Trabajo Fin de Máster

Modelado de Vulnerabilidades mediante técnicas de Machine Learning

Julio José Rodríguez Martínez

Máster en Técnicas Estadísticas

Curso 2022-2023

Propuesta de Trabajo Fin de Máster

Título en galego: Modelado de vulnerabilidades mediante técnicas de Machine Learning
Título en español: Modelado de Vulnerabilidades mediante técnicas de Machine Learning
English title: Vulnerability modeling using Machine Learning techniques
Modalidad: Modalidad B
Autor/a: Julio José Rodríguez Martínez, Universidade de Santiago de Compostela
Director/a: Marta Sestelo Pérez, Universidade de Vigo
Tutor/a: Laura Pérez Vilarelle, Gradiant
Breve resumen del trabajo: Para determinar la explotabilidad de un CVE se utilizarán técnicas de balanceo y distintos criterios de selección de variables que permitan mejorar el rendimiento de los modelos utilizados. Se ha probado con el modelo de regresión logística, GAM y *Random Forest*, siendo este último el que mejor rendimiento ha presentado.
Recomendaciones:
Otras observaciones:

Doña Marta Sestelo Pérez, Profesora Ayudante Doctora de la Universidade de Vigo y doña Laura Pérez Vilarelle, Investigadora Senior de Gradiant informan que el Trabajo Fin de Máster titulado

Modelado de Vulnerabilidades mediante técnicas de Machine Learning

fue realizado bajo su dirección por don Julio José Rodríguez Martínez para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal.

En Vigo, a 27 de Enero de 2023.

SESTELO PEREZ MARTA - 76998300G

Firmado digitalmente por SESTELO PEREZ MARTA - 76998300G
Fecha: 2023.01.27 13:26:24 +01'00'

La directora:
Doña Marta Sestelo Pérez

Firmado por PEREZ VILARELLE LAURA - ***6107** el día 27/01/2023 con un certificado emitido por AC FNMT Usuarios

La tutora:
Doña Laura Pérez Vilarelle

El autor:
Don Julio José Rodríguez Martínez

Declaración responsable. Para dar cumplimiento a la Ley 3/2022, de 24 de febrero, de convivencia universitaria, referente al plagio en el Trabajo Fin de Máster (Artículo 11, Disposición 2978 del BOE núm. 48 de 2022), **el/la autor/a declara** que el Trabajo Fin de Máster presentado es un documento original en el que se han tenido en cuenta las siguientes consideraciones relativas al uso de material de apoyo desarrollado por otros/as autores/as:

- Todas las fuentes usadas para la elaboración de este trabajo han sido citadas convenientemente (libros, artículos, apuntes de profesorado, páginas web, programas,. . .)
- Cualquier contenido copiado o traducido textualmente se ha puesto entre comillas, citando su procedencia.
- Se ha hecho constar explícitamente cuando un capítulo, sección, demostración,. . . sea una adaptación casi literal de alguna fuente existente.

Y, acepta que, si se demostrara lo contrario, se le apliquen las medidas disciplinarias que correspondan.

Índice general

Resumen en español

Este trabajo consiste en determinar la explotabilidad en 3, 6 y 12 meses de una vulnerabilidad a partir de la publicación de un CVE.

En el análisis de la respuesta se ha detectado una gran diferencia entre el número de registros de los CVE's explotados y los que no, lo que ha obligado a aplicar técnicas de balanceo, así como la consideración de otras métricas que puedan distinguir mejor el rendimiento de los modelos y una modificación en el umbral de clasificación.

Para determinar que variables incluir en los modelos se han usado tanto métodos gráficos, como diagramas de violín, como analíticos, como la selección por correlaciones, métodos de regularización y criterios automáticos. Además, la ventana temporal se ha escogido buscando potenciar las variables de las redes sociales.

Los modelos que han sido probados han sido regresión logística, GAM y *Random Forest*, de los cuales el que ha obtenido mejores resultados ha sido este último.

English abstract

This academic paper consists in determining the exploitability in 3, 6 and 12 months of a vulnerability after the publication of a CVE.

In the analysis of the response a huge difference has been detected between the number of exploited CVE's and those that are not, forcing the application of data balancing techniques, as well as the consideration of other metrics that can better distinguish the performance of the models and a modification in the classification threshold.

To determine which variables are interesting to include in the models, graphic methods are considered, such as violin diagrams. There are also considered analytical methods , such as correlations selection, regularization methods and automatic criteria. In addition, the window gap considered has been choosed to enhance the variables of social networks.

The models that have been tested are logistic regression, GAM and *Random Forest*. Making a comparative of the results, the *Random Forest* is considered the best of the three models.

Capítulo 1

Introducción

En este capítulo se tratarán los siguientes temas: una breve descripción de Gradiant, empresa en la cual se ha desarrollado el trabajo, así como una una explicación de la motivación del desarrollo del mismo, y del estado del arte sobre el problema planteado.

1.1. Gradiant

Gradiant es un centro tecnológico privado creado en 2008, que se estructura en una fundación con un patronato mixto que integra a las tres universidades gallegas, a las compañías más importantes que operan en Galicia y a la Asociación Empresarial INEO, que a su vez agrupa a un gran número de empresas gallegas de dicho ámbito.

La misión de Gradiant se resume en "contribuir al dinamismo innovador, el crecimiento y la mejora competitiva del tejido empresarial gallego a través del desarrollo tecnológico y la innovación en el uso de las Tecnologías de la Información y la Comunicación (TIC)".

Su filosofía es ayudar a las empresas a generar negocio integrando la transferencia de conocimiento y diseñando soluciones especializadas para la industria, sumando el valor añadido de los profesionales

del Centro. Gradiant es el socio TIC clave para que las empresas puedan generar beneficios, innovando y diseñando productos "a la carta"para cada cliente.

1.2. Motivación del problema

A día de hoy, la mayor parte de las soluciones TIC necesitan integrarse o colaborar con otros componentes, típicamente desarrollados por terceros. Este tipo de procedimientos, a pesar de que resultan clave a la hora de mantener la productividad y competitividad en las empresas, también pueden suponer un gran riesgo, ya que en la mayor parte de los casos no hay manera de verificar si tienen vulnerabilidades o si han sido desarrolladas siguiendo las mejores prácticas de seguridad. Una vulnerabilidad es una debilidad que puede explotarse en un ciberataque para obtener acceso no autorizado o realizar acciones no autorizadas en un sistema informático. Estas son recogidas en el glosario de vulnerabilidades y exposiciones comunes (CVE), que es un proyecto de seguridad centrado en *software* de lanzamiento público, financiado por la División de Seguridad Nacional de EE.UU y mantenido por MITRE Corporation.

Para enfrentarse a este reto, es importante que las empresas cambien su mentalidad. Según estudios recientes[1], sólo un 29 % de los negocios TIC son conscientes que sus socios cumplan los niveles de seguridad esperados. Sin embargo, dado que los ciberataques pueden tener un alto impacto, no se puede depender exclusivamente de la confianza en que estos no sucedan. Los componentes TIC deben poder proporcionar garantías verificables en cuanto a sus propiedades de seguridad y privacidad. Además, es importante detectar de forma más precisa las vulnerabilidades en estos componentes, y entender cómo dichas vulnerabilidades se pueden propagar a través de la cadena de valor de los ecosistemas TIC. Sin embargo, la mayor parte de las vulnerabilidades pueden permanecer ocultas durante años, por lo que es necesario proporcionar herramientas avanzadas que garanticen la resiliencia y mejorar las estrategias de mitigación.

Un *exploit* es un programa informático, una parte de un *software* o una secuencia de comandos que se aprovecha de un error o vulnerabilidad para provocar un comportamiento no intencionado o imprevisto en un *software*, *hardware* o en cualquier dispositivo electrónico. Este trabajo formará parte del proyecto *Building Trust in Ecosystems and Ecosystem Components* (BIECO), un *framework* holístico financiado por la Unión Europea que será desarrollado hasta Agosto de 2023. Su objetivo será predecir la existencia de un *exploit* a partir de la publicación de un CVE y se estructura en cuatro capítulos:

- Capítulo 2: Análisis descriptivo, donde se ha explicado la naturaleza de las distintas variables y se realizará un estudio de la variable respuesta y de los predictores.
- Capítulo 3: Modelos utilizados, en el cual se habla de los distintos modelos que se emplean.
- Capítulo 4: Desarrollo del trabajo, donde se ha descrito como se llevó a cabo todos los conceptos vistos en los dos capítulos anteriores.
- Capítulo 5: Conclusiones y trabajo futuro.

1.3. Estado del arte

En la literatura se han descrito algunas contribuciones, como Adjerid et al (2019) y Anand et al (2020), que estudian si un CVE se usa en un *exploit*. Una pregunta importante que crece entre los investigadores es pronosticar cuándo podría aparecer un *exploit*. Este problema es especialmente importante, debido a la escasez de recursos para tomar acciones correctivas cuando aparece una nueva vulnerabilidad, como vemos en Chen et al (2019).

[1] Presentados en Bieco Deliverable D3.3 "Report of the Tools for Vulnerability Detection and Forecasting"

Cuando se divulga información de una vulnerabilidad, existe la posibilidad de que un atacante la explote y, en consecuencia, una tendencia de los proveedores a lanzar parches. Debido a que parchear es costoso, muchas vulnerabilidades quedan sin parchear debido a la limitada cantidad de recursos. Por lo tanto, es fundamental priorizar la aplicación de parches en función de los resultados de los modelos de predicción sobre cuándo se explotará una vulnerabilidad.

Cabe destacar que existen dos tipos de *exploits*:

- *Exploits* de prueba de conceptos (PoC, proveniente del inglés *proof of concept*), que son aquellos en los que alguien genera un código de explotación de muestra para demostrar la vulnerabilidad en un entorno controlado.
- *Exploits* que se utilizan en ataques reales.

Publicaciones como las de Bozorgi et al (2010) y Almukaynizi et al (2017) se centran en la predicción de *exploits* del mundo real. Dumitras et al (2015) desarrollaron métodos para predecir tanto PoC como del mundo real, siendo los primeros en demostrar que la precisión de la predicción se podía mejorar mediante el uso de datos de Twitter (más concretamente, de los datos de un año).

Recientemente, se han realizado muchos más esfuerzos para estudiar casos reales donde, además de la *National Vulnerability Database* (NVD) y el *Exploit Database Archive* (EDB), se han utilizado los datos de *Zero Day Initiative* (ZDI) y publicaciones en la *dark weeb* y la *deep web*.

