Revista Politécnica ISSN 1900-2351 (Impreso), ISSN 2256-5353 (En línea), Año 18, Número 35, páginas 128-139, Enero-Junio 2022

DOI: https://doi.org/10.33571/rpolitec.v18n35a9

MACHINE LEARNING APLICADO AL ANÁLISIS DEL RENDIMIENTO DE DESARROLLOS DE SOFTWARE

Víctor Daniel Gil-Vera[1], Cristian Seguro-Gallego[2]

[1]Ingeniero de Sistemas, Docencia e Investigación, Grupo de Investigación SISCO, Universidad Católica Luis Amigó, victor.gilve@amigo.edu.co
[2]Ingeniero de Sistemas, Docencia e Investigación, Grupo de Investigación SISCO, Universidad Católica Luis Amigó, cristian.segurog@amigo.edu.co

RESUMEN

Las pruebas de rendimiento son determinantes para medir la calidad de los desarrollos de software, ya que permiten identificar aspectos que se deben mejorar en pro de alcanzar la satisfacción del cliente. El objetivo de este trabajo fue identificar la técnica óptima de Machine Learning para predecir si un desarrollo de software cumple o no con los criterios de aceptación del cliente. Se empleó una base de datos de información obtenida en pruebas de rendimiento a servicios web y la métrica de calidad F1-score. Se concluye que, a pesar de que la técnica de *Random Forest* obtuvo el mejor puntaje, no es correcto afirmar que sea la mejor técnica de Machine Learning; la cantidad y la calidad de los datos empleados en el entrenamiento desempeñan un papel de gran importancia, al igual que un procesamiento adecuado de la información.

Palabras clave: Análisis de Resultados, Inteligencia Artificial, Modelado Predictivo, Pruebas de Rendimiento.

Recibido: 22 de febrero de 2022. Aceptado: 22 de abril de 2022
Received: February 22, 2022. Accepted: April 22, 2022

MACHINE LEARNING APPLIED TO THE ANALYSIS OF PERFORMANCE OF SOFTWARE DEVELOPMENTS

ABSTRACT

Performance tests are crucial to measure the quality of software developments, since they allow identifying aspects to be improved in order to achieve customer satisfaction. The objective of this research was to identify the optimal Machine Learning technique to predict whether or not a software development meets the customer's acceptance criteria. A dataset with information obtained from web services performance tests and the F1-score quality metric were used. This paper concludes that, although the Random Forest technique obtained the best score, it is not correct to state that it is the best Machine Learning technique; the quantity and quality of the data used in the training play a very important role, as well as an adequate processing of the information.

Keywords: *Artificial Intelligence, Performance Analysis, Performance Testing, Predictive Modeling.*

Como citar este artículo: Gil-Vera, V., D., & Seguro-Gallego, C. (2022 Machine learning aplicado al análisis del rendimiento de desarrollos de software, 18(35), 128-139. https://doi.org/10.33571/rpolitec.v18n35a9

1. INTRODUCCIÓN

Todas las empresas dedicadas a la fabricación de software o las que tienen un departamento en el que se desarrolla esta actividad, son cada vez más conscientes de que se debe garantizar la construcción de productos con altos estándares de calidad dentro de los cuales se encuentra la eficiencia de desempeño, la cual es definida como: "característica que representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones" [1]. En esta área de la calidad, la definición de los criterios de aceptación, la definición de los escenarios de pruebas y el análisis de los resultados son componentes necesarios para garantizar un alto nivel de calidad en las soluciones. Gracias al Machine Learning (ML), específicamente al aprendizaje supervisado, es posible analizar los resultados de las pruebas de rendimiento.

El objetivo de este trabajo fue identificar la técnica de ML (*Regresión Lineal, Regresión Logística, Árbol de decisión, Maquinas de Vector Soporte, Naive Bayes, Knn, K-Means, Random Forest*) óptima para predecir si un desarrollo de software cumple o no con los criterios de aceptación del cliente. Se realizó un análisis de los resultados obtenidos en la evaluación de las pruebas de rendimiento en servicios web, se construyó un modelo de cada una de las técnicas de ML en el lenguaje de programación Python. Finalmente, se realizó un comparativo para identificar cuál tenía la mayor precisión y la menor tasa de error.

Para la construcción de los modelos se empleó una base de datos de una compañía financiera, las pruebas de rendimiento realizadas se hicieron a servicios web que son soporte de los canales digitales de dicha compañía, la base de datos estaba conformada por ocho variables y por 712.433 registros. A continuación, se describen brevemente cada una de las variables:

- *Elapsed ('VO'):* mide el tiempo transcurrido desde justo antes de enviar la solicitud hasta justo después de que se ha recibido la última parte de la respuesta.
- *responseCode ('V1'):* código del estado de la respuesta a la petición, el cual está definido por el estándar RFC 7231.
- *success ('V2')*: su valor es TRUE si el resultado de la petición es el esperado / FALSE si el resultado de la petición es fallido o si aun siendo exitoso no es el resultado esperado.
- *bytes ('V3'):* cantidad de bytes recibidos.
- *sentBytes ('V4'):* cantidad de bytes enviados.
- *allThreads ('V5'):* todos los hilos.
- *Latency ('V6'):* mide la latencia desde justo antes de enviar la solicitud hasta justo después de recibir la primera parte de la respuesta.
- *Connect ('V7'):* mide el tiempo que se tarda en establecer la conexión, incluido el protocolo de enlace SSL.
- *Result ('V8'):* cumple/no cumple.

