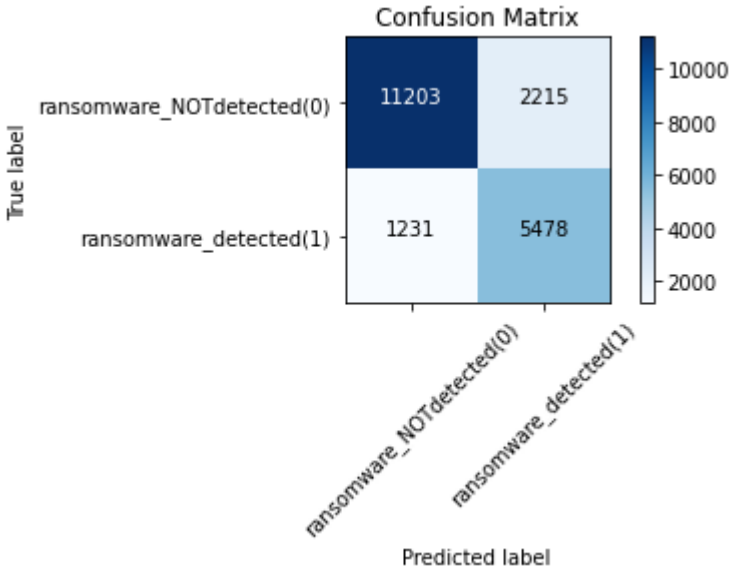
Viendo que los resultados en la primera prueba fueron bastante prometedores y la gráfica mostraba indicios de la mejora del aprendizaje del modelo, en la segunda prueba se han entrenado batches de tamaño 8051 (también de la base de datos reservada de test) durante 500 épocas, obteniéndose los siguientes resultados:

```
629/629 [==============================] - 1s 2ms/step - loss: 0.3630 - accuracy: 0.8288
[0.362983763217926, 0.8287872076034546]
```

- Accuracy: 0.8287
- Loss: 0.3629
- Recall: 0.8165
- Specificity: 0.8349
- Precision: 0.7120

Vemos que tenemos una precisión y una exhaustividad bastante alta, por lo que se podría considerar que este modelo también maneja bastante bien la clase.





Se puede observar que el modelo no ha mejorado las predicciones de la forma que se esperaba. De hecho, parece que a partir de la etapa 150, ambas líneas de train y test comienzan a distanciarse y a generar predicciones más alejadas de la realidad. En la gráfica de Loss se puede observar un dato atípico. Esto lo que puede representar es un sobre entrenamiento, de tal forma que, a pesar de entrenar durante 5 veces más de tiempo, el modelo no ha conseguido mejorar de forma proporcional.

Podemos concluir que el primer modelo, a pesar de tener valores ligeramente peores, es mejor. No sufre sobre entrenamiento y es mucho más eficiente, pues consigue prácticamente los mismos resultados en sus predicciones en la quinta parte de tiempo.

Como mejoras de implementación futuras para el entrenamiento de la red neuronal se puede utilizar TensorBoard, herramienta que permite analizar el accuracy y el loss gráficamente en tiempo real e incluso realizar “Early stopping” durante el entrenamiento cuando se detecta que el algoritmo no continúa aprendiendo.

4. Conclusiones

Para tomar una decisión sobre cuál de los anteriores modelos es, bajo nuestro punto de vista, el mejor para predecir nuevos datos, nos centraremos centraremos en las siguientes métricas:

- **Accuracy.** Gracias a esta métrica conoceremos cómo de bien predice nuestro modelo los datos test.
- **Recall.** En nuestro modelo vamos a intentar siempre reducir la tasa de falsos negativos, es decir, de aquellos que se han predecido como negativos, siendo realmente positivos, ya que es importante detectar todos los posibles casos de ransomware. Por ello, esta métrica será bastante importante a la hora de determinar el modelo (diferenciamos la clase de los positivos correctamente).
- **Specificity.** Con esta métrica observaremos cómo de bien detectamos la clase de los negativos. No será la más importante de todas a tener en cuenta, pero servirá de ayuda.
- **Precision.** Con esta métrica sabremos qué porcentaje de predicciones positivas son realmente positivas, por lo que le daremos bastante importancia a la hora de elegir el modelo.
- **F1-Score:** Es la media armónica entre la precisión y la exhaustividad. Mide cómo de preciso es el clasificador y cómo es de robusto.

De todos los modelos resultantes en la sección anterior, no tendremos en cuenta el modelo de regresión logística, ya que los resultados son bastante malos (se comprueba observando la matriz de confusión).

Por otro lado, en cuanto a las redes neuronales, hemos decidido obviar los modelos de redes neuronales resultantes cuando la muestra test y train están balanceadas, ya que se acordó que se realizarían las pruebas de validación sobre una muestra de datos de test NO balanceada (Prueba 2 y Prueba 3). Estas muestran mejores resultados en cuanto a precisión y F1 score, además de presentar gráficas de *accuracy* y *loss* mucho más ajustadas. De esta forma, se ha decidido mantenerlas en el informe debido a los buenos resultados obtenidos y como información para estudios futuros, aunque no se tengan en cuenta para realizar la comparación entre los diferentes algoritmos. Por tanto, los modelos finales a tener en cuenta para la comparación serán los siguientes:

- Decision Tree.
- Random Forest.
- Boosted Trees.
- Máquinas de Vectores Soporte.
- Red Neuronal (Prueba 1).

De todos ellos, el que mejor accuracy tiene es el de Máquinas de Vectores Soporte, pero, sin embargo, el recall asociado tiene un valor de 0,1076, lo cual nos dice que no detecta

bien a los que son realmente positivos. Por ello, pasamos a ver quién tiene el segundo mejor accuracy. En este caso, es el modelo resultante por Boosted Trees, del cual se obtiene un accuracy del 0.9252 de la muestra test. Además, si nos fijamos en el Recall, es bastante alto (0.8147), y se diferencia en muy pocas milésimas de los asociados a los demás modelos. Con lo cual, parece que tenemos un modelo que detecta con bastante certeza la clase de los positivos. Por otro lado, la métrica Specificity de este modelo tiene el mejor valor de todos los modelos (0.9268), por lo cual también diferencia bastante bien la clase de los negativos. Además, la métrica Precision de este modelo es de 0.1382, la cuál es la segunda más alta de los modelos que estamos comparando. Esto nos dice, que el porcentaje de predicciones positivas correctas no es muy alto, pero, al tener un alto recall, aquellos casos que sean realmente positivos se predicen con una alta probabilidad como positivos. De esta forma, esto es lo que estábamos buscando, intentar reducir el número de falsos negativos, es decir, detectar bien la clase de los positivos.

Por tanto, de todos los modelos que hemos desarrollado a lo largo de la actividad, consideramos que aquel modelo que mejor se ajusta a los datos es el modelo resultante al realizar Boosted Trees.

5. Anexos

En este último apartado se incluye una carpeta compartida de Google Drive, donde se almacenan los Notebooks con los algoritmos utilizados y el análisis descriptivo previo (junto a la selección de características), además de los ficheros CSV con las predicciones realizadas sobre el conjunto de test:

https://drive.google.com/drive/folders/1BtsTtU-GHRJoxhjigB31apTAREd2LG4-?usp=sharing