Taxonomía de algoritmos basados en Machine Learning aplicados en la Ingeniería de Software.

Alex Paul Enciso Rolon[1][0000−1111−2222−3333], Osvaldo González Prieto[1][1111−2222−3333−4444], and Benjamin Barán[2][2222−−3333−4444−5555]

[1] Facultad Politécnica, Universidad Nacional del Este, Ciudad del Este, Paraguay
{alexpenciso,osvalcde}@fpune.edu.py
[2] Facultad de Informática, Universidad Comunera, Asunción, Paraguay bbaran@cba.com.py

Abstract. The constant growth of the software industry has driven companies to explore new ways to improve their processes, generating novel techniques to optimize the tasks involved in software development, in order to increase the efficiency of these processes. At the same time, the terms "Artificial Intelligence" and "Machine Learning" (ML), are being increasingly used, but there still is a certain lack of knowledge about these concepts. Given this context, our main objective is to establish a connection between these disciplines, in order to better understand the benefit of using ML in Software Engineer. In this work, a systematic analysis of the scientific literature published between 2018 and 2023 has been carried out in order to create a taxonomy of Machine Learning algorithms applied to the stages required for software development. The most prominent results indicate that the testing phase in the software development cycle is one of the most researched areas in relation to the aforementioned challenges. Furthermore, it has been observed that some ML algorithms such as Random Forest demonstrate acceptable performance in optimizing one or more tasks simultaneously in the software development process.

Keywords: Machine Learning · Software Engineering · Taxonomy.

1 Introducción

Este trabajo se originó en la necesidad de conocer nuevas herramientas aplicables a la Ingeniería de Software con el propósito de optimizar sus procesos y mejorar la calidad del producto. La elección de investigar el uso de Aprendizaje de Máquinas mas comunmente mencionado como Machine Learning (ML), en esta área se basó en la percepción de que muchos procesos podían ser conceptualizados como problemas de optimización, y ML ofrecía una posible solución proporcionando una aproximación a la función objetivo, según se señala en [35]. El objetivo principal es optimizar los procesos de la Ingeniería de Software para facilitar la creación de aplicaciones, ya sean simples o complejas.

La industria de desarrollo de software experimentaría un crecimiento del 21% para el año 2028 [33]. La tendencia predominante es la entrega rápida de soluciones, comprometiendo la calidad en fases como la codificación y pruebas, esta última considerada crítica en el desarrollo de software [1]. La ingeniería de software, vista como una construcción altamente compleja, presentaba desafíos cuando se enfrentaba a situaciones con un gran número de variables, motivando la exploración del ML como posible solución.

Estas situaciones nos lleva plantear las siguientes Preguntas de Investigación (PI):

PI1- ¿Existen evidencias de optimización en el proceso de desarrollo de software utilizando algoritmos basados en ML?

PI2- ¿Qué técnicas existen para la generación de modelos basados en ML en la ingeniería de software?

PI3- ¿Cuáles son las herramientas que se utilizan en el ML dentro de la Ingeniería de Software?

PI4- ¿Cuáles son los algoritmos más usados de ML aplicados para optimizar tareas de la Ingeniería de Software?

PI5- ¿Qué tareas de la ingeniería de software son optimizadas por los modelos de ML?

La necesidad de la industria de software de entregar soluciones de manera rápida y garantizar su calidad, con ML emergiendo como una herramienta potencialmente útil. El propósito del proyecto fue contribuir al conocimiento de los avances relacionados con el ML, explicando sus aplicaciones dentro de la ingeniería de software y cómo puede mejorar las distintas fases de este proceso.

En cuanto a la delimitación del alcance del trabajo, se planteó realizar un estudio técnico sobre algoritmos y modelos basados en Machine Learning aplicados en las diferentes etapas de la ingeniería de software, con el objetivo de identificar las ventajas derivadas de la utilización de estas herramientas.

Este trabajo, se propone como objetivo principal elaborar una taxonomía del uso de algoritmos basados en ML en la Ingeniería de Software.

2 Discusión de literatura relevante

Se identificaron algunos trabajos relevantes sobre este tema. Destacan diversas contribuciones, como el trabajo de Raphael Jenni [15], que exploró la falta de conocimiento de Machine Learning en ingeniería de software, centrándose en Deep Learning para ofrecer una visión detallada de tecnologías y herramientas en las fases de codificación y testeo del desarrollo de programas. Asimismo, el estudio de Julian Aron Prenner et al. [27] investigó el uso de "Codex" para la reparación automática de programas, concluyendo que, aunque posible, presenta limitaciones, resaltando su singularidad al enfocarse en una herramienta de ML para resolver errores en el desarrollo de software. Además, el trabajo de Chuan-Yung Tsai et al. [34] se destacó por la eficiente generación de entradas de prueba en casos automáticos de software mediante un framework asistido por aprendizaje reforzado profundo, concluyendo su viabilidad para el Cosmos SDK (Software Development Kit) y diferenciándose al emplear ML específicamente en la fase de prueba del software. Otro aporte relevante fue el trabajo de Juan Cruz-Benito et al. [10], que comparó arquitecturas de Deep Learning para la generación automática de código en Python, identificando fortalezas y limitaciones, focalizando su atención exclusivamente en esta etapa del desarrollo de software. Finalmente, el trabajo de Deepika Nagpal et al. [23] abordó la incertidumbre en el uso de Machine Learning en la ingeniería de requisitos, concluyendo que es viable emplear ML y procesamiento de lenguaje natural para clasificar requisitos, resaltando su singularidad al centrarse en la etapa de ingeniería de requisitos en el ámbito de la ingeniería de software.