Después de entrenar los modelos con los parámetros óptimos haciendo uso de la función GridSearchCV de Python se identificó que el modelo óptimo fue Random Forest, el cual obtuvo un F1-Score de 0.99849. Se priorizó esta métrica sobre las métricas de "Accuracy", "Precisión", "Recall" y "Specifity", ya que las clases no estaban completamente balanceadas, el F1-Score tiene la capacidad de hacer buenas predicciones en los resultados de las pruebas de rendimiento, razón por la cual es más confiable que otras métricas de precisión, sobre todo cuando se presenta un desequilibro en las clases. El principal aporte de esta investigación es facilitar a la industria una herramienta / implementación que madure la evaluación de los resultados de las pruebas realizadas por los analistas dedicados a esta función.

2. ANTECEDENTES

Es indispensable realizar pruebas de rendimiento a los productos de software antes de ser lanzados al mercado [2]. Investigaciones señalan que es baja la cantidad de personas que realizan pruebas de rendimiento, cerca del 50% de desarrolladores dedican menos del 5% de su tiempo a realizar estas pruebas [3], lo que denota una mala práctica, ya que dichas pruebas necesitarían alrededor del 60% del tiempo en el ciclo de desarrollo de software [4]. Las pruebas de rendimiento deben ejecutarse por largos periodos de tiempo para garantizar la calidad en los resultados, el cual se logra en conjunto con una réplica (lo más fiel posible) de los entornos de ejecución de la prueba y de operación del producto [5].

Según [6], en la mayoría de los sistemas a gran escala las fallas obedecen más a problemas de rendimiento que a problemas funcionales. Es por esto que las pruebas de rendimiento juegan un papel importante para

sistemas que son utilizados por millones de personas al tiempo [7]. Diversas investigaciones señalan la importancia de incrementar el presupuesto destinado a medir la calidad de los desarrollos en proyectos de software [8], [9] y [10], ya que esta tarea es de vital importancia para desarrollar productos con que cumplan con los estándares de calidad (requerimientos funcionales) de los usuarios finales.

Hacer pruebas de rendimiento implica evaluar el producto en aspectos multidimensionales como lo son: velocidad, carga, trafico, estrés, entre otros [9]. En [7] plantean que la razón principal para realizar pruebas de rendimiento es encontrar cuellos de botella en el sistema, al igual que se señala en [10], donde adicionalmente mencionan que las pruebas de rendimiento consisten en evaluar que tan eficaz es la respuesta que el sistema le da a un usuario. Un cuello de botella, de acuerdo con [11], es un elemento (lógico o físico) que hace que el rendimiento del desarrollo sea inferior al esperado. El rendimiento de software hace referencia a la capacidad que tiene una solución para completar transacciones, a la velocidad y a la precisión con que lo hace, sin tener en cuenta la cantidad de usuarios que hacen peticiones al sistema [12].

Las pruebas de rendimiento buscan evaluar en un sistema los objetivos de desempeño, por ejemplo: tiempos de respuesta, rendimiento y utilización de recursos. Además, encontrar problemas funcionales que se evidencian bajo cargas de trabajo [13]. Para [14], las pruebas de rendimiento son un recurso que permiten garantizar la estabilidad y confiabilidad de un software. Una prueba de rendimiento requiere de unas acciones y de un conjunto de datos para realizar dichas acciones, durante la ejecución se requiere monitorear variables como la latencia, tiempo de ejecución, rendimiento y uso de recursos; todo con el fin de evitar que se presenten tiempos altos de respuestas, bajo rendimiento y fallas en la aplicación que puedan tener un efecto adverso en la satisfacción del cliente / usuario del producto [15]. Al realizar pruebas de rendimiento se debe tener en cuenta que son costosas, ya que requiere gran cantidad de recursos y tiempo en la evaluación de los resultados obtenidos [16].

Para realizar pruebas de rendimiento y obtener buenos resultados, se requiere la definición de criterios de aceptación o de requisitos no funcionales (RNF), los cuales definen el punto de partida para la evaluación a realizar. En [17] señalan que los RNF son las definiciones de calidad deseada, garantizan el éxito en los proyectos de software y en la mayoría de los casos están mal definidos debido a que prima el enfoque ágil de fabricar entregables rápidamente. Los requisitos no funcionales son todas las limitaciones o condiciones que debe cumplir el desarrollo del sistema [18]. Todos los sistemas cuentan con RNF, pero no siempre están bien definidos y son explícitos, ya sea porque se desconoce la importancia de estos, o porque no se cuenta con las habilidades necesarias para la elicitación de los atributos no funcionales [19].

De acuerdo con [20], la elicitación de requisitos es una de las actividades más críticas, ya que implica un mayor grado de conocimiento y más cuando se trata de los RNF, ya que no hay un consenso sobre técnicas y mecanismos para ello. En esta investigación realizaron un sondeo en el que el 69.4% de los participantes establecieron que los RNF son entregados por las empresas para las que laboran, pero de este número, solo el 11.8 % considera que dichos RNF están bien definidos, mientras el 67.6% y el 20.6% definen los RNF entregados como razonables y como mal definidos respectivamente. Tal como se plantea en [3], medidas comunes en las pruebas de rendimiento son: tiempo de respuesta de las peticiones, rendimiento en cuanto a número de transacciones por unidad de tiempo y uso de recursos disponibles para responder las solicitudes. Según [21], los atributos de calidad son: el uso de recursos, el tiempo de respuesta y el procesamiento de transacciones o rendimiento; además de estos agrega la latencia, la precisión y la perdida de datos.