Por otro lado, *Common Vulnerability Scoring System* (CVSS), es un *framework* abierto y universalmente utilizado que establece unas métricas para la comunicación de las características, impacto y severidad de vulnerabilidades que afectan a elementos del entorno de seguridad IT. Aunque este se ha convertido en un estándar de la industria para evaluar las características fundamentales de las vulnerabilidades, han sido identificadas algunas limitaciones, como la ausencia de una entidad autorizada para actualizar los valores métricos y falta de datos para informar la puntuación. Siguiendo esta idea, Chen et al (2019) describieron un método para predecir cuándo se explotará una vulnerabilidad en función de la identificación del CVE y de los datos de discusión de Twitter, sin la necesidad del CVSS. Además, Adjerid et al (2019) propusieron un primer sistema abierto de puntuación de amenazas basado en datos, llamado *Exploit Prediction Scoring* (EPSS), para predecir la probabilidad de que una vulnerabilidad sea explotada dentro de los 12 meses siguientes a la divulgación pública.

Teniendo en cuenta el creciente número de vulnerabilidades y su consiguiente necesidad de priorizarlas, el objetivo en BIECO será desarrollar una herramienta que permita predecir cuando una determinada vulnerabilidad será explotada. Siguiendo las ideas propuestas por Dumitras y Adjerid, entre otras, se probarán diferentes características o variables como entrada para entrenar tres modelos de *Machine Learning* (ML) que actuarán sobre tres franjas temporales distintas: 3, 6 y 12 meses posteriores a la publicación del CVE. Estos se activarán secuencialmente: si la predicción para el modelo de 3 meses es negativa, se recurrirá al de 6 meses, y si ésta siguiera siéndolo, se procede a utilizar el de 12, dando como salida la explotabilidad (o no) del CVE.

Capítulo 2

Análisis Descriptivo

En este capítulo se describe el conjunto de datos que ha sido utilizado a lo largo de este trabajo. Se ha comenzado presentando las variables que lo conforman, así como el problema de desbalanceo que surge asociado a la variable respuesta. Además, se ha justificado la necesidad de considerar dos períodos de tiempo distintos a la hora de recoger los registros, y se han comentado las distintas selecciones de variables tenidas en cuenta.

2.1. Descripción de las variables.

Las variables utilizadas en este trabajo han sido motivadas por el estado del arte y por un análisis de distintos expertos. Estas han sido obtenidas de diversas fuentes:

- *Data Collection Tool* (DCT)[1], herramienta desarrollada por BIECO que recoge detalles descriptivos de la vulnerabilidad o medidas de impacto y detalles de su publicación como pueden ser la fecha, una breve descripción de la misma o el CVSS.

- *Exploit DB*[2], una base de datos de *exploits* que tienen por fin mejorar la seguridad pública, proporciona la información relativa a la explotabilidad de las vulnerabilidades, necesaria para el entrenamiento de los modelos supervisados.

- API de Twitter[3] y diversos chats de Telegram[4], pretendiendo captar la actividad, presencia o impacto de la aparición de la vulnerabilidad en la sociedad y agregar información relativa a la posible inmediatez de su explotabilidad.

Tras la extracción de las mismas, se han obtenido una totalidad de 90 variables, que pueden ser agrupadas de la siguiente forma:

- Variables relacionadas con el CVE: fecha de publicación del CVE, *Common Weakness Enumeration* (CWE) al que pertenece, un sistema de categorías para las debilidades y vulnerabilidades del software, y nombre del mismo.

- Variables relacionadas con las métricas CVSS: dos puntuaciones (v2 y v3, que provienen información muy similar, siendo esta última más actual), donde se distinguirán tres *scores*: *Base*, *Exploitability* e *Impact Score*, que aportan una información general sobre el CVE, así como sobre la explotabilidad e impacto del mismo.

[1] https://dct.bieco.org/
[2] https://www.exploit-db.com/
[3] https://developer.twitter.com/en
[4] https://web.telegram.org/z/

- Variables *tags*, *products* y *vendors*: los *tags* representativos obtenidos en las descripciones publicadas para cada vulnerabilidad han sido empleados como variables dicotómicas a evaluar para cada vulnerabilidad. Con idéntico criterio, se obtiene una colección finita y representativa de *vendors* y *products* para evaluar en cada vulnerabilidad, que han sido asociados a los productos que puedan contener una vulnerabilidad.

- Variables relacionadas con las redes sociales (RRSS): estas serán obtenidas de Twitter y Telegram. De la primera red social se han obtenido el número de *tweets*, *retweets*, *likes*, citas y respuestas, buscando recoger la mayor información sobre el impacto que pueda tener la publicación de un CVE en esta red social, mientras que de la segunda únicamente el número de mensajes. En ambas redes sociales se han considerado los registros comprendidos hasta una semana posterior a la publicación del CVE.

2.2. Análisis de la respuesta: El problema del balanceo

Para comenzar el análisis descriptivo, se ha realizado un diagrama de sectores para ver como se clasifican las distintas observaciones de la variable respuesta. En este primer análisis solamente se distinguen dos clases: "explotado" o "no explotado", sin hacer referencia a ninguna franja temporal concreta.

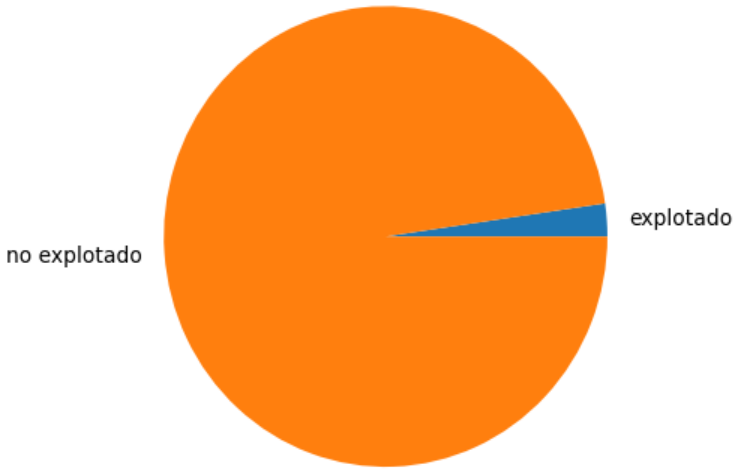


Figura 2.1: Diagrama de sectores sobre las observaciones de la variable respuesta.

A la vista del diagrama de sectores (Figura 2.1), se observa un claro desbalanceo entre el número de observaciones de las dos clases (al solo encontrar una cifra cercana al 3 % en los registros de los CVEs explotados). Debido a esto, los modelos de ML tenderán a clasificar la mayoría de las nuevas observaciones por la clase mayoritaria (CVE's no explotados), ignorando la clase minoritaria (CVE's explotados), lo que resulta un gran problema.

Por lo tanto, se han probado distintas alternativas para atajar estas dificultades, como métodos de balanceo, la consideración de métricas que funcionen de manera óptima frente a conjuntos de datos desbalanceados y variar el umbral que delimita la clasificación para intentar corregir los problemas causados por el desbalanceo de las clases.

2.2.1. Métodos de balanceo

Una primera alternativa para resolver el problema del desbalanceo consiste en "nivelar" la cantidad de observaciones de cada clase. Estos procedimientos son conocidos como técnicas de balanceo.

A continuación, se describen dos opciones de balanceo: una de sobremuestreo, en la que se crean muestras artificiales de la clase minoritaria, y otra de submuestreo, en la que se eliminan muestras de la mayoritaria.

Método SMOTE

El método *Synthetic Minority Oversampling Technique* (SMOTE) (Wang et al 2006) es una técnica de sobremuestreo que aumenta el número de muestras de una clase minoritaria en la proporción pedida.

Las muestras artificiales son generadas mediante el algoritmo de los k vecinos: se selecciona una observación aleatoria y, a través de sus k-vecinos más cercanos, se genera, mediante interpolación, una nueva muestra sintética.

En la Figura 2.2, se puede ver la nube de puntos con la que se ejemplifica el uso de los métodos de balanceo. En las Figuras 2.2 y 2.3 los registros asociados a la clase minoritaria están representados de color azul, mientras que los de la mayoritaria de color rojo, poniéndose de manifiesto la gran diferencia entre las proporciones de ambas clase (Figura 2.2) y el funcionamiento del método SMOTE (Figura 2.3).

Tras la aplicación de este método se puede observar como la mayoría de puntos generados artificialmente se encuentran muy próximos, consecuencia de haber sido generados mediante la interpolación de los k-vecinos.

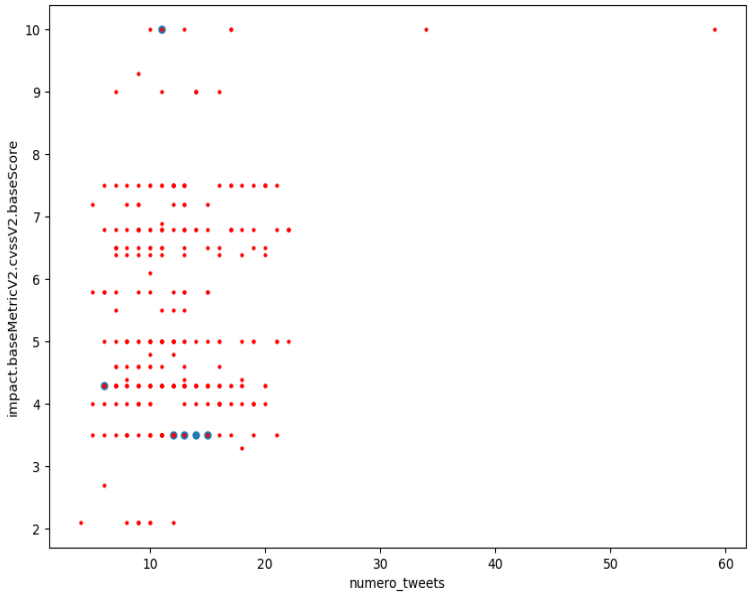


Figura 2.2: Nube de puntos para ejemplificar el funcionamiento de los métodos de balanceo

Método *NearMiss*

El método *Near Miss* (Adeliyi 2021) es un método de submuestreo que elimina las observaciones de la clase mayoritaria que más se parezcan a los de la clase minoritaria. Para ello, este algoritmo selecciona los k-vecinos más cercanos para cada muestra de la clase minoritaria y los elimina, manteniendo las observaciones de la clase mayoritaria más diferentes a las de la clase minoritaria.

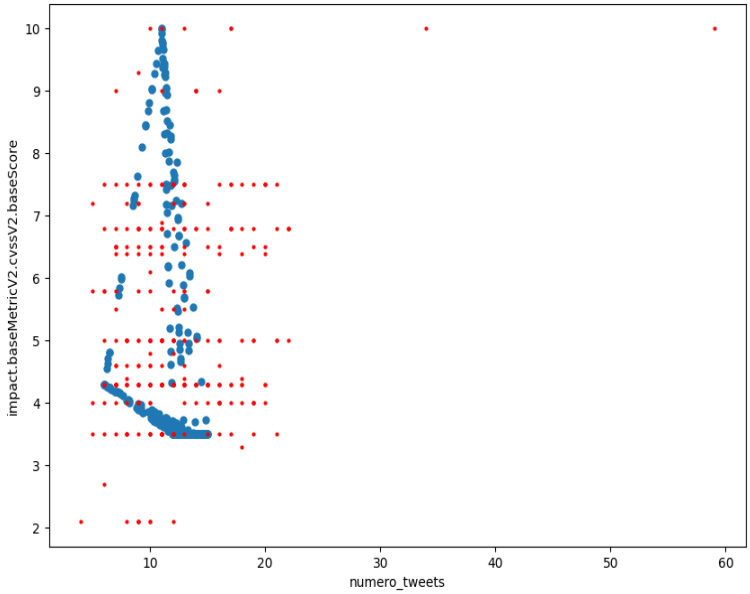


Figura 2.3: Nube de puntos tras aplicar el método SMOTE.