Machine Learning (ML), traducido al español como aprendizaje automático o aprendizaje de máquina, "es el estudio sistemático de algoritmo y sistemas que mejoran su conocimiento o desempeño con experiencia" con el fin de solucionar un problema o tarea [13]. Otra definición de ML "cuando el agente de aprendizaje es una computadora, se llama Machine Learning Una computadora observa algunos datos, construye un modelo basado en los datos, y usa ese modelo para hacer hipótesis sobre el mundo y como una pieza de software que puede solucionar problemas" [29]. En otras palabras, ML es la utilización de uno o combinación de diversos algoritmos para la generación de un modelo, el cual es utilizado para solucionar una tarea específica. Este modelo es generado mediante la formulación de un problema de aprendizaje.

"La ingeniería de software es una disciplina que se interesa por todos los aspectos que interfieren en el proceso de desarrollo de software. Comienza con las primeras etapas de especificación del sistema hasta el mantenimiento de este cuando ya está puesto en marcha" [32]. Presenta un enfoque sistemático conocido como proceso de software, el cual es una secuencia de actividades que tienen como objetivo la elaboración de un producto de software. Consta de cuatro actividades fundamentales que son comunes a todos los procesos de software: Especificación del Software, Desarrollo de Software, Validación del software y Evolución del software [32]. Otra definición de Ingeniería de software es "La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software" [28]. Consta de varias capas, las cuales son: capa de proceso, capa de métodos, capa de herramientas y capa de compromiso con la calidad. Podemos decir entonces que la ingeniería de software consiste en la generación de un producto de software, siguiendo una serie de pasos denominados procesos los cuales ayudan a la construcción del producto. En términos generales, los antecedentes revisados compartían el uso de Machine Learning para abordar problemas en ingeniería de software. A diferencia de estos, este trabajo se enfoca en un estudio del estado del arte, buscando validar la hipótesis de una conexión entre la ingeniería de software y el ML. La contribución de este trabajo radica en proporcionar una perspectiva sólida y fundamentada, que puede servir como base para investigaciones y desarrollos futuros en este ámbito interdisciplinario. En [14] se identifican dos formas de relacionar la Ingeniería de Software con Machine Learning, las cuales son las siguientes:

1. Ingeniería de Software para Machine Learning: es la ejecución de las etapas de la Ingeniería de Software para el diseño, desarrollo y mantenimiento de sistemas que utilicen ML. Se busca diferenciar las implicancias de la elaboración de software que utilice ML en comparación al software tradicional.
2. Machine Learning para la Ingeniería de Software: se refiere a la utilización uno o más algoritmos de ML para optimizar diversos procesos que deben ser ejecutados durante las etapas de la Ingeniería de Software, tales como predicción de fallas de software, predicción de métricas de usabilidad, y estimación de costos, entre otros. Los investigadores utilizan los modelos de ML generados a partir de los datos de la Ingeniería de Software (código fuente, especificación de requisitos, casos de pruebas, etc.) para la elaboración de software de forma más eficiente y efectiva.

Este trabajo se enfoca únicamente en el punto 2. Es decir, Machine Learning para la Ingeniería de Software, explorando uno de los conceptos fundamentales de la Ingeniería de Software: las etapas o fases de este proceso, junto con el empleo de algoritmos de Machine Learning para optimizar las tareas relacionadas con cada etapa de esta disciplina.

3 Método

El esquema de trabajo para la elaboración de la taxonomía, presentado en la Figura 1, consiste en las siguientes etapas: Búsqueda de artículos, Selección de artículos, Extracción de Datos y Análisis de Resultados.

3.1 Proceso de búsqueda y selección de artículos

Para el presente trabajo, se definió y ejecutó un análisis del estado del arte sobre el Machine Learning y la Ingeniería de Software con el fin de estudiar los artículos más relevantes. Esta sección se ocupó del proceso de selección de la literatura, es resumido en la Figura 2. Tomando como ejemplo la metodología propuesta por [19], seguimos el siguiente marco de trabajo.

Fig. 1. Esquema de trabajo para la elaboración de la taxonomía

Búsqueda de Palabras Claves El proceso de selección de los artículos relevantes empieza con la búsqueda de artículos en la base de datos de CICCO [cicco.conacyt.gov.py] con las siguientes palabras claves en el título y/o resumen de los artículos: ["Machine Learning" AND "Software Engineering"]. La consulta está compuesta por dos partes: la primera es el objeto de estudio que en este caso es el "Machine Learning" y el otro elemento es en donde se aplica, el cual es en la "Ingeniería de Software". Además, se establece un filtro de fecha de publicación de los últimos 5 años que va desde 2018 a 2023. Al realizar este proceso de búsqueda de palabras claves, se obtuvieron 116 artículos.

Lectura de artículos El proceso de selección de artículos está compuesto por dos etapas, llamadas primer filtrado y segundo filtrado. En el primer filtrado, el investigador, solo lee el título y el resumen para la selección de los artículos relacionados al Machine Learning aplicado en la Ingeniería de Software y se aplican unos criterios de inclusión y criterios de exclusión detallados en la Tabla: 1.Al aplicar los criterios de selección durante el primer filtro se obtuvieron 39 artículos que tratan sobre técnicas de ML aplicados a optimizar al menos una tarea de la Ingeniería de Software. El segundo filtrado consiste en la lectura completa de los artículos seleccionados en la etapa anterior y se vuelven a aplicar los criterios de selección. Los artículos que pudieron ser accesibles en su versión completa fueron 35, de estos se identificaron cuales presentan resultados de aplicación de al menos una técnica de ML, cumpliendo con este último requisito, resultando en el universo final a ser estudiado, un total de 22 artículos.