Por otra parte, también señalan la diferencia entre requisitos no funcionales internos y externos, los primeros definen la capacidad de mantenimiento, la escalabilidad y la capacidad de prueba de software, los externos incluyen el rendimiento, la seguridad y la experiencia del usuario. En [22], afirman que el tiempo de respuesta y el rendimiento, son las métricas en las que más confían los profesionales en la materia, esto debido a que la experiencia les ha enseñado que los problemas de rendimiento salen a la luz cuando el usuario final experimenta tiempos altos de respuesta y altas tasas de latencia. Para pruebas de rendimiento en nube, existe la limitante de que se desconoce acerca del funcionamiento, ya que son cajas negras que están al servicio de los clientes para desplegar sus implementaciones [23]. Otras de las métricas o RNF que podemos medir son la CPU, el uso de memoria, la tasa de errores, la tasa de retraso en las respuestas y el número máximo de errores [24]. Otra propuesta, determina que entre los RNF debería estar la definición de los usuarios óptimos y los usuarios máximos, los cuales son el número de usuarios con los que la aplicación debería tener un buen rendimiento y el número de usuarios con el cuál el funcionamiento de la aplicación quiebra [14].

Los criterios de aceptación o los RNF deben ser definidos para tener un punto en el cual poder determinar si el software desarrollado tiene o no un buen rendimiento [25]. Por otro lado, en [26] indican que esta definición se debería hacer en el transcurso de la operación del software. En un estudio realizado, en donde se evalúan las tres razones principales que llevan al fracaso a un proyecto de software, encontramos que la definición de los RNF tiene una participación del 13% [27]. Esto nos llevaría a cuestionarnos si es eficiente el definir dichos requisitos en la marcha, ya que en ese momento se haría de una forma correctiva y no de una forma preventiva, lo que contradice lo planteado por [26].

En las pruebas de rendimiento comúnmente se cuentan con pruebas de carga, que son pruebas que buscan detectar problemas relacionados con el funcionamiento, debido a la carga de trabajo y a la violación a los RNF relacionados con la calidad bajo carga [28]. También vemos pruebas de estrés que buscan conocer los alcances máximos del sistema bajo prueba [12]. Adicional a estas pruebas se encuentran otros tipos de pruebas como el definido por el mismo autor; la prueba de remojo, resistencia o soak en la que el objetivo es poner el sistema en prueba bajo una carga continúa esperada que busca encontrar diferencias entre el comportamiento del sistema al inicio de la ejecución y mucho tiempo después de estar recibiendo solicitudes.

Otras propuestas, cómo la de [29], recomienda realizar dos fases en las pruebas de rendimiento, una exploratoria en la que se determinan las transacciones o flujos críticos para el rendimiento, y otra en la que se profundiza en aquellas transacciones críticas para el rendimiento con el fin de identificar muchos más problemas en un periodo de tiempo más corto que el que se llevaría en una prueba común. Según [2], el rendimiento en software se puede medir realizando pruebas de carga, o construyendo modelos de rendimiento de las aplicaciones. Los modelos de rendimiento pueden dar sugerencias útiles de los problemas de rendimiento o problemas en la arquitectura, pero están limitados a que no pueden representar la totalidad del sistema y alguno de esos elementos que no es posible representar también pueden aportar al mal rendimiento de la aplicación [30]. Una propuesta de [31], plantea una mezcla entre las pruebas de carga que requieren de mucho tiempo y un análisis de rendimiento por medio de modelos que tienen menor calidad. En un estudio realizado el 36% de los proyectos utilizaron microbenchmarks para pruebas de rendimiento que es otra técnica que permite medir el rendimiento para predecir cómo se comportará el sistema al cambio de entornos. Para lograrlo se deben hacer varias benchmarks o comparativas [32]. Adicionales a las pruebas, se plantea una forma para abordarlas ya que, según el autor de esta idea, se pueden utilizar mecanismos para acortar las pruebas debido a que hay un momento en el que la prueba empieza a repetir resultados y debería tener la capacidad de detenerse sin comprometer la calidad de los resultados [33].

Las pruebas de rendimiento están definidas para medir el grado de calidad o de cumplimiento de la eficiencia de desempeño. Estas pruebas nos ayudan a entregar soluciones que cumplan con unos objetivos propuestos que aportan a la satisfacción del usuario. La mayoría de los autores concuerdan en que los RNF más comunes son los tiempos de respuesta, el rendimiento y el consumo de recursos, adicionalmente proponen otras métricas útiles para medir el rendimiento de los sistemas bajo prueba. Entre los tipos de pruebas de rendimiento aplicadas actualmente, sobresale en la literatura explorada las pruebas de carga, las pruebas de estrés y los modelos de análisis de rendimiento. Para cada uno de estos tipos de pruebas se proponen variantes y se deja claro que los dos primeros tipos demandan mucho tiempo, pero que el tercer tipo si bien no demanda tiempo, sacrifica calidad en los resultados.