En la Figura 2.4 se puede observar el funcionamiento del *Near Miss* utilizando el mismo conjunto de datos que en el caso anterior. Como se comentó anteriomente, los puntos eliminados de la clase mayoritaria son los que más se parecen a la otra clase. De todas formas, al haber tal grado de desbalanceo, se puede ver como aun se están conservando puntos que toman los mismos valores para las dos variables (por ejemplo, las observaciones con 12 *tweets* y un valor de la métrica de 3.5). Al igual que pasaba con las Figuras 2.2 y 2.3, en la Figura 2.4 los registros asociados a la clase minoritaria están representados de color azul, mientras que los de la mayoritaria de color rojo.

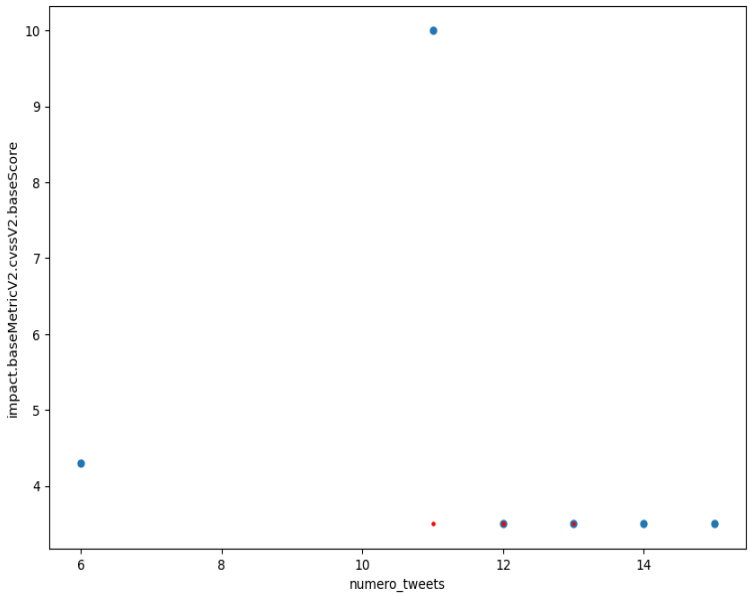


Figura 2.4: Nube de puntos tras aplicar el método *Near Miss*.

2.2.2. Evaluación de modelos

A continuación, se presentan las métricas más habituales utilizadas en la comparación del rendimiento de los modelos, y, motivado por el desbalanceo de las clases, se han buscado otras métricas alternativas que puedan ser de mayor utilidad.

Métricas usuales

En problemas de clasificación binaria, se codifica la pertenencia a las dos clases de la siguiente manera:

- 0: pertenencia a la clase negativa, asociada a la no explotabilidad del CVE.
- 1: pertenencia a la clase positiva, asociada a la explotabilidad del mismo.

A partir de esta codificación, surgen las siguientes definiciones:

- *True Positive* (TP): número de positivos acertados en la predicción.
- *True Negative* (TN): número de negativos acertados en la predicción.
- *False Negative* (FN): número de negativos predichos que son positivos.
- *False Positive* (FP): número de positivos predichos que son negativos.

Teniendo en cuenta estos últimos conceptos, se pueden definir algunas métricas que sean de ayuda en la evaluación del rendimiento de los modelos. Las más comunes son:

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} = \text{proporción de aciertos totales.}$$
$$\text{Precisión} = \frac{TP}{TP + FP} = \text{proporción de positivos predecidos acertados.}$$
$$\text{Sensibilidad} = \frac{TP}{TP + FN} = \text{proporción de positivos reales acertados.}$$
$$\text{Especifidad} = \frac{TN}{TN + FP} = \text{proporción de negativos predecidos acertados.}$$

De entre todas estas métricas, la más utilizada es la Exactitud, ya que aporta una visión general de cuanto acierta cada modelo a la hora de realizar predicciones.

Esta medida no va a ser la más adecuada a utilizar en este contexto, ya que este valor aporta una visión general de la proporción de aciertos, sin hacer ninguna distinción entre el acierto producido en cada una de las dos clases. Por lo tanto, clasificando la mayoría de observaciones como la clase mayoritaria, este valor sería muy alto, clasificando incorrectamente gran proporción de las observaciones de la clase minoritaria, lo que va en contra de los intereses de este trabajo. Consecuentemente, es necesario buscar otras métricas que favorezcan la clasificación correcta de la clase minoritaria.

Métricas alternativas

Consultando la literatura (Cohn 2013), dos métricas interesantes serán el *F1-Score* y el AUC (*Area Under the roc Curve*), que pueden funcionar mejor con datos desbalanceados.

El *F1-Score* se define de la siguiente manera:

$$\textit{F1-Score} = \frac{2 \cdot \text{Precisión} \cdot \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}},$$

que no es más que la media harmónica entre la Precisión y Sensibilidad. Para que el valor de esta métrica sea alta, ambos valores han de ser altos.

Pese a ser una buena alternativa, esta medida aún se puede ver afectada por el desbalanceo de las clases. Alternativamente, y como se puede ver en el artículo anteriormente citado, la métrica más útil en estos casos será la del AUC.

La curva *Receiver Operating Characteristic* (ROC) (Mauri L. et al 2007) es una representación bidimensional que muestra la evolución de la 1-Especifidad frente a la Sensibilidad, para los distintos valores del *threshold* (valor que sirve como umbral para delimitar la clasificación de las dos clases, el cual se detallará más profundamente la subsección 2.2.3).

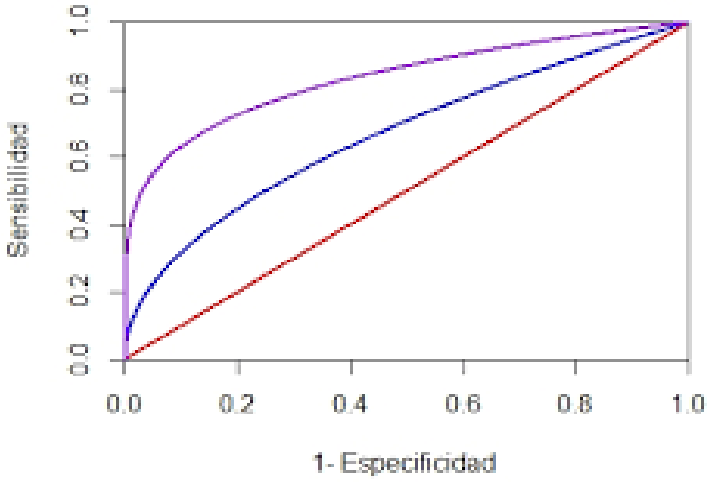


Figura 2.5: Visualización de la curva ROC.

Observando la Figura 2.5, y teniendo en cuenta que TNR=1-Especifidad, se está representando el *True Negative Rate* (TNR, la proporción de negativos acertados) en el eje X frente al *True Positive Rate* (TPR, proporción de negativos acertados) en el eje Y.

Intuitivamente, una nueva métrica podría aparecer motivada por la busqueda de que la curva estuviese lo más arriba a la izquierda posible de la representación, obteniendo valores altos de ambas métricas. A partir de esta idea es como surge el concepto del AUC, que computa el área bajo la curva ROC.

El AUC es un valor comprendido entre 0 y 1 que computa el área comprendida por debajo de la curva ROC, y que proporciona la probabilidad de que el modelo sea capaz de distinguir entre las dos clases. Por ejemplo: si el valor del AUC es de 0.6, el modelo tiene un 60 % de posibilidades de distinguir entre las dos clases.

Por lo tanto, uno de los criterios utilizados a lo largo de este trabajo para la comparación del rendimiento de los modelos y en la elección de los hiperparámetros será el AUC.

2.2.3. Elección del *threshold*.

En esta última parte se describe un concepto que juega un papel clave en los problemas de clasificación: el *threshold*. Esto puede ser visto como otra solución para los conjuntos de datos desbalanceados, ya que ajustar este valor adecuadamente puede ser de ayuda para obtener una clasificación más equilibrada entre las dos clases.

A la hora de realizar la clasificación, los modelos devuelven una probabilidad y, a partir de establecer un umbral, conocido como *threshold*, las nuevas observaciones se clasificarán como pertenecientes a una

clase u otra. Por defecto, muchos modelos utilizarán un valor de 0.5 como *threshold*. Esto carece de sentido, ya que este puede variar en función de cómo está distribuída nuestra variable respuesta.

Por lo tanto, se necesitará algún criterio para la selección de este valor. Algunos de los estadísticos más utilizados para la selección automática del valor del *threshold* son:

$$\begin{aligned} \text{Estadístico G-means} : G &= \sqrt{\text{Sensibilidad} \cdot \text{Especifidad}} \\ \text{Estadístico J de Youden} : J &= TPR - FPR. \end{aligned}$$

El primero busca maximizar los valores altos de Sensibilidad y Especifidad, buscando un equilibrio entre ellos, mientras que el segundo busca maximizar la proporción de positivos predichos acertados.

En base a la opinión de expertos, se concluye que el interés está centrado en optimizar la proporción de aciertos de la clase minoritaria. Para ello, se ha creado el siguiente estadístico, que busca minimizar la distancia entre el TPR y el TNR:

$$\text{Estadistico} : E = |TNR - TPR|$$

La elección de este estadístico se sustenta en los siguientes factores:

- Los valores de TPR y TNR son inversamente proporcionales según cómo varíe el valor del *threshold*. Por ejemplo: si este aumenta, también lo harán el número de observaciones clasificadas como negativas, y, consecuentemente, el TNR aumentará o, en caso de que todas las nuevas clasificaciones fueran incorrectas, permanecerá constante. De la misma manera, el TPR disminuirá o permanecerá constante.

- Empíricamente, se ha comprobado que una vez el valor del TPR supera al del TNR este primero aumenta muy lentamente mientras que el segundo desciende de forma mucho más rápida, debido a la gran diferencia entre el número de observaciones de las dos clases.

En base a esto, se ha establecido el estadístico E como medida para obtener el mejor valor del *threshold*, ya que, además de obtener una proporción alta de aciertos en la clase minoritaria, también mantiene un valor alto en el acierto de la clase mayoritaria.