CI	**Descripción del Criterio de Inclusión**
CI1	El artículo trata sobre el Machine Learning aplicado a almenos un proceso de Ingeniería de Software
CI2	El artículo trata sobre optimizar una tarea de la Ingeniería de Software
CI3	El artículo usa al menos una técnica de ML
CI4	El artículo fue publicado entre el año 2018 y 2023
CI5	El artículo proviene de una fuente bibliográfica reconocida (IEEE, ACM, Springer, ScienceDirect)
CE	**Descripción del Criterio de Exclusión**
CE1	El artículo es un análisis sistemático, taxonomía
CE2	El artículo está en un idioma distinto al español, portugués o inglés
CE3	El artículo no presenta resultados demostrados
CE4	El artículo completo no es accesible para su lectura por el equipo de investigadores

Table 1. Criterios de inclusión y exclusión de artículos

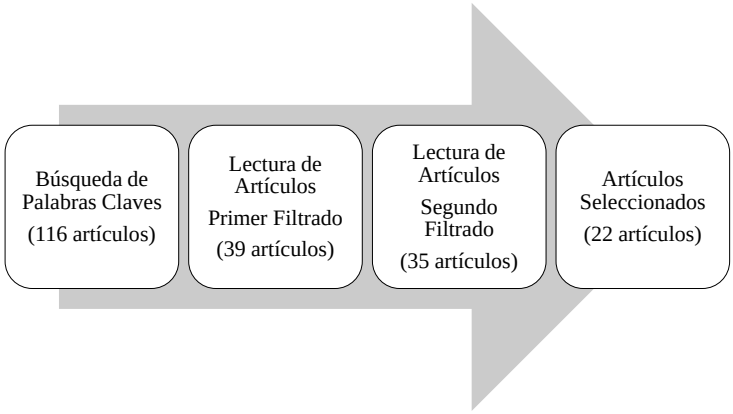

Fig. 2. Proceso para la búsqueda y selección de los artículos

3.2 Extracción de Datos

La etapa de extracción de datos, Figura 3, consiste en el análisis de los artículos seleccionados para extraer las informaciones necesarias (Tabla 2) que responden a las preguntas de investigación. Consta de las siguientes etapas: Extracción de Palabras Claves, Agrupación de palabras claves relacionadas, Definición de las métricas y Caracterización de los artículos. La Figura 4 muestra un gráfico que representa la distribución de los artículos según su fuente de origen.

Extracción de Palabras Claves En esta etapa se empieza leyendo el título, resumen y conclusión de los artículos seleccionados con la finalidad de extraer las palabras claves. Estas palabras claves deben guardar alguna relación con algunas de las siguientes métricas: preguntas de investigación, metodología aplicada en el artículo, variables estudiadas en el artículo y el tipo de pruebas o validación aplicada (Tabla 2).

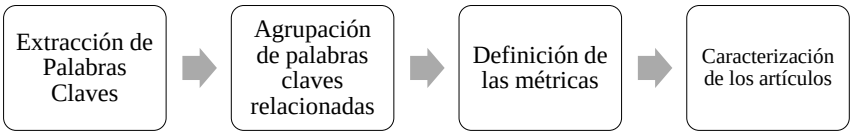

Fig. 3. Proceso para la extracción de datos

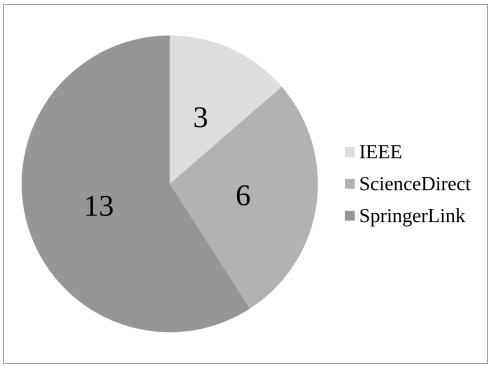

Fig. 4. Distribución de artículos por fuente

Agrupación de palabras claves relacionadas Una vez obtenida las palabras claves se procede a agruparlas según el tema al que se refieren. Ej.: palabras como "Ingeniería de Requisitos", "Diseño del software", "Desarrollo del Software" y "Prueba del Software" se pueden agrupar como Fases de la Ingeniería de Software.

Definición de las variables Una vez obtenidas las agrupaciones de las palabras claves, estas agrupaciones son utilizadas para el establecimiento de las variables de cada artículo. En esta etapa se define como son medidas cada una de las variables. Cada variable debe estar relacionada con al menos una pregunta de investigación (Tabla 2), ya que el objetivo es usar esas variables para poder responder las preguntas de investigación.

Caracterización de los artículos Para completar la etapa de extracción de datos, se elabora un Tabla de datos en una planilla electrónica con todos los datos bibliográficos y variables obtenidas de cada artículo. Esta Tabla esta compuesto por los campos que representan los datos bibliográficos de cada articulo y las variables que responden a al menos una de las preguntas de investigación.