3. MÉTODO

Para la construcción de los modelos se empleó una base de datos de una compañía del sector financiero. Las pruebas de rendimiento realizadas se hicieron a servicios web que eran soporte de los canales digitales de la compañía, los resultados de los diferentes servicios probados fueron obtenidos entre noviembre de 2020 y septiembre de 2021. Los criterios de aceptación establecidos para las pruebas de rendimiento se enfocaron en tiempos de respuesta de las solicitudes realizadas al sistema y transacciones por segundo.

En la planeación de las pruebas de rendimiento se definió: el alcance de las pruebas, la descripción general del sistema, riesgos, datos de prueba, configuración de los entornos de prueba y de operación, y las herramientas utilizadas para la prueba. En el alcance de las pruebas se debe delimitar el sistema a probar, indicando que hace parte y qué no de la prueba, es importante que sea clara y no se preste para ambigüedades, ya que es el marco que delimitará el trabajo de las pruebas. En la descripción general del sistema, se debe plasmar de forma general qué hace el sistema y que componentes hacen parte de su flujo; además, es necesario determinar cada uno de los servicios que serán probados con cada uno de sus métodos. Además, se deben identificar los riesgos asociados a las pruebas y se debe evaluar su probabilidad de materialización e

impacto, esta tarea es desarrollada por el equipo de desarrollo junto con el equipo de calidad y el equipo o persona dueña de la iniciativa o proyecto.

Debido a que las pruebas de rendimiento replican un escenario cercano al que el servicio web experimentaría en producción, es necesario garantizar una gran cantidad de datos de prueba que permitan simular gran cantidad de peticiones al servicio en la misma forma como ocurriría en la operación. Para lograr contar con esta cantidad de datos, desde la planeación de las pruebas se debe definir cuál sería el mecanismo para la generación de los datos y en caso de ser necesario su mantenimiento. Adicionalmente, se deben dar los detalles de las configuraciones de los ambientes en los que se desarrollarían las pruebas y el ambiente productivo con el fin de evaluar cuál es su diferencia y determinar desde los recursos los impactos qué puede tener esto en el rendimiento del sistema bajo prueba. Finalmente, se debe definir cuál sería la herramienta o las herramientas de prueba que serán utilizadas para llevar a cabo las pruebas. En este caso, la herramienta que se empleó para las pruebas fue Apache JMeter™.

Existen pruebas de línea base, carga y estrés. En las pruebas de línea base, se busca validar la estabilidad del servicio a probar inyectando únicamente un 10% de los usuarios virtuales o hilos calculados para la concurrencia de las pruebas. La prueba de carga, busca simular lo que se espera en la operación del servicio, busca medir el comportamiento del servicio web bajo una carga esperada. La prueba de estrés tiene como objetivo verificar si el sistema puede soportar una carga mayor a la esperada con el fin de identificar momentos picos de la operación o el crecimiento de los usuarios que hacen uso del servicio y por ende el crecimiento de su tráfico en un lapso de tiempo estimado.

Posteriormente, se procede a implementar la prueba en la herramienta seleccionada. Para la implementación es importante tener en cuenta la adherencia de la herramienta a la solución que se desea probar debido a que no todas las herramientas funcionan en todas las tecnologías en las que se puede implementar servicios web. Otro aspecto importante es que se debe garantizar las buenas prácticas en el desarrollo y el resguardo de la información sensible debido a que por la naturaleza de las pruebas se tiene acceso a información sensible en la mayoría de las soluciones evaluadas. Se realizó la ejecución de cada una de las pruebas definidas en la fase de diseño, cada tipo de prueba se realizó mínimo en dos oportunidades con el fin de generar una mayor calidad en el resultado debido a que se puede comparar si los resultados de cada tipo de prueba están homogéneos en los dos test. Una vez se realizó la ejecución, se obtuvieron los resultados de las pruebas en formato .csv con la siguiente información:

- timeStamp: fecha en formato timestamp en la que se realizó la prueba
- elapsed: mide el tiempo transcurrido desde justo antes de enviar la solicitud hasta justo después de que se ha recibido la última parte de la respuesta.
- label: nombre del servicio al que se le está realizando la petición.
- responseCode: código del estado de la respuesta a la petición. Este código está definido por el estándar RFC 7231.
- responseMessage: corresponde al mensaje del estado de respuesta correspondiente al código de respuesta o respose code.
- threadName: nombre del hilo o del subproceso. La notación nombreSubproceso <a>/<b> donde a y b son números enteros que significan que es el Hilo a de b subprocesos totales.
- datatype: tipo de dato, regularmente siempre el valor de este campo es igual a “text”.
- success: su valor es TRUE si el resultado de la petición es el esperado, y FALSE si el resultado de la petición es fallido o si aun siendo exitoso no es el resultado esperado.
- failureMessage: mensaje del fallo.
- bytes: cantidad de Bytes recibidos.
- sentBytes: cantidad de bytes enviados.
- grpThreads: grupo de Hilos.
- allThreads: todos los Hilos.
- URL: es la Url por la cual se puede hacer el consumo del servicio web.
- Latency: mide la latencia desde justo antes de enviar la solicitud hasta justo después de recibir la primera parte de la respuesta.
- IdleTime: tiempo de inactividad. Normalmente es cero.
- Connect: mide el tiempo que se tarda en establecer la conexión, incluido el protocolo de enlace SSL.