2.3. Análisis de los predictores

Para el análisis de los predictores se han utilizado diagramas de violín. Estos nos aportan una herramienta para visualizar la distribución de los datos y su densidad de probabilidad, mediante una combinación de un diagrama de caja y un diagrama de densidad girado y colocado a cada lado, para mostrar la forma de distribución de los datos de cada clase.

Los elementos más destacados de los diagramas de violín son una barra negra gruesa en el centro, que representa el rango intercuartílico, una fina línea que se extiende desde ella, que representa una extensión de este intervalo hasta abarcar el 95 % de los datos ordenados, y un punto blanco que representa la mediana.

Debido al gran número de variables, se muestra únicamente el diagrama de violín para la variable *impact.baseMetricV2.impactScore*, que mide el impacto que puede tener una vulnerabilidad.

En la Figura 2.6 se puede comparar las densidades de ambas clases, llegando a la conclusión de que para valores bajos de esta variable hay mucha mayor probabilidad de que pertenezca a la clase negativa. Por esto, se puede sospechar que la variable *impact.baseMetricV2.impactScore* nos va a permitir diferenciar entre las dos clases, y será de utilidad incluirla en el modelo. Con un procedimiento análogo se puede hipotetizar sobre la importancia que tendrán las distintas variables en nuestros modelos.

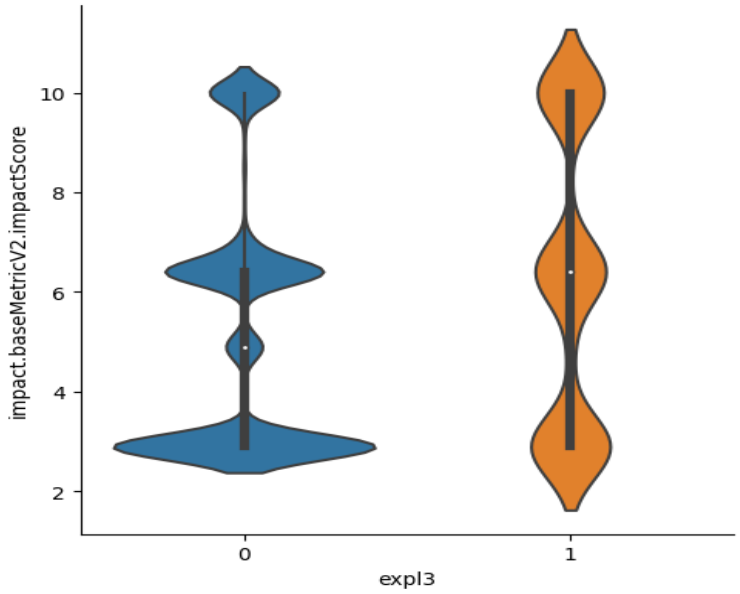


Figura 2.6: Diagrama de violín de la variable *impact.baseMetricV2.impactScore*.

2.4. Elección de la ventana temporal

Para entrenar los modelos se dispone de un *dataset* que incluye las vulnerabilidades publicadas entre 2008 y 2021 y recogidas en DCT, como ya se comentó anteriormente.

En la realización de este estudio se han contemplado distintas ventanas temporales, buscando potenciar el impacto de las variables relacionadas con las RRSS en nuestros modelos. A la hora de realizar la selección de estas franjas se han tenido en cuenta dos factores:

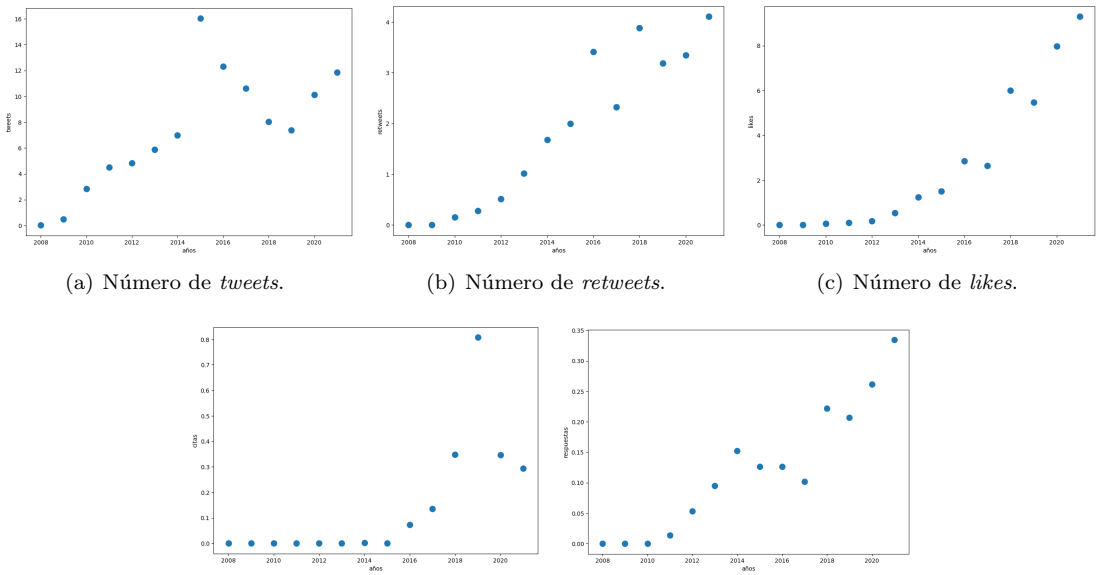
- Para que los métodos de balanceo funcionen correctamente es necesario tener un número mínimo de registros de la clase minoritaria.

- Cuanto más reciente sea la franja seleccionada, mayor importancia tendrán las variables asociadas a las RRSS, debido al incremento de la actividad de las mismas en los últimos años.

En la Figura 2.7 se aprecia el comportamiento en el número de *tweets* 2.7(a), *retweets* 2.7(b), *likes* 2.7(c), citas 2.7(d) y respuestas 2.7(e). En base a esto, se han contemplado únicamente dos franjas temporales distintas: la primera, desde 2008, incluye la totalidad de los registros, mientras que en la segunda, motivada por darle una mayor importancia a las variables de las RRSS (en particular de Twitter), se han reducido el número de registros, al considerar unicamente los datos comprendidos desde 2015, período desde el cual se produjo una mayor actividad de esta red social.

2.5. Métodos de selección de variables

En esta subsección se exponen los métodos de selección de variables que han sido utilizados durante este estudio. El primero (selección por correlaciones) y los terceros (métodos de regularización) surgen debido a la presencia de colinealidad entre las variables, mientras que el segundo (selección por criterios automáticos) se ha realizado buscando optimizar los resultados de los modelos utilizados.



(a) Número de *tweets*. (b) Número de *retweets*. (c) Número de *likes*.

(d) Número de citas. (e) Número de respuestas.

Figura 2.7: Evolución de los registros de las variables de Twitter.

2.5.1. Selección por correlaciones.

La selección de variables basada en las correlaciones consiste en, para cada par de predictores altamente correlados, eliminar el que presente una correlación más baja con la respuesta. Para ello, se ha utilizado el coeficiente de correlación de Spearman (Daniel 1995), el coeficiente de correlación biserial (John 2008) y el Tau-c de Kendall (Daniel 1995), según la naturaleza de las variables (continua-continua, continua-binaria y binaria-binaria, respectivamente).

El coeficiente de correlación de Spearman sirve para para medir la relación entre dos variables continuas. Este surge como una alternativa al coeficiente de correlación de Pearson cuando se incumple alguna de las suposiciones en las que se basa, como la normalidad de los datos o la relación lineal. Se define de la siguiente manera:

$$\rho = 1 - \frac{6\sum_i d_i^2}{n(n^2-1)},$$

siendo:

- n: número de datos de cada variable.
- d_i: diferencia de rango en el elemento n-ésimo.

El valor de este coeficiente oscilará entre -1 y 1, siendo más cercano a los extremos cuanto más fuerte sea la dependencia entre las dos variables. Además, el signo determina si la relación entre ambas variables es directa (positivo) o inversa (negativo).

El coeficiente de correlación biserial es un caso particular del coeficiente de correlación de Pearson, que se utiliza para medir la relación entre una variable continua y una binaria. Este, denotado por r_{pb}, se puede definir como:

$$r_{pb} = \frac{M_1 - M_0}{s_n}\sqrt{pq},$$

siendo:

- $M_1 = \mathbf{E}(X|Y=1)$, la media de la variable continua X para la clase positiva.

- $M_0 = \mathbf{E}(X|Y = 0)$, la media de X para la clase negativa.
- $s_n = SD(Y)$, la desviación típica de la respuesta.
- $p = \frac{1_{Y=1}}{n}$, siendo n el número total de observaciones, representa la proporción de la clase positiva.
- $q = \frac{1_{Y=0}}{n}$, representa la proporción de la clase negativa.

El coeficiente biserial toma valores entre 0 y 1. Cuanto más se aproxime a 1, mayor será la relación entre ambas variables.

Para la correlación entre dos variables binarias usaremos el Tau-c de Kendall. Ante la posibilidad de empates, y teniendo en cuenta que la tabla de contingencia asociada es de tipo rectangular, usaremos el Tau-c de kendall, τ_c, que se define como:

$$\tau_c = \frac{2(n_c - n_d)}{n^2 \, \dfrac{(m-1)}{m}},$$

siendo:

- n_c = número de pares concordantes.
- n_d = número de pares discordantes.
- m = mín(número de categorías, número de variables).

Al igual que con el coeficiente biserial, la interpretación será análoga: cuanto más se aproxime al 1, mayor será la relación entre ambas variables.

2.5.2. Selección por criterios automáticos.

La selección por criterios automáticos se basa en la eliminación recursiva de variables mediante la función del módulo sklearn.feature_selection (Buitinck et al 2013) **RFECV()** (*Recursive feature elimination with cross-validation to select features*).

La función **RFECV**() realiza la selección de variables considerando conjuntos de variables cada vez menores. Partiendo de la totalidad de las variables, y mediante el AUC como métrica para comparar el rendimiento de los modelos, las variables son eliminadas hasta que este valor no se mejore reduciendo el número de variables.

2.5.3. Métodos de regularización

Estos últimos métodos de selección de variables, conocidos como métodos de regularización (Hastie et al 2013), consisten en penalizar los coeficientes del modelo de forma que estos tomen valores cercanos a cero, buscando reducir la varianza del modelo y arreglar posibles problemas de colinealidad.

Por lo tanto, a la hora de estimar los parámetros será interesante buscar un criterio que busque que el número de variables utilizadas en el modelo sea menor. Una idea intuitiva es tener en cuenta, a la hora de realizar la estimación de los parámetros, una penalización que haga aumentar el valor del RSS (*Residual Sum of Squares*) según el número de variables que se estén utilizando en el modelo. Por sencillez, se verá como aplicar esta idea en la estimación de los coeficientes en los modelos lineales. Esto se conoce como mínimos cuadrados penalizados, que consiste en minimizar la siguiente cantidad:

$$\hat{\beta} = \min_{\beta} \sum_{i=1}^{n} (Y_i - \beta_i x_i)^2 + \lambda P,$$

donde el primer sumando penaliza la diferencia entre la estimación y los valores reales, y el segundo penaliza el número de variables incluídas en el modelo, siendo λ el parámetro de penalización y P una función que penalizará el número de variables. En este trabajo se han utilizado tres penalizaciones distintas, que han sido descritas a continuación.

