Técnicas de aprendizaje automático para mejorar el rendimiento de aplicaciones web: un mapeo sistemático de literatura

Jean Carlo Zúñiga-Madrigal, Christian Quesada-López, Marcelo Jenkins, Alexandra Martínez

Universidad de Costa Rica, San Pedro, Costa Rica
{jean.zunigamadrigal, cristian.quesadalopez, marcelo.jenkins, alexandra.martinez}@ucr.ac.cr

Resumen. La proliferación de las aplicaciones web ha generado una cultura cada vez más demandante por parte de los usuarios. Se estima que un usuario promedio espera solo dos segundos antes de abandonar una aplicación web, si esta no responde dentro de ese lapso. Por esta razón, la creación de sitios web eficientes se ha convertido en una prioridad para los desarrolladores. El objetivo de este estudio es identificar las principales técnicas de aprendizaje automático utilizadas para mejorar el rendimiento de las aplicaciones web, sus indicadores de desempeño, usos, y capas de aplicación. Para esto se realizó un mapeo sistemático de literatura que identificó 37 estudios primarios. Los resultados indican que los árboles de decisión son el algoritmo más explorado para mejorar el rendimiento de las aplicaciones, seguido por las redes neuronales y las máquinas de soporte vectorial. Los indicadores más utilizados para medir el rendimiento de las aplicaciones web fueron el uso de caché, el tiempo del primer *byte* y el tiempo de carga de página. El principal uso de aprendizaje automático reportado está en la predicción efectiva de futuras acciones de un usuario durante su interacción con una aplicación web para así pre cargar recursos necesarios en memoria antes que sean requeridos. Las capas de infraestructura y *back-end* fueron las que más reportaron el uso de aprendizaje automático para mejorar el rendimiento.

Keywords: aprendizaje automático, rendimiento, aplicaciones web, mapeo sistemático

1 Introducción

El aumento exponencial de aplicaciones en Internet es evidente si analizamos las páginas indexadas por el motor de búsqueda Google: su primer índice en 1996 contenía alrededor de 50 millones de aplicaciones, mientras que en el 2011, la cantidad de aplicaciones indexadas era de alrededor de tres trillones [1]. De la mano de este crecimiento en la cantidad de sitios web, ha habido un aumento en la cantidad de usuarios que navegan en Internet diariamente. Es por esto que resulta relevante analizar alternativas para mejorar el rendimiento que ofrecen

las aplicaciones web, de manera que se ajuste a las necesidades de la creciente demanda de usuarios que las frecuentan.

La motivación de este mapeo de literatura es identificar técnicas del área de aprendizaje automático que se hayan usado con el fin de mejorar el rendimiento de aplicaciones web, considerando sus indicadores de desempeño y las capas de la aplicación donde se implementaron dichas técnicas. Emplearemos este conocimiento en nuestros trabajos, actualmente muchas compañías de desarrollo de *software* se dedican a crear aplicaciones web para sus clientes. Entregar productos con buen rendimiento es importante, por lo que explorar el aprendizaje automático como medio para lograrlo es una alternativa que quisimos explorar.

Para alcanzar el objetivo planteado, se desarrolló un mapeo sistemático de literatura que buscaba responder las siguientes preguntas de investigación: ¿Cuáles técnicas de aprendizaje automático se han utilizado para mejorar el rendimiento de las aplicaciones web? (RQ1), ¿Cuáles indicadores se han usado para medir el rendimiento de aplicaciones web? (RQ2), ¿En cuáles capas de aplicación se han usado las técnicas de aprendizaje automático que mejoran el rendimiento? (RQ3).

El resto del artículo está estructurado de la siguiente manera: la sección 2 presenta el marco teórico con los conceptos que sustentan esta investigación, luego la sección 3 describe el trabajo relacionado en el área de rendimiento en sitios web, la sección 4 discute la metodología que se utilizó para desarrollar esta investigación, el análisis de los resultados se muestra en la sección 5, y finalmente, las conclusiones de esta revisión están en la sección 6.

2 Marco teórico

En esta sección se definen conceptos relacionados al aprendizaje automático y a las aplicaciones web. Estos conceptos son los que sustentan el mapeo realizado.

Aplicaciones web

Una aplicación web es un sitio en línea que puede ser accedido por usuarios a través de un navegador web [2]. Existen distintos tipos de aplicaciones web, por ejemplo, para el marco de desarrollo de Microsoft: *Web Forms*, *Web Forms* con *Asynchronous JavaScript and XML* (AJAX), *Web Forms* con controles *Silverlight*, *Model-View-Controller* (MVC) y aplicaciones *Dynamic Data* [2].