En la preparación de los datos se eliminaron las columnas:

- timeStamp: fecha y hora de la ejecución de la prueba en formato timeStamp, es un formato de hora en segundos desde las cero horas del 1 de enero de 1970 GTM.
- label: trae la información de la etiqueta o nombre personalizado del servicio que se va a probar.
- response messeage: dato en texto que define el responseCode, es decir, contiene la descripción del código de estado de la respuesta.
- Data Type: este valor es “text” para todas las solicitudes, por ser igual en todos los casos se elimina.
- URL: se elimina por ser la misma para todas las solicitudes
- IdleTime: se elimina por ser igual a cero para todas las solicitudes.
- failureMessage: mensajes de error que solo tienen contenido cuando una solicitud falla
- grpThreads: identificador del grupo de hilos
- threadName: identificador del nombre de hilo

Se cambiaron los datos categóricos a representaciones numéricas:
- success
- responseCode: normalmente esta celda contiene códigos de respuesta 200,400,404,500 entre otros que son numéricos, pero en ocasiones presenta contenido en *string* por errores de la herramienta los cuales deben ser eliminados antes de entrenar el modelo.

Cómo lo datos tienen rangos muy diversos en sus variables, se debe usar la normalización de datos por medio de *MinMaxScaler* de *sklearn*. Finalmente, para el entrenamiento del modelo, se agregará una columna de “cumplimiento” donde el resultado es 1 si cumple y 0 si no cumple. Los modelos fueron entrenados en una proporción de 75/25 (75% de los datos para entrenar, y 25% de los datos para test). Gracias a esto se puede observar la precisión que tienen los modelos. Todos los modelos se desarrollaron en Python haciendo uso de las herramientas: Google Colaboratory y el IDE - Spyder. Para conocer los parámetros ideales para cada uno de los modelos, se empleó la función *GridSearchCV*. La Tabla 1 presenta las librerías y los ajustes preliminares realizados a cada uno de los modelos:

Tabla 1. Librerías de Python empleadas

Modelo	Librería Python	Ajustes
Regresión Lineal	LinearRegression	Se ajusta el modelo para predecir si el servicio web cumple o no cumple con los criterios de aceptación de acuerdo a las demás variables. El modelo se crea utilizando mínimos cuadrados ordinarios de Statsmodels.
Arboles de Decisión	DecisionTreeClassifier	Se identifica la profundidad optima del árbol para reducir la varianza y aumentar la capacidad predictiva del modelo (Pruning)
kNN	KNeighborsClassifier	Se identifica la cantidad de vecinos que tienen mejor precisión para aumentar la calidad de las predicciones realizadas por el modelo. Se válida el modelo con las métricas antes descritas.
Regresión Logística	LogisticRegression	Se realiza la predicción en el modelo usando los datos que se designaron para el test. Se válida el modelo con las métricas haciendo uso de la matriz de confusión.
Máquinas de Vector Soporte	SVC	
Naive Bayes	GaussianNB	
K-Means	KMeans	
Random Forest	RandomForestClassifier	

Para evaluar la precisión de los modelos desarrollados se empleó una matriz de confusión, la cual consiste en una representación matricial de los resultados de las predicciones realizadas, ver Figura 1:

Valores Actuales		Valores Predichos	
		No Cumple	Cumple
	No Cumple	(TN) Verdaderos Negativos	(FP) Falsos Positivos
	Cumple	(FN) Falsos Negativos	(TP) Verdaderos Positivos

Figura 1. Matriz de confusión

Donde:

TN: Verdadero Negativo, valores que en la predicción fueron negativos o de la clase no cumple y en los valores actuales también lo eran.

TP: Verdadero Positivo, valores que en la predicción fueron positivos o de la clase cumple y en los valores actuales también lo eran.

FN: Falso Negativo, Valores que en la predicción fueron negativos o de la clase no cumple y en los valores actuales no lo eran.

FP: Falso Positivo, valores que en la predicción fueron positivos o de la clase cumple y en los valores actuales no lo eran.

A partir de los valores de la matriz de confusión se calcularon las métricas que se presentan a continuación:

Exactitud o Accuracy: porcentaje de predicciones correctas

$$\mathbf{Accuracy} = \frac{(TP + TN)}{Total} \quad (1)$$

Sensibilidad, exhaustividad o Recall: Porcentaje de casos positivos detectados

$$\mathbf{Recall} = \frac{TP}{(TP + FN)} \quad (2)$$

Especificidad o Specifity: porcentaje de casos negativos detectados

$$\mathbf{Specifity} = \frac{TN}{(TN + FP)} \quad (3)$$

Precisión o Precision: porcentaje de predicciones positivas correctas

$$\mathbf{Precision} = \frac{TP}{(TP + FP)} \quad (4)$$

F1 Score: Medida armónica de la precisión y exhaustividad, 1 denota exhaustividad y precisión perfectas.

$$\boldsymbol{F1 - Score} = 2\left(\frac{precisión * Exhaustividad}{precisión + Exhaustividad}\right) \quad (5)$$

Curva de características operativas del receptor (ROC): Donde AUC=1 es lo ideal, AUC =0.5 el modelo no tiene capacidad para distinguir entre las clases y un AUC = 0 quiere decir que la predicción está correspondiendo las clases.