Penalización *Ridge*

La penalización *Ridge* utiliza la siguiente función de penalización:

$$P = ||\beta_{(0)}||_2,$$

siendo $\beta_{(0)}$ el vector de parámetros quitando la ordenada en el origen y $||\beta_{(0)}||_2$ la norma en L_2, obtenida mediante la raíz cuadrada de la suma de los cuadrados de los elementos del vector.

De esta manera, la penalización *Ridge* consiste en minimizar la siguiente cantidad:

$$\hat{\beta} = \min_{\beta} \sum_{i=1}^{n} (Y_i - beta_i x_i)^2 + \lambda ||\beta_{(0)}||_2.$$

Penalización *LASSO*

Por su parte, la penalización *LASSO* busca que una gran cantidad de explicativas valgan cero. En este caso, consideraremos la siguiente penalización:

$$P = ||\beta_{(0)}||_1$$

siendo $||\beta_{(0)}||_1$la norma en L_1, que consiste en la suma de los valores absolutos de los elementos del vector de parámetros.

Por lo tanto, se obtiene la siguiente expresión a minimizar:

$$\hat{\beta} = \min_{\beta} \sum_{i=1}^{n} (Y_i - \beta_i x_i)^2 + \lambda ||\beta_{(0)}||_1$$

Como se ha visto anteriormente, ambas penalizaciones tienen puntos a favor, por lo que es algo natural pensar en utilizar una combinación de ambas buscando aprovechar las virtudes de las dos. Es así como nace el último método de regularización.

Penalización *Elastic Net*

Como se ha comentado anteriormente, esta última regularización combina el efecto de las dos anteriores. Por lo tanto, tenemos que:

$$P = \alpha ||\beta_{(0)}||_1 + (1 - \alpha) ||\beta_{(0)}||_2,$$

siendo α un valor entre 0 y 1. En el caso en el que $\alpha = 0$ estaríamos en el caso de penalización *Ridge*, y si $\alpha = 1$ en penalización *LASSO*.

Así, la expresión que se minimiza para realizar el ajuste es la siguiente:

$$\hat{\beta} = \min_{\beta} \sum_{i=1}^{n} (Y_i - \beta_i x_i)^2 + \lambda(\alpha ||\beta_{(0)}||_1 + (1 - \alpha) ||\beta_{(0)}||_2).$$

Capítulo 3

Modelos utilizados

En esta sección se describen los modelos más importantes que han sido utilizados en este trabajo: regresión logística, *Generalized Aditive Models* (GAM) y *Random Forest*. Además, se ha probado otros modelos populares como *Support Vector Machine* (SVM) y redes neuronales.

Para la implementación de estos modelos han sido utilizadas las librerías *pyGam* (Brummit et al 2018), *sklearn* (Buitinck et al 2013) e *imbalanced learn* (Aridas et al 2016).

3.1. *Generalized Aditive Models*

Para describir el GAM se ha desarrollado una evolución partiendo desde un modelo muy simple, el modelo lineal ordinario, a partir de generalizaciones y flexibilizaciones, llegando hasta el GAM, como se verá a continuación.

Modelo lineal

El primero de los modelos descritos en esta sección será el modelo lineal ordinario (Hastie et al 2013). Este es el más sencillo de todos, y establece una relación proporcional entre los predictores y la respuesta. Se puede formular de la siguiente manera:

$$Y_i = x_i\beta + \epsilon_i \text{ para } i \in \{1, ..., n\}$$

Asociada a esta formulación hay una serie de hipótesis básicas que se han de cumplir para que este modelo sea válido:

- Linealidad: la relación entre la variable respuesta y los predictores va a ser lineal. Esto implica que si el valor de uno de los predictores aumenta, también lo hará la respuesta.
- Los errores son independientes y siguen una distribución normal, con media nula y varianza constante.

Una de las consecuencias de la formulación del modelo junto con la hipótesis de que los errores sigan una distribución normal es que la variable respuesta ha de seguir una distribución normal también, lo cual es bastante restrictivo. Para poder relajar esta suposición surgen los *Generalized Lineal Models* (GLM) (Hastie et al 2013) , que permiten que la variable respuesta no siga una distribución normal.

GLM

Como se ha comentado anteriormente, los GLM permiten evitar la suposición de que la variable respuesta siga una distribución normal. La principal diferencia entre ambos modelos reside en la introducción de una función de enlace (o *link*), g, que relacione la esperanza condicional de la respuesta

con los predictores:

$$g(E(Y|X)) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p,$$

donde dicha función tiene que ser invertible, para poder obtener los valores de la esperanza condicional de la siguiente manera:

$$E(Y|X) = g^{-1}(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p).$$

Para la estimación del vector de parámetros se utiliza un procedimiento de máxima verosimilitud, que consiste en escoger como valor estimado del parámetro aquel que tiene mayor probabilidad de ocurrir según lo que se ha observado.

Sea $\eta = g(\mu)$, se puede definir la verosimilitud como:

$$\sum_{i=1}^{n} x_{ij} \frac{\partial \mu_i}{\partial \eta_i} V_i^{-1}(y_i - \mu_i) = 0, \qquad j = 0, 1, ..., p,$$

donde $V_i = var(Y_i)$.

Para resolver estas ecuaciones se usa el algoritmo *Iteratively Reweighted Least Squares* (IRLS), que es un método iterativo que en cada paso se realiza la resolución de un problema de mínimos cuadrados ponderados.

Regresión logística

Un caso particular de los GLM es el de la regresión logística (Hastie et al 2013), en el que la respuesta tiene que ser binaria. La idea de este modelo es la predicción de probabilidades y, a partir de ellas, poder clasificar las nuevas observaciones como 0 ó 1 a partir de un *threshold*.

Por lo tanto, se necesitará construir un modelo que determine la posibilidad de éxito, denotada como $\pi(x)$:

$$\pi(x) = P(Y_i = 1|X = x).$$

Para poder considerar un modelo con una relación entre la respuesta y los predictores similar al lineal, se busca realizar una serie de transformaciones que pasen las probabilidades del intervalo [0,1] a todo el intervalo real, para luego poder asumir esta relación. Para simplificar la notación se considerará $p = \pi(x)$.

La primera transformación llevará el intervalo [0,1] en la parte positiva de la recta real:

$$p \in [0, 1] \rightarrow \frac{p}{1-p} \in R^+.$$

Ahora, basta aplicar una última transformación que lleve la parte positiva de la recta real en toda ella, como puede ser el logaritmo:

$$\frac{p}{1-p} \in R^+ \rightarrow log\ (\frac{p}{1-p}) \in R.$$

A la composición de ambas transformaciones se le conoce como función *logit*, que llevará el valor de una probabilidad en todo el intervalo real:

$$g(p) = log\ (\frac{p}{1-p})\ \forall p \in [0, 1].$$

Por lo tanto, como su dominio es la recta real, se puede escribir de la siguiente manera, asumiendo una forma lineal entre los predictores:

$$log\ (\frac{p}{1-p}) = x_i \beta.$$

Despejando p, llegamos a que:

$$p = \frac{e^{x_i \beta}}{1 + e^{x_i \beta}}.$$

Por lo tanto:

$$\pi(x_i, \beta) = \frac{e^{x_i\beta}}{1 + e^{x_i\beta}}.$$

Para estimar los parámetros de este modelo se utiliza la estimación por máxima verosimilitud. Teniendo en cuenta la distribución de la respuesta, la función de verosimilitud adoptará la siguiente forma:

$$\mathrm{L}(\beta) = \prod_{i=1}^{n} [\pi(x_i, \beta)^{y_i}(1 - \pi(x_i, \beta)^{1-y_i})].$$

Una práctica muy habitual es trabajar con la log-verosimilitud, por las buenas propiedades que poseen los logaritmos. Por lo tanto:

$$\log \mathrm{L}(\beta) = \sum_{i=1}^{n} [y_i \log \pi(x_i, \beta) + (1 - y_i)\log(1 - \pi(x_i, \beta))].$$

Para maximizar esta función se deriva respecto a β y se iguala a 0:

$$\frac{\partial \log \mathrm{L}(\beta)}{\partial \beta} = \sum_{i=1}^{n} \frac{\partial \pi(x_i, \beta)}{\partial \beta} \frac{1}{\pi(x_i, \beta)} [y_i - \pi(x_i, \beta)].$$

Como $\pi(x, \beta) = \dfrac{e^{x_i\beta}}{1 + e^{x_i\beta}}$, tenemos que:

$$\begin{aligned} \frac{\partial \pi(x, \beta)}{\partial \beta} &= \frac{x_i e^{x_i\beta}}{(1 + e^{x_i\beta})^2} \\ &= x_i \pi(x, \beta)(1 - \pi(x, \beta)). \end{aligned}$$

Con lo cual:

$$\frac{\partial \log \mathrm{L}(\beta)}{\partial \beta} = \sum_{i=1}^{n} x_{ii}[y_i - \pi(x_i, \beta)] = 0.$$

Para resolver esta ecuación no existe una solución explícita, por lo que se pueden utilizar métodos iterativos como Newton-Raphson o el método IRLS.

Modelo aditivo

Otra alternativa interesante será la de utilizar un modelo que permita establecer relaciones no lineales entre los predictores y la respuesta: el modelo aditivo (Hastie et al 2013). Previamente, se necesita describir los métodos de suavizado (Hastie et al 2013), que jugarán un papel fundamental en la estimación de los parámetros. Pese a que hay una infinitud de métodos de suavizado, únicamente se describirán dos de los más comunes: el *scatterplot* y los *splines* cúbicos naturales.

El *scatterplot*, o método de suavizado de diagramas de dispersión, sirve para los casos en los que se considera un único predictor. Este método busca captar la relación entre ambas variables mediante una regresión lineal, por lo que uno de sus grandes inconvenientes es que, al tratarse de una regresión lineal, resulta bastante lógico que este método produzca buenos ajustes cuando la relación entre ambas variables sea de este tipo, pero si no lo es el ajuste no será bueno. Por lo tanto, una buena idea alternativa es ajustar una curva que sea capaz de ajustarse mejor a los datos. Es aquí donde entra la segunda alternativa descrita dentro de los métodos de suavizados: los *splines*.

Un *spline* es una función $s : [a, b] \to R$ con J nodos $t_1, \cdots, t_J$ que cumple lo siguiente:

- $a = t_0 < t_1 < \cdots < t_J < t_{J+1} = b$.
- s es un polinomio de grado menor o igual que p en $[t_l, t_{j+1}] \forall j \in \{0, ..., J\}$.