4 Resultados

En la Tabla 3 se puede visualizar la cantidad de artículos encontrados agrupados según la fase de la Ingeniería de Software. Es importante mencionar que algunos trabajos analizados tratan con más de un algoritmo de ML, herramienta y/o involucran a más de una etapa de ingeniería de software. La Tabla 3 presenta las tareas de Ingeniería de Software que se examinaron en los artículos seleccionados. En particular, la tarea de 'Predicción de Fallas en el Software' en la etapa de Pruebas recibió la mayor atención

Campo	**Categoría**	**PI**
Titulo	Datos Bibliográficos	
Autores	Datos Bibliográficos	
Evento	Datos Bibliográficos	
Tipo de Evento	Datos Bibliográficos	
País	Datos Bibliográficos	
Fecha de Publicación	Datos Bibliográficos	
Algoritmo de Machine Learning	Algoritmos empleados para la solución del problema	PI1,PI4
Etapa de la Ingeniería de Software Tratada	Etapa en la que fue generado el modelo de ML	PI1,PI5
Tipo de Aprendizaje	Tipo de aprendizaje empleado para solucionar el problema	PI2
Mejor Algoritmo de ML	Algoritmo de ML que genero el modelo con mejor rendimiento	PI4
Herramientas Utilizadas	Herramientas utilizadas para la generación de los modelos de ML	PI3

Table 2. Clasificación de variables de los artículos analizados

en términos de investigación, posiblemente debido a la abundancia de conjuntos de datos disponibles para la creación de trabajos de entrenamiento relacionados con esta tarea. Se visualiza en la Figura 5 la cantidad de artículos agrupados por etapa de la ingeniería de software estudiada en cada investigación, con los siguientes detalles:

- *Etapa de Pruebas:* con un total de 11 artículos [11], [9], [22], [24], [7], [18], [25], [16], [36], [26] y [12];
- *Etapa de Ingeniería de Requisitos:* con 5 artículos [17], [3], [8], [4] y [6];
- *Etapa de Diseño:* con 2 artículos [30] y [31];
- *Etapa de Mantenimiento:* con 3 artículos [21], [20] y [5]
- *Etapa de Implementación:* con 1 artículo [2].

El tipo de tarea más investigada fue la de Predicción de fallas en el Software (Tabla 3). Esto probablemente se debe a que, la detección temprana de errores en el software es fundamental para garantizar su conformidad. Por lo tanto, es natural que sea la tarea que reciba más atención en investigación. La Figura 6 muestra la cantidad de artículos encontrados, agrupados según el tipo de aprendizaje que fue empleado para la generación del modelo de ML. En los 22 artículos analizados utilizan en su totalidad el tipo Aprendizaje Supervisado, además los artículos [8],[18] y [7] también utilizan el tipo Aprendizaje por Transferencia, este tipo de aprensizaje aprovecha un modelo pre-entrenado en una tarea específica para mejorar el rendimiento de otro modelo en una tarea relacionada. Esto acelera el proceso de entrenamiento y reduce la necesidad de grandes conjuntos de datos etiquetados para la nueva tarea, es decir, este tipo de entrenamiento primeramente utiliza el entrenamiento Aprendizaje Supervisado y cuando se incorporan nuevos grupos de datos a la base de conocimiento se utiliza el Aprendizaje por Transferencia. Los tipos 'Aprendizaje No Supervisado' y 'Aprendizaje por Refuerzo' no fueron empleados en ninguno de los artículos analizados. Suponemos que la razón por la que el aprendizaje supervisado predomina sobre los demás tipos de aprendizaje es debido al tipo de problema formulado por los trabajos investigados, ya que en su mayoría buscaban clasificar un determinado parámetro o característica dado un conjunto de valores pasados, por lo cual, el Aprendizaje Supervisado es el ideal para la solución de este tipo de objetivos.

La Figura 7 muestra las herramientas más populares usadas para la generación de los modelos de ML, con los siguientes detalles:

Tarea	Ref.
Ingeniería de Requisitos	
Clasificación de Requisitos de Seguridad	[17]
Demarcación de Requisitos	[3]
Detección de Requerimientos de Privacidad	[8]
Predicción de Recomendaciones de Seguridad del Software	[6]
Predicción de Requerimientos No Funcionales	[4]
Diseño	
Detección de la Severidad del Error de Estimación	[30]
Detección del Tipo de Error de Estimación	[30]
Estimación de Esfuerzo para Desarrollo del Software	[31]
Desarrollo	
(sin artículos)	
Pruebas	
Detección de Alcance de Comentarios de Código y Análisis Semántico	[9]
Detección de Deuda Técnica Admitida Consientemente	[18]
Detección de Inconsistencia en el Código	[7]
Detección de Olores en el Código	[24]
Predicción de Fallas en el Software	[11][22][25][16] [36][12]
Predicción de Test Inconsistentes	[26]
Implementación	
Detección de Commits Omitibles en Integración Continua	[2]
Mantenimiento	
Predicción de Clases Propensas al Cambio	[20]
Predicción de Mantenibilidad del Software	[21]
Predicción de Operaciones de Refactorización	[5]

Table 3. Tareas de la Ingeniería de Software estudiadas

- La herramienta más utilizada es *Weka* la cual fue utilizada en 7 artículos: [2], [11], [21], [3], [24], [16] y [6].
- *Scikit-Learn* fue utilizada en 6 artículos: [8], [18], [25], [26], [4] y [5],
- *Keras* en 4 artículos: [8], [7], [18] y [5],
- *R tool* en 3 artículos: [21], [24] y [30],
- *Python* en 3 artículos: [8], [25] y [16],
- *Keel Tool* [21],
- *Matlab* [16] y
- *Tensorflow* [7]

De los artículos analizados 9 no mencionaron la herramienta que utilizaron. Posiblemente la razón por la que no se mencionan en algunos artículos la herramienta utilizada es porque los algoritmos utilizados fueron implementados de manera manual en algún lenguaje de programación. También es importante destacar que no se encontró una diferencia significativa en la frecuencia de uso de una herramienta en particular. Los artículos no proporcionaron una explicación clara de porque seleccionaron una herramienta en particular.