$$\mathbf{1 - especificidad} = \frac{FP}{(TN + FP)} \qquad (6)$$

4. RESULTADOS

En el siguiente repositorio de GitHub está disponible todo el contenido del código del aplicativo que permite evaluar el desempeño de las pruebas de rendimiento usadas para entrenar los modelos:

https://onx.la/99887

Los resultados de las comparaciones entre los modelos seleccionados se encuentran disponibles en:

https://onx.la/082d6

Los datos para entrenar y comparar los resultados de los modelos fueron normalizados usando la función *MinMaxScaler* de *sklearn*, la Figura 2 presenta el diagrama de cajas y bigotes después de la normalización:

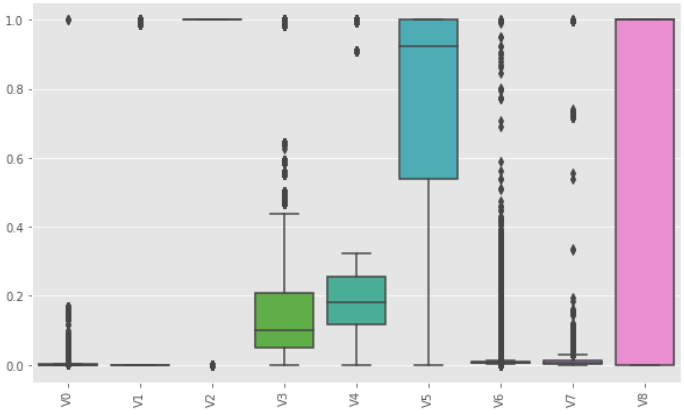

Figura 2. Diagrama de cajas y bigotes datos normalizados

En la siguiente tabla se puede ver el resumen de las métricas tomadas en cada uno de los modelos evaluados, estas métricas están ordenadas desde el modelo con mejor puntuación F1 al modelo que presenta una puntuación menor.

Tabla 2. Resultados del entrenamiento

Modelo	TN	FP	FN	TP	Accuracy	Precision	Recall	Especificidad	F1 Score
Random Forest	61612	236	115	116140	0,998	0,997	0,999	0,996	0,998
KNN	61589	259	121	116134	0,997	0,997	0,998	0,995	0,998
MVS	59463	2385	1078	115177	0,980	0,979	0,990	0,961	0,985
Regresión Logística	52701	9147	2253	114002	0,935	0,925	0,980	0,852	0,952
Arboles de decisión	46636	15212	762	115493	0,910	0,883	0,993	0,754	0,935
Regresión Lineal	8738	53110	503	115752	0,698	0,685	0,995	0,141	0,811
Naive Bayes	7701	54147	336	115919	0,694	0,681	0,997	0,124	0,809
K-Means	229855	17396	346962	118196	0,488	0,871	0,254	0,929	0,393

Los resultados permiten identificar que el modelo que mejor realiza la predicción del resultado dentro de las pruebas de rendimiento es “Random Forest”, debido a que el valor de la métrica F1-Score fue de 0.99849, esta métrica tiene la capacidad de hacer buenas predicciones en los resultados de las pruebas de rendimiento, en este caso se priorizo por encima de la métrica “Accuracy”, ya que las clases de la base de datos no estaban balanceadas. La Figura 3 presenta la curva ROC / AUC, se puede apreciar que tiene un ajuste adecuado en la esquina superior izquierda. La Figura 4 presenta la matriz de confusión

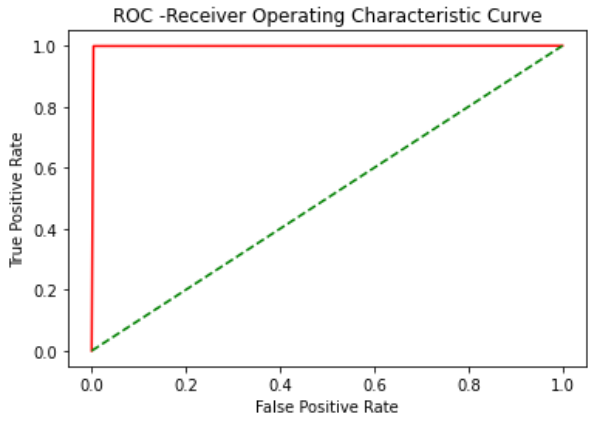

Figura 3. Curva ROC / AUC

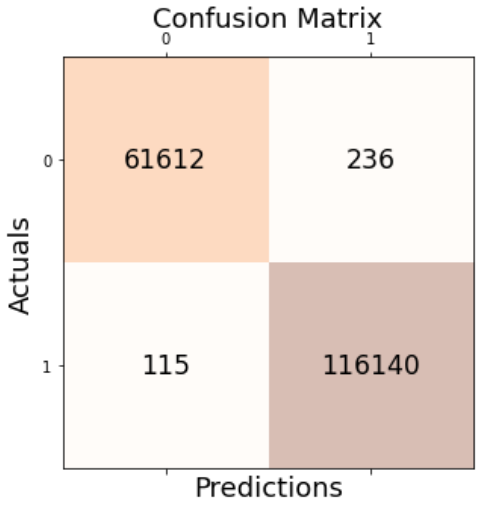

Figura 4. Matriz de confusión

Otros modelos que tuvieron un buen desempeño fueron el k - Vecinos Próximos (K-NN) con una puntuación F1 de 0.99837 y el de máquinas de vectores de soporte (MVS) con un F1-Score de 0.98519. La Regresión logística también mostró una tasa menor que los modelos anteriormente mencionados para detectar los resultados que no cumplen con los criterios. Sin embargo, la precisión estuvo en 0.92572, lo que muestra que la capacidad para hacer predicciones positivas correctas es inferior que los modelos anteriores. Los modelos de Regresión Lineal y Naive Bayes fueron los modelos con menor capacidad para identificar los resultados que no cumplen con los criterios de aceptación, para estos dos modelos esta métrica fue inferior a 0.14, lo que los hace menos eficientes a la hora de hacer predicciones. Finalmente, el modelo menos eficiente de todos para predecir si los resultados cumplen o no con los criterios de aceptación fue el de K-Means.