Los *splines* más utilizados son los *splines* cúbicos naturales, que añaden la restricción adicional de que se cancelen sus segundas y terceras derivadas en los nodos frontera:

$$s''(a) = s''(b) = s'''(a) = s'''(b) = 0.$$

Geométricamente, los *splines* cúbicos naturales pueden ser interpretados como un *spline* cúbico en el que los nodos extremos son lineales.

Ahora, ya estamos en disposición de presentar los modelos aditivos. Estos son modelos no paramétricos que buscan una mayor flexibilidad en la relación entre la variable respuesta y los predictores. Mientras que en los modelos lineales esta relación era proporcional, en los modelos aditivos el efecto de cada variable no asume ninguna forma paramétrica, como se puede ver en su formulación:

$$Y = \alpha + f_1(X_1) + \cdots + f_p(X_p) + \epsilon,$$

donde f_j, con j $\in \{1, ..., p\}$, son funciones sin ningún tipo de restricción, y los errores son independientes de los predictores, homocedástivos y con media nula.

Para estimar estas funciones se utiliza el algoritmo se utilizará el algoritmo de *Backfitting*, un algoritmo muy general que permite el ajuste no lineal utilizando otros métodos de ajuste.

Algoritmo *Backfitting*.

---

1. Inicialización:
   - $\hat{\alpha} = \dfrac{1}{N}\displaystyle\sum_{i=1}^{N} y_i.$
   - $\hat{f}_j = 0 \; \forall j.$
2. Proceso iterativo:
   - $\hat{f}_j = s[\{y_i - \hat{\alpha} - \displaystyle\sum_{k \neq j} \hat{f}_j(x_{ik})\}_1^N]$
   - $\hat{f}_j = \hat{f}_j - \dfrac{1}{N}\displaystyle\sum_{i=1}^{N} \hat{f}_j(x_{ij})$

   donde s es un método de suavizado, como por ejemplo un *spline* cúbico natural, el cual ya se describió anteriormente.
3. Seguir con el proceso iterativo hasta que las funciones f_j no cambian entre dos iteraciones consecutivas.

GAM

Los GAM (Hastie et al 2013) son una generalización de los modelos aditivos de una manera similar a como los GLM generalizaban a los modelos lineales: para evitar la suposición de normalidad de la respuesta, se utiliza una función link que se aplica a la esperanza condicional de la respuesta. Teniendo esto en cuenta, se pueden formular de la siguiente manera:

$$g(E(Y|X)) = \alpha + f_1(X_1) + \cdots + f_p(X_p),$$

siendo las f_j funciones suaves desconocidas.

Para estimar estas funciones se utiliza el algoritmo *local scoring*, el cual se basa en el ajuste repetido de un modelo aditivo.

Algoritmo *local scoring.*

1. Inicialización:
$$\alpha = g\left(\sum_{i=1}^{n} \frac{y_i}{n}\right),\ f_1^0 = \cdots = f_p^0 = 0.$$

2. Se construye la variable a la que le ajustaremos un modelo aditivo:
$$z_i = \eta_i^0 + (y_i - \mu_i^0)\left(\frac{\partial \eta_i}{\partial \mu_i}\right)_0$$
donde $n_i^0 = \alpha^0 + \sum_{j=1}^{p} f_j^0(x_{ij})$ y $\mu_i^0 = g^{-1}(\eta_i^0)$.

3. Se construyen los pesos:
$$w_i = \frac{1}{g'(\mu_{0i}^2)\hat{\mathrm{Var}}(Y_i)}(V_i^0)^{-1}$$

4. A partir de la variable y los pesos construídos, ajustar un modelo aditivo, obteniendo las funciones f_j^1, el predictor η^1 y estimaciones μ_i^1.

5. Repetir el proceso hasta que el valor:
$$\triangle(\eta^1, \eta^0) = \frac{\sum_{j=1}^{p} ||f_j^1 - f_j^0||}{\sum_{j=1}^{p} ||f_j^0||}$$
sea más bajo que el umbral establecido.

3.2. *Random Forest*

A continuación se describirán los modelos *Random Forest* (Hastie et al 2013), que resultan una alternativa interesante ya que nos permitirá hacer una reducción de la varianza mediante un promedio de las predicciones obtenidas mediante distintos arboles de decisión.

Árboles de decisión

El primer modelo descrito será el árbol de decisión (Hastie et al 2013), que será clave en la construcción del *Random Forest*. Un árbol de decisión es un método muy simple, que consiste en ir partiendo el espacio predictor dicotómicamente según el valor que tome cada variable y estableciendo un umbral que irá separando los datos según si el valor de estos superan el umbral o no.

En la Figura 3.1 se puede ver el funcionamiento de este método. A partir de un nodo inicial, donde se agrupan la totalidad de las observaciones, se va ramificando según el valor que tome cada variable.

Como se puede observar, en cada nodo se realiza una partición del espacio predictor a partir de cierta condición, que aparece en la parte superior. Esta condición vendrá dada por un valor umbral de cierta variable, que agrupará en dos nodos distintos: las observaciones que cumplan la condición será agrupadas en el nodo situado en la siguiente división a la izquierda, mientras que las que no en el de la derecha.

Por ejemplo, en el primer nodo se va a dividir la totalidad de nuestras muestras según el valor de su número de retweets. Todas las muestras con un número de *retweets* igual o inferior a 105.5 (54185) serán agrupadas en el siguiente nodo a la izquierda, mientras que las que no (269) irán al nodo de la derecha.

En cada nodo se pueden ver el total de muestras que llegan en el índice *samples* y como se dividen tras la imposición de la condición en *value*. Además, en *class* podemos ver la clasificación de los elementos del nodo.

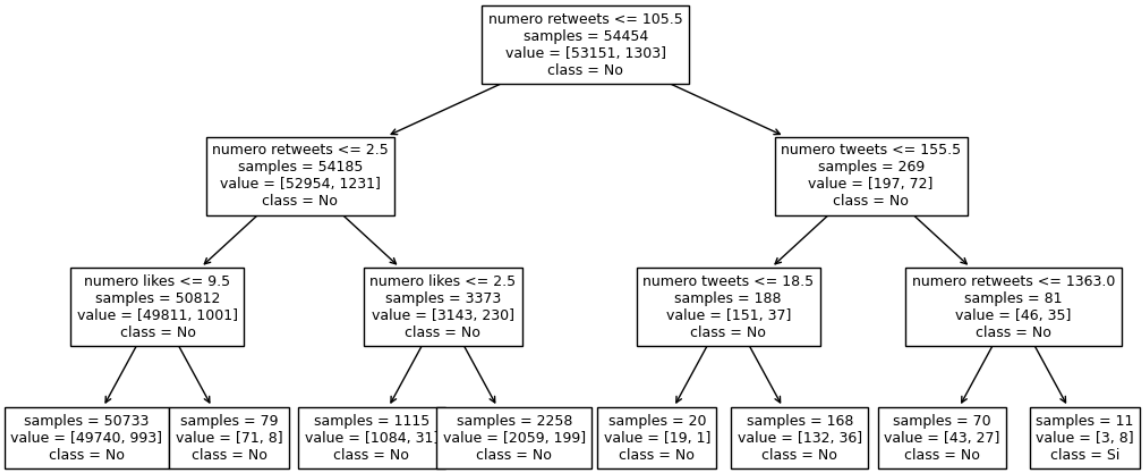


Figura 3.1: Ejemplo del funcionamiento de un árbol de decisión.

Una de las grandes ventajas de este método es que es bastante intuitivo, y tiene una interpretación bastante sencilla. En este caso, los CVE's con menos de 105.5 *retweets*, menos de 2.5 *retweets* y menos de 9.5 *likes* no van a ser explotados.

Como se comentó anteriormente, para construír el modelo se "parte" el espacio predictor en distintas regiones. Suponiendo que se divide en p regiones, estas se pueden denotar como $R_1, \cdots, R_p$.

Ahora, el problema radica en como seleccionar dichas regiones correctamente para que las predicciones sean óptimas. Primero, se describirá como resolver este problema para el caso de regresión, y luego se adaptará para la clasificación.

Una primera idea podría ser utilizar mínimos cuadrados (RSS), algo muy común en los modelos de ML, buscando minimizar el siguiente valor:

$$RSS = \sum_{j=1}^{p} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2.$$

Esto será inviable debido a la infinitud de distintas regiones R_j. Para solventar este problema, una alternativa es utilizar el siguiente proceso iterativo conocido como método CART.

Algoritmo Método CART.

1. En la primera iteración, se consideran todos los datos. Se selecciona una variable X_j y un punto de corte s, de manera que se definan los siguientes hiperplanos, que delimitarán los dos siguientes nodos:

$$R_1 = \{X : X_j \leq s\}$$
$$R_2 = \{X : X_j > s\}$$

   Por lo tanto, se seleccionan los valores de j y s de manera que minimicen el siguiente valor:

$$\sum_{i \in R_1} (y_i - \hat{y}_{R_1})^2 + \sum_{i \in R_2} (y_i - \hat{y}_{R_2})^2$$

2. El proceso será análogo para el resto de iteraciones, salvo que en vez de utilizar la totalidad de los datos, se utilizarán los datos contenidos en las regiones anteriores.

De manera natural, surge la pregunta de cuanto grande interesa que sea el árbol. Para tomar esta decisión, se tendrán en cuenta los siguientes factores:

- Si el árbol es demasiado grande, va a ser muy dificil de interpretar, y va a haber sobreajuste. De hecho, si se hace crecer el árbol lo suficiente se podría llegar a un ajuste perfecto en la muestra de entrenamiento, haciendo divisiones hasta tal punto que cada nodo terminal incluya una única observación, lo que no nos interesará ya que la varianza será muy elevada.

- Por otra parte, si el árbol es muy pequeño, la interpretación será mucho más sencilla y la varianza será mucho más baja, pero habrá infraajuste.

Por lo tanto, el siguiente punto de interés será determinar el tamaño del árbol de tal forma que se logre un equilibrio entre el sobreajuste y el infraajuste. Para ello, se introduce un concepto conocido como "poda de árboles", que consiste en colapsar cualquier cantidad de nodos internos, reduciendo el tamaño del árbol original. El nuevo árbol que se origina es conocido como subárbol.

A la hora de obtener el mejor subárbol, se podría pensar en buscar el que tenga menor error, lo que será impracticable debido al elevado número de subárboles. Para solucionar este problema, se considerará el tamaño del árbol como un hiperparámetro y se incluye una penalización en el RSS que dependa del tamaño del árbol:

$$RSS_\alpha = \sum_{j=1}^{t} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 + \alpha \cdot t,$$

siendo α un parámetro de penalización, y t el tamaño del árbol.

Para seleccionar este valor, existen dos alternativas:

- Opción 1: Seleccionar el valor que minimice el error. Esto será mucho más costoso computacionalmente, por lo que se buscarán otras alternativas.

- Opción 2: Utilizar un criterio *one-standard-error*, que selecciona el árbol más simple posible dentro de los que tienen un error menor que el estándar.