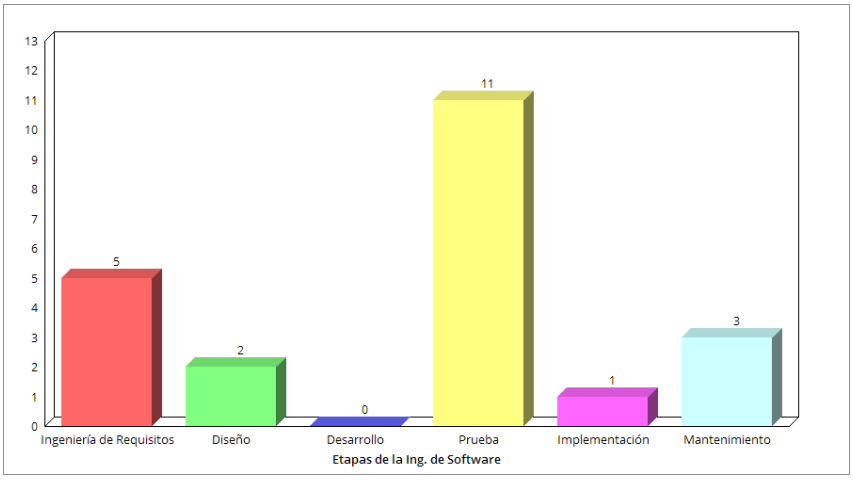

Fig. 5. Cantidad de artículos agrupados por etapas de la Ingeniería de Software

4.1 Cruce de información

En la Tabla 4, se pueden apreciar los algoritmos que demostraron un rendimiento sobresaliente en relación a las distintas etapas de la Ingeniería de Software. El artículo [21] no especifica cual es el mejor algoritmo entre los que utiliza para sus pruebas. El algoritmo 'Random Forest' destacó al aparecer en casi todas las fases de la Ingeniería de Software. Esto se debe, en gran medida, a que Random Forest es un modelo de conjunto (ensemble) compuesto por varios modelos, lo que le otorga una mejor capacidad de generalización en cualquier fase de la Ingeniería de Software.

Como se puede observar en la Tabla 4 y la Figura 5, ninguno de los artículos analizados trata sobre un algoritmo relacionado con la fase de desarrollo, se considera que esto no se debe a la falta de tareas optimizables en esta etapa, sino más bien a las limitaciones de nuestra investigación, los datos se han extraído de un universo limitado de 22 artículos, lo que constituye una muestra todavía pequeña para confirmar o refutar la posibilidad de optimizar tareas en la etapa de desarrollo de software utilizando algoritmos de ML.

Por otro lado, en la etapa de pruebas se identificaron más algoritmos de ML utilizados para optimizar tareas realizadas dentro de esta etapa. Se considera que la información presentada en la Tabla 4 está estrechamente relacionada con el hecho de que una gran parte de los artículos analizados se centra en el análisis de la etapa de pruebas. Esta focalización en la etapa de pruebas puede explicar la mayor variedad de algoritmos utilizados en dicha fase, ya que se le ha prestado una mayor atención en la literatura.

5 Discusión y conclusiones

Respondiendo a las preguntas de investigación planteadas al inicio de este artículo:

PI1- Podemos decir que existen evidencias que demuestren que es factible optimizar tareas realizadas en el proceso de desarrollo de software. En todos los artículos estudiados se presentaron resultados que mejoran en al menos una variable las tareas estudiadas utilizando algoritmos de ML. La tarea Predicción de Fallas en el Software de la etapa de Pruebas fue la tarea más estudiada.

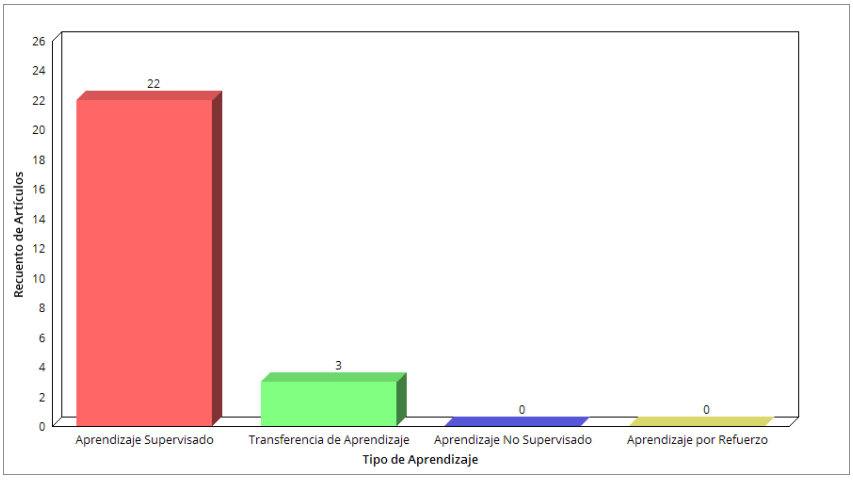

Fig. 6. Cantidad de artículos agrupados según tipo de aprendizaje

PI2- Existen diferentes algoritmos de ML en los que se destaca 'Random Forest', este fue utilizado en 19 de los 22 artículos estudiados para optimización de varias tareas de diferentes etapas de la ingeniería de software. También demostró ser el más eficiente en 10 de las 19 investigaciones que utilizan este algoritmo (Tabla 4).