5. CONCLUSIONES

A pesar de que la técnica de Random Forest obtuvo el mejor puntaje para identificar la técnica óptima de ML para predecir si un desarrollo de software cumple o no con los criterios de aceptación del cliente, no es correcto afirmar que sea la mejor entre todas las técnicas de ML. El objetivo de las técnicas de ML es que aprendan patrones que generalicen bien los datos que no fueron analizados en lugar de memorizar datos que aprendieron durante el entrenamiento; no es correcto afirmar con certeza que una técnica sea mejor que otras, la cantidad de información y el objetivo que tenga el investigador desempeñan un papel importante. Es necesario evaluar todas las métricas de precisión para decidir cuál es la mejor y no solo centrarse en “Accuracy”. Es necesario observar los modelos que se separen más del caso aleatorio, y no solo fiarse de precisiones altas, ya que es posible que las clases estén desbalanceadas y se presenten problemas de sub-entrenamiento o sobre-entrenamiento. Futuras investigaciones pueden enfocarse en expandir los límites y encontrar la forma de hacer que los criterios de aceptación puedan ser paramétricos y permitir evaluar no solo los resultados particulares de un proyecto, si no, de cualquier otra iniciativa que esté usando las pruebas de rendimiento para medir la calidad de sus desarrollos y garantizar desde las fases previas a la salida a producción, la satisfacción del cliente final en cuanto al rendimiento se refiere.

6. REFERENCIAS BIBLIOGRÁFICAS

[1] ISO/IEC. (2011). BSI Standards Publication Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. BSI Standards Publication. https://www.iso.org/standard/35733.html

[2] Apte, V., Devidas, T. V. S. V., Akhilesh, G., & Anshul, K. (2017). AutoPerf : Automated Load Testing and Resource Usage Profiling of Multi-Tier Internet Applications. Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, 115–126. https://doi.org/10.1145/3030207.3030222

[3] Bezemer, C., Eismann, S., Ferme, V., Grohmann, J., Heinrich, R., Jamshidi, P., Shang, W., Hoorn, A. Van, Villavicencio, M., Walter, J., & Willnecker, F. (2019). How is Performance Addressed in DevOps? A Survey on Industrial Practices. 45–50. https://doi.org/10.1145/3297663.3309672

[4] Mohammad, R., & Hamad, H. (2019). Scalable Quality and Testing Lab (SQTL): Mission- Critical Applications Testing. 2019 International Conference on Computer and Information Sciences (ICCIS), 1–7. https://doi.org/10.1109/ICCISci.2019.8716404

[5] Arif, M.M., Shang, W. & Shihab, E. Empirical study on the discrepancy between performance testing results from virtual and physical environments. Empir Software Eng 23, 1490–1518 (2018). https://doi.org/10.1007/s10664-017-9553-x

[6] Syer, M.D., Shang, W., Jiang, Z.M. et al. Continuous validation of performance test workloads. Autom Softw Eng 24, 189–231 (2017). https://doi.org/10.1007/s10515-016-0196-8

[7] Lenka, R. K., & Dey, M. R. (2018). Performance and Load Testing: Tools and Challenges. 2257–2261. https://doi.org/10.1109/ICRIEECE44171.2018.9009338

[8] Oriol, M. (2020). Systematic Literature study on estimation and prioritization of quality requirements in software development. June, 24–27. https://doi.org/10.23919/CISTI49556.2020.9140957

[9] Pradeep, S., & Sharma, Y. K. (2019). A Pragmatic Evaluation of Stress and Performance Testing Technologies for Web Based Applications. 2019 Amity International Conference on Artificial Intelligence (AICAI), 399–403. https://doi.org/10.1109/AICAI.2019.8701327

[10] Silva, Lady. (2020). Model Driven Engineering for Performance Testing in Mobile Applications. https://doi.org/10.1109/SEEDA-CECNSM49515.2020.9221828

[11] Kaplan, A., & Haenlein, M. (2019). Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence. Business Horizons, 62(1), 15–25. https://doi.org/10.1016/j.bushor.2018.08.004

[12] Jacob, A. (2018). Scrutiny on Various Approaches of Software Performance Testing Tools. Iceca, 509–515. https://doi.org/10.1109/ICECA.2018.8474876

[13] Moghadam, M. H., Saadatmand, M., Borg, M., & Bohlin, M. (2020). Poster: Performance Testing Driven by Reinforcement Learning. 402–405. https://doi.org/10.1109/ICST46399.2020.00048

[14] Li, H., Li, X., Wang, H., Zhang, J., & Jiang, Z. (2019). Research on Cloud Performance Testing Model. 2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE), 2015, 179–183. https://doi.org/10.1109/HASE.2019.00035