Una vez vista como se realiza la elección de las regiones para el caso de regresión, veamos que modificaciones son necesarias para adaptarlo para métodos de clasificación[1]. Para ello, se consideran

[1] En nuestro caso, al tratarse de un problema de clasificación binaria, únicamente habrá dos categorías ($K = 2$).

otras medidas alternativas al RSS:

$$\text{Proporción de errores de clasificación} : 1 - \max_{\text{k}} \hat{p}_k.$$

$$\text{Índice de Gini} : \sum_{k=1}^{K} \hat{p}_k(1 - \hat{p}_k).$$

$$\text{Entropía} : -\sum_{k=1}^{K} \hat{p}_k \log(\hat{p}_k).$$

Bagging

El siguiente modelo visto será el *Boostrap Aggregating (Bagging)* (Hastie et al 2013), que también resultará clave en la construcción de los modelos *Random Forest*, al igual que los árboles de decisión.

El *Bagging* es una técnica de ensamblado utilizada para mejorar la estabilidad y reducir la varianza. Este proceso se lleva a cabo mediante la generación de muchas muestras de entrenamiento con las que poder entrenar al modelo para luego promediar las predicciones, reduciendo la variabilidad ya que los errores de cada modelo individual se compensan mutuamente.

Las nuevas muestras son generadas por *Bootstrap*, que utiliza muestreo uniforme (el tamaño generado en cada muestra será siempre el mismo) y con reemplazo (un elemento puede ser seleccionado varias veces para una misma muestra) para generar las distintas muestras de entrenamiento. Estas nuevas muestras serán utilizadas para obtener predicciones, tantas como muestras generadas, y a partir de la media (en regresión) o de la moda (en clasificación) se obtendrá la predicción definitiva.

En la Figura 3.2 podemos ver una idea intuitiva del funcionamiento del *Bagging*. En este caso, se puede ver que en el muestreo no se repite ninguna de las observaciones, pero, como se comentó en el anterior párrafo, al tratarse de un muestreo con reemplazo, este podría no ser el caso.

De hecho, una muestra *Bootstrap* va a contener muchas muestras repetidas. Aproximadamente, utiliza una proporción de $(1 - \frac{1}{n})^n \approx 1 - e^{-1} = 0{,}6321$ muestras. Las restantes son conocidas como muestras *Out-Of-Bag* (OOB), que pueden ser utilizadas para obtener una medida del error, utilizándolas como si fueran un conjunto de test.

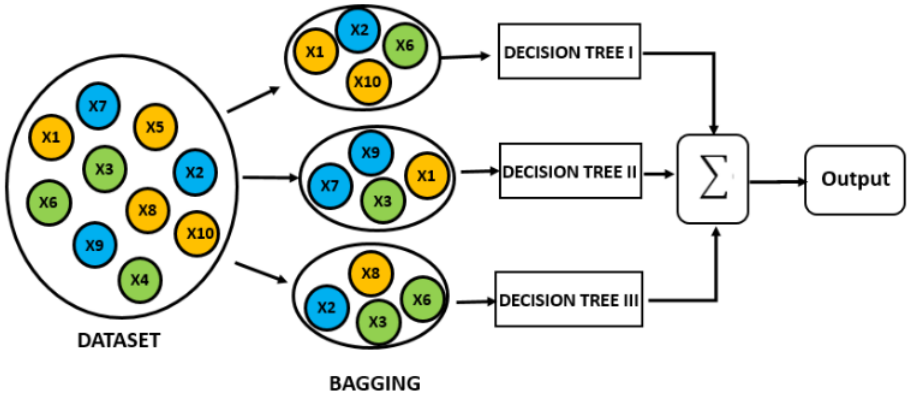


Figura 3.2: Ejemplo del funcionamiento del *Bagging*.

Random Forest

Por lo tanto, una vez descritos los árboles de decisión y el *Bagging* ya se podrá proceder con el *Random Forest* (Hastie et al 2013). El *Random Forest* es una variante del *Bagging* diseñado para

trabajar con árboles de decisión. La idea de este modelo es crear un gran número de árboles y combinar sus predicciones para obtener un mejor rendimiento.

La principal diferencia entre el *Random Forest* y el *Bagging* reside en la generación de los árboles: cuando se utiliza *Bagging*, los árboles generados son muy parecidos en la parte alta, provocando que estén altamente correlados y que la reducción de la varianza sea mucho menor, mientras que el *Random Forest* buscará como solucionar este problema.

Por lo tanto, el procedimiento del *Random Forest* consiste en generar los árboles utilizando unicamente m predictores escogidos de manera aleatoria a la hora de crear cada árbol, buscando que los estos sean más diferentes y provocando que la reducción de la varianza sea mucho mayor. Para seleccionar el valor de m se considerará como un hiperparámetro, pero lo habitual es usar $m = \frac{p}{3}$ en regresión y $m = \sqrt{p}$ en clasificación.

Capítulo 4

Desarrollo del trabajo

En esta sección se describe como fue implementada toda la teoría que ha sido explicada en los anteriores capítulos, así como una recopilación de los resultados. Para ello, serán de vital importancia las librerías *stats*(Perktold et al 2010), *sklearn*(Buitinck et al 2013) e *imbalanced_learn*(Aridas et al 2016) del lenguaje *Python*(Drake et al 1995).

4.1. Implementación

En los modelos de *Machine Learning* resulta imprescindible la división de nuestro conjunto de datos en dos partes: una de entrenamiento, que corresponderá a la mayor parte de nuestro *dataset* y que será utilizado para entrenar nuestros modelos, y otra de menor tamaño sobre la que se producirá la evaluación del mismo. De esta forma, se obtienen los subconjuntos de datos *Train* y *Test* mediante la función **train_test_split()** en una proporción de 70 % − 30 %. Además, como esta partición se hace de manera aleatoria (mediante el parámetro *shuffle="True"*), será necesario fijar una semilla.

A continuación, se aplicarán los métodos de balanceo explicados en la subsubsección 2.2.1 sobre las muestras de entrenamiento:

- Para el método SMOTE se ha usado la función **SMOTE()**, fijando una semilla debido a la aleatoriedad del algoritmo, y utilizando el número de k-vecinos por defecto (5). Buscando una mayor proporción de acierto al predecir la clase minoritaria, se establece el parámetro *sampling_strategy=1*, obteniendo el mismo de número de muestras de las dos clases.

- Para el método *Near Miss* se ha utilizado su última versión, indicando *version=3* en la función **NearMiss()**. Además, en este algoritmo se han usado el número de k-vecinos por defecto (3) y, con un razonamiento análogo al anterior, *sampling_strategy=1*.

- En Regresión Logística se usa la asignación de pesos mediante el parámetro *weight="balanced"* en la función **LogisticRegression()**.

- También se han utilizado funciones que ya tienen implementados los métodos de balanceo, como **BalancedRandomForestClassifier()**.

Durante el proceso de selección de variables, se distinguen distintos criterios:

- Las variables con varianza cercana a cero se eliminarán, por su nula contribución al modelo. Esto no se ha aplicado a las variables binarias, ya que pueden aportar información muy relevante pese a manifestarse en pocas ocasiones, como puede pasar con los *tags, products* y *vendors*.

- Como se describió en la subsección 2.5.1, la selección por correlaciones consiste en eliminar, para cada par de predictores que estén altamente correlados entre si, el que menor correlación tenga

con la respuesta. El coeficiente de correlación de Spearman ha sido obtenido mediante la función **spearmanr()**. Para calcular la correlación entre dos variables binarias se ha empleado la función **kendalltau()**, indicando *"variant=c"*. Con esta función también se obtiene el p-valor sobre el contraste que tiene como hipótesis nula la ausencia de relación. Para el cálculo de la correlación entre una binaria y una continua se utiliza la función **pointbiserialr()**.

- La selección por criterios automáticos se realiza mediante la función **RFECV()**, de la que ya se explicó su funcionamiento en la subsección 2.5.2. Se han usado los parámetros por defecto, salvo *"cv=StratifiedKFold(2)"* al estar trabajando con una respuesta binaria, y *scoring="roc_auc"* para usar el AUC como métrica que compare el rendimiento de los modelos.

Los métodos de regularización *Elastic Net* han sido desarrollados con el parámetro *l1_ratio* dentro de la función **LogisticRegression()**, siendo este el valor que toma α. Este se ha considerado como un hiperparámetro, y su valor se obtiene como se detallará más adelante cuando se describa la elección de los mismos.

Para la obtención del *threshold* se utilizará el estadístico que minimiza la distancia entre el valor absoluto de TPR y TNR, detallado en la subsubsección 2.2.3. Para ello, se obtendrán los valores del FPR (teniendo en cuenta que FPR=1-TNR), TPR y de los distintos *thresholds* mediante la función **metrics.roc_curve()**. Además, la obtención del AUC se lleva a cabo mediante la función **metrics.auc()**.

Adicionalmente, se ha creado una rejilla con los diferentes valores para obtener los hiperparámetros que se pretenden optimizar. Para ello, se ha usado la función **GridSearchCV()**, en la que los parámetros se optimizan mediante una búsqueda exhaustiva con validación cruzada. Se han usado el número de capas de validación cruzada por defecto (5), y como métrica para la comparación de modelos y determinación de los hiperparámetros el AUC, como ya se había justificado con anterioridad en la subsubsección 2.2.2.

Finalmente, la comparación del rendimiento de los modelos se ha llevado a cabo mediante las métricas de interés (TPR y TNR) obtenidas de la matriz de confusión. Para ello, se ha utilizado la función **confusion_matrix()** y se priorizan los modelos que tengan un valor más alto (y equilibrado) de ambas métricas.

4.2. Resultados

En la segunda parte de este capítulo se han recopilado los resultados de los distintos modelos utilizados en el trabajo. En el Cuadro 4.1 se muestran los resultados obtenidos de los mejores modelos:

	3 MESES		**6 MESES**		**12 MESES**	
	TPR(%)	**TNR(%)**	**TPR(%)**	**TNR(%)**	**TPR(%)**	**TNR(%)**
LOGIT	66	65	66	65	66	65
GAM	68	65	66	66	69	69
RF	75	74	73	73	73	73

Cuadro 4.1: Cuadro de recopilación de resultados.

Las filas representan a los modelos que han sido utilizados durante la realización de este estudio:

LOGIT (Regresión logística), GAM y RF (*Random Forest*). Por otra parte, cada par de columnas hace referencia a una franja temporal, distinguiendo el porcentaje de acierto del TNR y del TPR.

Además, se ha realizado una comparativa de las predicciones realizadas por los distintos modelos para la clase mayoritaria, recopiladas en el Cuadro 4.2, a partir de un subconjunto de ocho observaciones tomadas aleatoriamente del conjunto de *Test*.

	threshold	Obs. 1	Obs. 2	Obs. 3	Obs. 4	Obs. 5	Obs. 6	Obs. 7	Obs. 8
LOGIT	0.49	0.44	0.52	0.59	0.46	0.55	0.35	0.56	0.41
GAM	0.36	0.44	0.53	0.59	0.46	0.55	0.35	0.56	0.42
RF	0.5	1	0.83	0.49	0.32	0.46	0.36	0.74	0.02

Cuadro 4.2: Cuadro de recopilación de las predicciones.