PI3- Se identificaron 8 herramientas utilizadas para la creación y utilización de los algoritmos de ML siendo las más utilizadas Weka y Scikit-Learn (Figura 7).

PI4- Los algoritmos de ML más utilizados son 'Random Forest', como fuera mencionado anteriormente, seguido de 'Logistic Regression' y 'Support Vector Machines', ambos se mencionan en 15 artículos, otro algoritmo mencionado con frecuencia es el 'Naive Bayes' en un total de 14 artículos.

PI5- Las tareas de Ingeniería de Software más estudiadas son de la etapa de Pruebas, en estas se buscan optimizar la identificación de errores que afectan el desempeño del software.

La elaboración y ejecución de esta investigación permitió crear una Taxonomía de algoritmos de ML utilizados en diversas etapas de la Ingeniería de Software. Esta proporciona una base que permite vislumbrar preguntas para nuevas investigaciones en este campo, especialmente en relación con las etapas del desarrollo de software. Por ejemplo, no se identificaron algoritmos que optimicen tareas específicas comunes de la ingeniería de software como tareas de la etapa de desarrollo. A pesar de que se identificaron una gran variedad de algoritmos de ML, las investigaciones analizadas muestran que el 90% utilizan prioritariamente el algoritmo 'Random Forest', evidenciando una escasa de profundización en el estudio de los demás algoritmos.

El trabajo realizado representa un avance significativo en la comprensión y clasificación de algoritmos de Machine Learning en el contexto de Ingeniería de Software. La creación de una taxonomía específica proporciona una estructura organizada para entender la diversidad de estas técnicas, de relevancia para profesionales e investigadores en el campo.

Además, la identificación de herramientas relevantes para abordar los desafíos del estudio ofrece a los lectores recursos prácticos y aplicables, sirviendo como guías valiosas al enfrentarse a problemas específicos en la implementación de soluciones basadas en ML para la Ingeniería de Software. Quedó demostrado

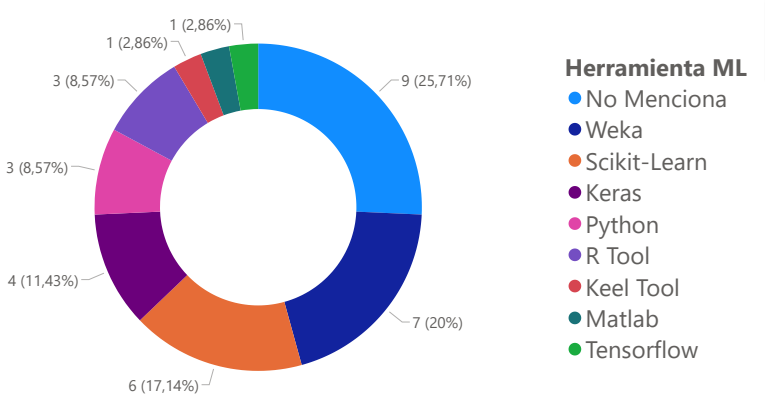

Fig. 7. Herramientas para generación de modelos ML más utilizados

que existen varias tareas de la Ingeniería de software que pueden ser optimizadas usando ML, entre las cuales se incluyen la Clasificación de requisitos de seguridad, Demarcación de requisitos, Detección de *commits* omitibles en Integración Continua, Detección de alcance de comentarios de código y análisis semántico, Detección de inconsistencia en el código, Estimación de esfuerzo para desarrollo del software, Predicción de test inconsistentes, Predicción de requerimientos no funcionales, Predicción de operaciones de refactorización, Predicción de recomendaciones de seguridad del software y otros especificados en la Tabla 3. Estas tareas son claves para el éxito del desarrollo del software, para que la misma pueda ser entregada en tiempo, sin alteración del costo estimado inicial y con la calidad requerida.

Este trabajo también destaca por abrir nuevas perspectivas y direcciones para investigaciones futuras, señalando posibles extensiones y áreas no cubiertas, como por ejemplo: escasos trabajos en la etapa de levantamiento de requisitos, ningún trabajo que cubra tareas de la etapa de desarrollo y el estudio en profundidad del algoritmo 'Random Forest' por ser el más utilizado y que tiene mejor resultado. De esta manera, proporciona una hoja de ruta para las siguientes etapas, que podrían explorar aspectos más detallados de ciertos algoritmos de ML, el desarrollo de nuevas herramientas o enfoques, y la exploración de aplicaciones específicas en Ingeniería de Software que requieran una mayor atención, como citados anteriormente.

6 Sugerencias para futuras investigaciones

A continuación, algunas sugerencias de trabajos futuros.

1. **Profundización de la taxonomía de algoritmos de ML según una fase específica de la Ingeniería de Software.**
   Los resultados de esta investigación revelan escasas investigaciones que aborden tareas de las etapas de Diseño, Mantenimiento e Implementación, además, no se identificaron artículos que aborden tareas relacionadas con la fase de desarrollo. Por lo tanto, sería importante realizar investigaciones que se concentren exclusivamente en una fase particular del ciclo de ingeniería de software, por ejemplo, en

Mejor algoritmo de ML	R	D	E	P	I	M
Random Forest	[3]	[30]		[11], [9], [22], [24], [25], [26]	[2]	[5]
Support Vector Machines	[4]			[7]		
Covolutional Neural Networks(CNN)	[8]			[18]		
J48	[6]			[24]		
BAGGING						[20]
Classification and Regression Trees		[31]				
CNN-KELM				[36]		
Logistic Regression(LR)	[3]					
LR-Decision Trees				[16]		
Multinomial Naive Bayes	[17]					
Naive Bayes				[24]		
ONERULE				[24]		
Sequential Minimal Optimization				[24]		
US-XGBOOST				[12]		

Table 4. Taxonomía de Algoritmos de ML con Fases de Ingeniería de Software - Ingeniería de Requisitos(R), Diseño(D), Desarrollo(E), Pruebas(P), Implementación(I), Mantenimiento(M)

la fase de desarrollo de software se podría implementar algoritmos de ML que optimicen la calidad de escritura del código fuente o las bases de datos.