[15] Ahmad, T., Ashraf, A., Truscan, D., Domi, A., & Porres, I. (2020). Using Deep Reinforcement Learning for Exploratory Performance Testing of Software Systems with Multi-Dimensional Input Spaces. 8. https://doi.org/10.1109/ACCESS.2020.3033888

[16] Postolski, I., Braberman, V., Garbervetsky, D., & Uchitel, S. (2019). Simulator-Based Diff-Time Performance Testing. 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), 81–84. https://doi.org/10.1109/ICSE-NIER.2019.00029

[17] Lopez, L., Mart, S., Vollmer, A. M., Rodr, P., Franch, X., Oivo, M., Mart, S., Vollmer, A. M., Franch, X., Karhapää, P., Lopez, L., Burgués, X., Vollmer, A. M., Rodríguez, P., & Franch, X. (2019). Management of quality requirements in agile and rapid software development: a systematic mapping study Woubshet Behutiye Martínez-Fernández. Information and Software Technology, 106225. https://doi.org/10.1016/j.infsof.2019.106225

[18] Cares, C. (2014). Top Ten NFR to Survive in the “Brave New World” of E-government Applications. 2014 9th Iberian Conference on Information Systems and Technologies (CISTI), 6. https://doi.org/10.1109/CISTI.2014.6876999

[19] Silva, A., & Barroso, J. (2016). A survey about the situation of the elicitation of non-functional requirements. 11th Iberian Conference on Information Systems and Technologies (CISTI). https://doi.org/10.1109/CISTI.2016.7521427

[20] Silva, A., Fortaleza, U. De, Albuquerque, A. B., Fortaleza, U. De, & Barroso, J. (2016). A Process for Creating the Elicitation Guide of Non-functional Requirements A Process for Creating the Elicitation Guide of Non- Functional Requirements. January. https://doi.org/10.1007/978-3-319-33622-0

[21] Yu, L., Alégroth, E., Chatzipetrou, P., & Gorschek, T. (2020). Utilising CI environment for efficient and effective testing of NFRs ☆. 117(May 2019). https://doi.org/10.1016/j.infsof.2019.106199

[22] Hasnain, M. (2019). An Efficient Performance Testing of Web Services. 2019 22nd International Multitopic Conference (INMIC), 1–8. https://doi.org/10.1109/INMIC48123.2019.9022763

[23] He, S., Manns, G., Saunders, J., Wang, W., Pollock, L., & Soffa, M. Lou. (2019). A Statistics-Based Performance Testing Methodology for Cloud Applications: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 188–199. https://doi.org/10.6084/m9.figshare.7749356

[24] Shariff, S. M., Li, H., & Bezemer, C. (2019). Improving the Testing Efficiency of Selenium-based Load Tests. 14th International Workshop on Automation of Software Test, 14–20. https://doi.org/10.1109/AST.2019.00008

[25] Alshazly, A. A., Elnainay, M. Y., El-zoghabi, A. A., & Abougabal, M. S. (2020). A cloud software life cycle process (CSLCP) model. Ain Shams Engineering Journal. https://doi.org/10.1016/j.asej.2020.11.004

[26] Bhowmik, T., & Do, A. Q. (2018). Refinement and Resolution of Just-in-Time Requirements in Open Source Software and a Closer Look into Non-Functional Requirements. Journal of Industrial Information Integration. https://doi.org/10.1016/j.jii.2018.03.001

[27] Flores-Rios, B. L., & Pino, F. J. (2018). Non-functional requirements elicitation based on stakeholders’ s knowledge management. 26, 142–156. https://doi.org/10.22395/rium.v17n32a8.

[28] Jiang, Z. M., & Hassan, A. E. (2015). A Survey on Load Testing of Large-Scale Software Systems. 5589(2), 1–32. https://doi.org/10.1109/TSE.2015.2445340

[29] Ayala-rivera, V., Murphy, J., Portillo-dominguez, A. O., Darisa, A., & Kaczmarski, M. (2018). One Size Does Not Fit All: In-Test Workload Adaptation for Performance Testing of Enterprise Applications. ACM/SPEC International Conference on Performance Engineering, December 2010, 211–222. https://doi.org/10.1145/3184407.3184418

[30] Moghadam, M. H., Saadatmand, M., Borg, M., Bohlin, M., & Lisper, B. (2019). Machine Learning to Guide Performance Testing: An Autonomous Test Framework. 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 164–167. https://doi.org/10.1109/ICSTW.2019.00046

[31] Aichernig, B. K., Bauerst¨atter, P., J¨obstl, E., Kann, S., Journal, S. Q., Koroˇsec, R., Krenn, W., Mateis, C., Schlick, R., & Schumi, R. (2019). Learning and statistical model checking of system response times. Software Quality Journal. https://doi.org/10.1007/s11219-018-9432-8

[32] Leitner, P., & Bezemer, C. (2017). An Exploratory Study of the State of Practice of Performance Testing in Java-Based Open Source Projects. Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering - ICPE, 373–384. https://doi.org/10.1145/3030207.3030213

[33] AlGhamdi, H. M., Bezemer, C., Shang, W., Hassan, A. E., & Flora, P. (2020). Towards reducing the time needed for load testing. Journal of Software: Evolution and Process, April, 1–17. https://doi.org/10.1002/smr.2276