En la primera columna podemos ver el valor del *threshold* para cada método, que era obtenido automáticamente de manera que se optimizase la proporción de aciertos de la clase minoritaria. Es llamativo que este valor sea tan inferior en el GAM, lo que significa que, por lo general, los valores de las predicciones para la clase minoritaria serán más bajos que en los otros dos modelos.

Comparando las predicciones se puede observar como el RF hace predicciones más extremas que los otros dos métodos. Esto es coherente con el hecho de que el valor del AUC de este modelo sea el mayor de todos, ya que es el modelo que mejor distingue entre las dos clases.

Además, a partir del *threshold* y de las probabilidades recopiladas en el Cuadro 4.2, se puede realizar una comparativa de la clasificación hecha por cada uno de los modelos, como se puede observar en el Cuadro 4.3.

	Obs. 1	Obs. 2	Obs. 3	Obs. 4	Obs. 5	Obs. 6	Obs. 7	Obs. 8
LOGIT	0	1	1	0	1	0	1	0
GAM	1	1	1	1	1	0	1	1
RF	1	1	0	0	0	0	1	0
Real	0	0	0	0	1	0	0	1

Cuadro 4.3: Cuadro de recopilación de la clasificación realizada por los distintos métodos.

A la vista de la clasificación realizada por los tres modelos, se pone de manifiesto la priorización de clasificar por la clase minoritaria, ya que pese a que la explotabilidad del CVE solo se manifieste dos veces realmente, los modelos predicen que se va a manifestar 3, 4 y hasta en 7 de las 8 observaciones.

Se puede percibir las dificultades que presentan los modelos en la correcta clasificación. En particular, el mejor modelo (RF) no es capaz de detectar ninguna de las dos observaciones que han sido explotadas.

Para terminar esta recopilación de los resultados, se describirán brevemente los mejores modelos para cada franja temporal, en base a los resultados recopilados en el Cuadro 4.1.

Primero, se enunciarán las variables utilizadas en los modelos. Para los modelos de 3 y 6 meses se utilizarán las siguientes variables, obtenidas de la selección automática realizada con la función

RFECV():

- Variables relacionadas con las métricas CVSS, que aportan información sobre la explotabilidad e impacto de las vulnerabilidades, así como una puntuación general: *impact.baseMetricV2.cvssV2.baseScore*, *impact.baseMetricV2.exploitabilityScore*, *impact.baseMetricV2.impactScore*, *impact.baseMetricV3.cvssV3.baseScore*, *impact.baseMetricV3.exploitabilityScore*, y *impact.baseMetricV3.impactScore*.

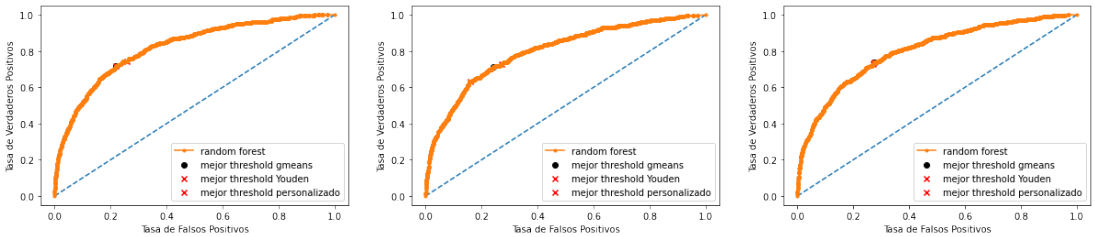
- Variables *tags, products* y *vendors*, que aportan información técnica sobre cada CVE, como se ha explicado en la subsección 2.1: *remote attacker*, *execute arbitrary*, *Cross site*, *buffer overflow*, *inject arbitrary*, *local user*, *web script*, *SQL injection*, *authenticated user*, *remote authenticated*, *code execution*, *Directory traversal*, *Base Score*, *network access*, *traversal vulnerability*, *cisco*, *linux*, *mozilla*, *microsoft*, *apple*, *oracle*, *ibm*, *sun*, *google*, *adobe*, *linux_kernel*, *ios*, *firefox*, *chrome*, *jre*,*jdk*,*opera_browser* y *php*.

- Variables relacionadas con las RRSS, que miden el impacto de la publicación de un CVE en las mismas: *numero_tweets*, *numero_retweets*, *numero_respuestas*, *numero_likes* y *numero_citas*.

Además, para el modelo de 12 meses se incluirán las variables *seamonkey*, *junos* (*tags, products* y *vendors*) y *numero_mensajes_telegram* (RRSS).

En la Figura 4.1 se representan las gráficas de la curva ROC del mejor modelo de cada franja temporal, junto con los distintos valores óptimos del *threshold* obtenidos por los estadísticos G-means, Youden y el creado de acuerdo a nuestros intereses.

El valor del AUC será de 0.822, 0.801 y 0.803 para los modelos de 3, 6 y 12 meses, respectivamente. En base a esto, se puede concluír que el modelo con mejor capacidad para distinguir entre las dos clases es el de 3 meses.



(a) RF para 3 meses. (b) RF para 6 meses. (c) RF para 12 meses.

Figura 4.1: Gráficas ROC para los mejores modelos.

Capítulo 5

Conclusiones y trabajo futuro

Para dar por finalizado el trabajo se han realizado una serie de conclusiones, a partir del repaso de las distintas etapas en las que se ha desarrollado el mismo:

- Tras el análisis de la variable respuesta, y en base al alto grado de desbalanceo entre las clases, resulta evidente la necesidad de solventar este problema. Para ello, han resultado capitales los métodos de balanceo para obtener más observaciones con las que poder entrenar a los distintos modelos utilizados. Buscando equilibrar el número de observaciones de las dos clases de forma que estas sean lo más distintas posible, para ayudar a nuestros modelos en la clasificación de las mismas, se ha probado con los métodos SMOTE y *Near Miss*. Además, la elección del estadístico para la obtención del *threshold* óptimo ha sido fundamental, y será imprescindible siempre y cuando se mantenga la preferencia por predecir la clase de los CVE's explotados.

- Observando las variables que han sido utilizadas en los mejores modelos, se puede afirmar que la mayoría de ellas aportan información relevante, sin presentar problemas de correlación importantes.

- La consideración de distintas ventanas temporales fue un éxito, ya que produjo una mejora en el rendimiento de los modelos. De aquí, se puede concluír que los datos más recientes de Twitter son de mayor calidad, por lo que parece que el rendimiento de los modelos puede seguir mejorando con el paso del tiempo si estos se siguen reentrenando con los nuevos CVEs que vayan siendo publicados.

- De entre todos los modelos con los que se ha trabajado, el que mejor rendimiento ha presentado, tanto a nivel predictivo como a nivel computacional, ha sido el *Random Forest*. Uno de los grandes inconvenientes de este modelo es la poca intepretabilidad que posee, lo cual puede hacernos pensar en considerar otros modelos que sean más fáciles de analizar, y que puedan presentar un rendimiento parecido. En este caso, se ha probado con otros como el modelo logístico o los árboles de decisión, pero sus resultados han sido notablemente peores.

Por lo tanto, y en base a lo expuesto anteriormente, el trabajo futuro deberá ir enfocado buscando facilitar el reconocimiento de las dos clases por parte de los modelos de aprendizaje automático, mediante otras técnicas de balanceo y la obtención de nuevas variables que permitan discernir distintos patrones entre los CVE's que no van a ser explotados y los que sí.

Bibliografía

[1] Adjerid, Edwards, Jacobs, Romanosky, Roytman (2019) Exploit Prediction Scoring System (EPSS). Digital Threats: Research and Practice, 2(3), 1-17.

[2] Chen, H., Liu, R., Park, N., Subrahmanian, V. S. (2019, July). Using twitter to predict when vulnerabilities will be exploited. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery data Mining (pp. 3143-3152).

[3] Bhatt, N., Anand, A., Yadavalli, V. S. (2021). Exploitability prediction of software vulnerabilities. Quality and Reliability Engineering International, 37(2), 648-663.

[4] Bozorgi, M., Saul, L. K., Savage, S., Voelker, G. M. (2010, July). Beyond heuristics: learning to classify vulnerabilities and predict exploits. In Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 105-114).

[5] Sabottke, C., Suciu, O., Dumitraș, T. (2015). Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits. In 24th USENIX Security Symposium (USENIX Security 15) (pp. 1041-1056).

[6] Almukaynizi, M., Nunes, E., Dharaiya, K., Senguttuvan, M., Shakarian, J., Shakarian, P. (2017, November). Proactive identification of exploits in the wild through vulnerability mentions online. In 2017 International Conference on Cyber Conflict (CyCon US) (pp. 82-88). IEEE.

[7] Cohn, De La Torre, Jeni (2013) Facing imbalanced data–recommendations for the use of performance metrics. Int Conf Affect Comput Intell Interact Workshops. 2013;2013:245-251. doi: 10.1109/ACII.2013.47. PMID: 25574450; PMCID: PMC4285355.

[8] Adeliyi, Mqadi, Naicker (2021) Solving misclassification of the credit card imbalance problem using near miss. Mathematical Problems in Engineering. doi:10.1155/2021/7194728

[9] Wang, Xu, Zhang (2006) "Classification of Imbalanced Data by Using the SMOTE Algorithm and Locally Linear Embedding".8th international Conference on Signal Processing, Guilin, China, 2006, pp. doi: 10.1109/ICOSP.2006.345752.

[10] Brummit, Servén (2018) pyGAM: Generalized Additive Models in Python. https://pygam.readthedocs.io/en/latest/ Accedido 27 de Enero de 2023

[11] Buitinck et al (2013) API design for machine learning software: experiences from the scikit-learn project. https://doi.org/10.48550/arXiv.1309.0238

[12] Perktold, Seabold (2010) Statsmodels: Econometric and statistical modeling with python. doi: 10.25080/Majora-92bf1922-011

[13] Aridas, Lemaitre, Nogueira (2016) Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning.

[14] Hastie, James, Tibshirani, Witten (2013) An Introduction to Statistical Learning with Applications in R. Springer.

[15] Drake, Rossum (1995) Python reference manual. https://docs.python.org/3/reference/ Accedido 27 de Enero de 2023

[16] Daniel (1990) Applied Nonparametric Statistics (2nd edición). Brooks/Cole.

[17] John, Linacre (2008) The Expected Value of a Point-Biserial (or Similar) Correlation. Rasch Measurement Transactions. 22 (1): 1154.

[18] Mauri L., O'Malley , Zou (2007). Receiver-operating characteristic analysis for evaluating diagnostic tests and predictive models. https://doi.org/10.1161/CIRCULATIONAHA.105.594929