2. **Explorando el Potencial del Aprendizaje no Supervisado y del Aprendizaje por Refuerzo en la Optimización de Tareas de Ingeniería de Software**
   Otra cuestión que consideramos que no se ha ejemplificado completamente es la utilización de otros métodos de aprendizaje. Gran parte de los artículos que se han analizado emplean el aprendizaje supervisado y han llevado a cabo tareas de clasificación. Por lo tanto, una investigación que se enfoque en otro tipo de aprendizaje, como el aprendizaje no supervisado o el aprendizaje por refuerzo, sería interesante para seguir contribuyendo al conocimiento en el campo de la optimización de tareas de ingeniería de software.
3. **Estudiar el algoritmo 'Random Forest'.**
   El algoritmo 'Random Forest' se destaca por ser el más mencionado y utilizado en los artículos estudiados, además de presentar los mejores resultados en varias tareas estudiadas en diversas etapas de la ingeniería de software, los buenos resultados de este algoritmo señalan el camino que debe ser seguido para plantear nuevos algoritmos de ML, ya sea mejorando el algoritmo 'Random Forest' o creando nuevos algoritmos que tengan características similares a este.

References

1. Software deployment — what are the major obstacles? | by ankita kapoor | backend developers | medium, https://medium.com/backenders-club/software-deployment-what-are-the-major-obstacles-4c53a7423f1
2. Abdalkareem, R., Mujahid, S., Shihab, E.: A machine learning approach to improve the detection of ci skip commits. IEEE Transactions on Software Engineering **47**, 2740–2754 (12 2021). https://doi.org/10.1109/TSE.2020.2967380, https://ieeexplore.ieee.org/document/8961089
3. Abualhaija, S., Arora, C., Sabetzadeh, M., Briand, L.C., Traynor, M.: Automated demarcation of requirements in textual specifications: a machine learning-based approach. Empirical Software Engineering **25**, 5454–5497 (11 2020). https://doi.org/10.1007/s10664-020-09864-1, https://link.springer.com/article/10.1007/s10664-020-09864-1

4. Alashqar, A.M.: Studying the commonalities, mappings and relationships between non-functional requirements using machine learning. Science of Computer Programming **218** (6 2022). https://doi.org/10.1016/j.scico.2022.102806, https://www.sciencedirect.com/science/article/pii/S0167642322000399
5. Aniche, M., Maziero, E., Durelli, R., Durelli, V.H.S.: The effectiveness of supervised machine learning algorithms in predicting software refactoring. IEEE Transactions on Software Engineering **48**, 1432–1450 (4 2022). https://doi.org/10.1109/TSE.2020.3021736, https://ieeexplore.ieee.org/document/9186715
6. Bettaieb, S., Shin, S.Y., Sabetzadeh, M., Briand, L.C., Garceau, M., Meyers, A.: Using machine learning to assist with the selection of security controls during security assessment. Empirical Software Engineering **25**, 2550–2582 (7 2020). https://doi.org/10.1007/s10664-020-09814-x, https://link.springer.com/article/10.1007/s10664-020-09814-x
7. Borovits, N., Kumara, I., Nucci, D.D., Krishnan, P., Palma, S.D., Palomba, F., Tamburri, D.A., van den Heuvel, W.J.: Findici: Using machine learning to detect linguistic inconsistencies between code and natural language descriptions in infrastructure-as-code. Empirical Software Engineering **27** (12 2022). https://doi.org/10.1007/s10664-022-10215-5, https://link.springer.com/article/10.1007/s10664-022-10215-5
8. Casillo, F., Deufemia, V., Gravino, C.: Detecting privacy requirements from user stories with nlp transfer learning models. Information and Software Technology **146** (6 2022). https://doi.org/10.1016/j.infsof.2022.106853, https://www.sciencedirect.com/science/article/pii/S0950584922000246
9. Chen, H., Huang, Y., Liu, Z., Chen, X., Zhou, F., Luo, X.: Automatically detecting the scopes of source code comments. Journal of Systems and Software **153**, 45–63 (7 2019). https://doi.org/10.1016/j.jss.2019.03.010, https://www.sciencedirect.com/science/article/pii/S016412121930055X
10. Cruz-Benito, J., Vishwakarma, S., Martin-Fernandez, F., Faro, I.: Automated source code generation and auto-completion using deep learning: Comparing and discussing current language model-related approaches. AI **2**(1), 1–16 (2021)
11. Elmishali, A., Stern, R., Kalech, M.: An artificial intelligence paradigm for troubleshooting software bugs. Engineering Applications of Artificial Intelligence **69**, 147–156 (3 2018). https://doi.org/10.1016/j.engappai.2017.12.011, https://www.sciencedirect.com/science/article/abs/pii/S0952197617303068
12. Esteves, G., Figueiredo, E., Veloso, A., Viggiato, M., Ziviani, N.: Understanding machine learning software defect predictions. Automated Software Engineering **27**, 369–392 (12 2020). https://doi.org/10.1007/s10515-020-00277-4, https://link.springer.com/article/10.1007/s10515-020-00277-4
13. Flach, P.: MACHINE LEARNING: The Art and Science of Algorithms. Cambridge University Press, New York (2012)
14. Giray, G.: A software engineering perspective on engineering machine learning systems: State of the art and challenges. Journal of Systems and Software **180**(111031) (2021)
15. Jenni, R.: Machine Learning for Programming Languages. Ph.D. thesis, OST Ostschweizer Fachhochschule (2022)
16. Johnson, F., Oluwatobi, O., Folorunso, O., Ojumu, A.V., Quadri, A.: Optimized ensemble machine learning model for software bugs prediction. Innovations in Systems and Software Engineering (3 2022). https://doi.org/10.1007/s11334-022-00506-x, https://link.springer.com/article/10.1007/s11334-022-00506-x
17. Kadebu, P., Sikka, S., Tyagi, R.K., Chiurunge, P.: A classification approach for software requirements towards maintainable security. Scientific African **19** (3 2023). https://doi.org/10.1016/j.sciaf.2022.e01496, https://www.sciencedirect.com/science/article/pii/S2468227622004008
18. Li, Y., Soliman, M., Avgeriou, P.: Identifying self-admitted technical debt in issue tracking systems using machine learning. Empirical Software Engineering **27** (11 2022). https://doi.org/10.1007/s10664-022-10128-3, https://ieeexplore.ieee.org/document/8961089
19. Lopez-Pires, F.: Many-Objective Resource Allocation for Elastic Infrastructures in Overbooked Cloud Computing Datacenters Under Uncertainty. Ph.D. thesis (07 2017)
20. Malhotra, R., Bansal, A.: Investigation of various data analysis techniques to identify change prone parts of an open source software. International Journal of System Assurance Engineering and Management **9**, 401–426 (4 2018). https://doi.org/10.1007/s13198-017-0686-5, https://link.springer.com/article/10.1007/s13198-017-0686-5

21. Malhotra, R., Lata, K.: An empirical study on predictability of software maintainability using imbalanced data. Software Quality Journal **28**, 1581–1614 (12 2020). https://doi.org/10.1007/s11219-020-09525-y, https://link.springer.com/article/10.1007/s11219-020-09525-y
22. Malhotra, R., Meena, S.: Defect prediction model using transfer learning. Soft Computing **26**, 4713–4726 (5 2022). https://doi.org/10.1007/s00500-022-06846-x, https://link.springer.com/article/10.1007/s00500-022-06846-x
23. Nagpal, M.D., Malik, K., Kalia, A.: A comprehensive analysis of requirement engineering utilizing machine learning techniques. Design Engineering pp. 2662–2678 (2021)
24. Oliveira, D., Assunção, W.K., Garcia, A., Fonseca, B., Ribeiro, M.: Developers' perception matters: machine learning to detect developer-sensitive smells. Empirical Software Engineering **27** (12 2022). https://doi.org/10.1007/s10664-022-10234-2, https://link.springer.com/article/10.1007/s10664-022-10234-2
25. Parashar, A., Goyal, R.K., Kaushal, S., Sahana, S.K.: Machine learning approach for software defect prediction using multi-core parallel computing. Automated Software Engineering **29** (11 2022). https://doi.org/10.1007/s10515-022-00340-2, https://link.springer.com/article/10.1007/s10515-022-00340-2
26. Pontillo, V., Palomba, F., Ferrucci, F.: Static test flakiness prediction: How far can we go? Empirical Software Engineering **27** (12 2022). https://doi.org/10.1007/s10664-022-10227-1, https://link.springer.com/article/10.1007/s10664-022-10227-1
27. Prenner, J.A., Robbes, R.: Automatic program repair with openai's codex: Evaluating quixbugs (2021)
28. Pressman, R.S.: Ingeniería del Software: Un enfoque práctico. McGRAW-HILL, New York (2013)
29. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson Education, Harlow (2022)
30. Sarro, F., Moussa, R., Petrozziello, A., Harman, M.: Learning from mistakes: Machine learning enhanced human expert effort estimates. IEEE Transactions on Software Engineering **48**, 1868–1882 (6 2022). https://doi.org/10.1109/TSE.2020.3040793, https://ieeexplore.ieee.org/document/9272884
31. Sehra, S.K., Brar, Y.S., Kaur, N., Sehra, S.S.: Software effort estimation using fahp and weighted kernel lssvm machine. Soft Computing **23**, 10881–10900 (11 2019). https://doi.org/10.1007/s00500-018-3639-2, https://link.springer.com/article/10.1007/s00500-018-3639-2
32. Sommerville, I.: Ingeniería de Software. Pearson Educación, México (2011)
33. Todorov, G.: 25+ fascinating software development statistics about the present and future. https://thrivemyway.com/software-development-statistics/ (Noviembre 2022), [En línea]. [Último acceso: 11 Noviembre 2022]
34. Tsai, C.Y., Taylor, G.W.: Deeprng: Towards deep reinforcement learning-assisted generative testing of software. arXiv preprint arXiv:2201.12602 (2022)
35. Zhang, D., Tsai, J.J.: Machine learning and software engineering. Software Quality Journal **11**(2), 87–119 (2003)
36. Zhu, K., Ying, S., Zhang, N., Zhu, D.: Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network. Journal of Systems and Software **180** (10 2021). https://doi.org/10.1016/j.jss.2021.111026, https://www.sciencedirect.com/science/article/pii/S0164121221001230