Independientemente de la arquitectura con la que se construya una aplicación web, esta debe contar con una clara división entre el cliente y el servidor [2]. En el cliente es donde se muestran las vistas de la aplicación web, las cuales son accedidas mediante navegadores web que obtienen y parsean los recursos que necesarios por la aplicación, como por ejemplo los archivos *HyperText Markup Language* (HTML) que contienen la estructura del sitio dispuestas en etiquetas, las hojas de estilo CSS (*Cascading Style Sheets*), y los archivos JavaScript que permiten introducir dinamismo en la capa del cliente para la interacción con los usuarios.

El primer paso a seguir por un navegador web es crear un árbol llamado *Domain Object Model* (DOM) para la estructura HTML y otro árbol de estilos para los archivos CSS. Durante este proceso, el JavaScript es interpretado y

ejecutado según aparezca en el orden de parseo de archivos. Al ejecutar estos archivos, se introducen nuevos cambios a la estructura del DOM, dependiendo de las reglas desarrolladas en el código ejecutado.

Por otro parte, en el lado del servidor es en donde ocurre el procesamiento que resuelve las necesidades de los usuarios. Siguiendo con el caso del marco de trabajo provisto por Microsoft, un ejemplo de servidor es el *Internet Information Services* (IIS), herramienta que facilita el despliegue de aplicaciones web al proveer servicios necesarios para servir contenido web a través del protocolo de comunicación *Hypertext Transfer Protocol* (HTTP) [2].

Aprendizaje automático

El aprendizaje automático es el proceso de encontrar, de forma automática, patrones significativos en grandes conjuntos de datos [3]. Este tipo de aprendizaje se utiliza en tareas relacionadas al reconocimiento, diagnóstico, planeamiento y predicción de salidas. La aplicación de aprendizaje automático en sistema computacionales provoca cambios en su estado actual, o inclusive la creación de nuevos sistemas, a raíz de la ejecución de las tareas asociadas al aprendizaje. Shalev-Shwartz y Ben-Dav [3] mencionan distintos tipos de aprendizaje automático, se listan a continuación:

- Supervisado y no supervisado: se denomina aprendizaje supervisado al que pretende formular predicciones tomando como insumos características y etiquetas provistas por quien entrene al algoritmo. Estos insumos no son sino las preguntas y respuestas con las que trabajará el algoritmo de aprendizaje automático. Por otro lado, el aprendizaje no supervisado no cuenta con ningún insumo de entre los descritos anteriormente, la finalidad es que sea el propio algoritmo quien agrupe características comunes que encuentre en la colección de datos con la que se entrena [3].
- Activo y pasivo: la forma pasiva de aprendizaje amerita la interacción continua con el ambiente de aprendizaje, mientras que la forma pasiva se basa en la constante observación resultante de los cambios aplicados al ambiente de aprendizaje [3].
- En línea y por lotes: el aprendizaje en línea ocurre cuando un sistema toma decisiones basadas en su contexto actual, con el riesgo de cometer errores, pero que eventualmente serán cada vez menores hasta contar con un porcentaje de aciertos mayor, a raíz de la experiencia recabada. Por otro lado, el aprendizaje por lotes sucede cuando un sistema cuenta con una colección vasta de datos con los cuales trabajar para llegar al nivel de aprendizaje deseado [3].

Para este mapeo se utilizó la taxonomía creada por Shalev-Shwartz y Ben-Dav [3]. En síntesis, esta taxonomía expone las siguientes categorías: regresión, clasificación, agrupamiento, reducción de la dimensionalidad, redes neuronales, algoritmos probabilísticos, algoritmos heurísticos, y otros enfoques. A continuación se detallan estas familias de algoritmos de aprendizaje:

- Regresión: es un modelo de aprendizaje supervisado que abarca los algoritmos que buscan la relación entre variables. Dentro de los tipos de algoritmos

que se listan bajo esta categoría se mencionan la Regresión lineal simple y la Regresión lineal múltiple.
- Clasificación: modelo de aprendizaje supervisado, los algoritmos que califican dentro de esta categoría son los que reciben una o más entradas e intentan predecir el valor de una o más salidas. Algunos de los algoritmos dentro de esta categoría son las Máquinas de soporte vectorial, Árboles de decisión y Naïve Bayes.
- Agrupamiento: modelo utilizado para el análisis exploratorio de datos, intenta ubicar grupos significativos de información en grandes conjuntos de datos dados puntos de información como guía para así agrupar la información que comparte características en común. Entre los algoritmos que pertenecen a esta clasificación están: *K-means* y Agrupamiento jerárquico.
- Reducción de la dimensionalidad: este modelo busca tomar datos de un espacio dimensional que está a un alto nivel y mapearlo a otro espacio cuya dimensionalidad sea más baja. Entre las razones para completar esta tarea se pueden mencionar el reducir el costo computacional de operar en un espacio dimensional alto, mejorar la generalización del algoritmo de aprendizaje automático que opera el espacio dimensional, y finalmente, dar una mejor interpretación a la información.
- Redes neuronales: se trata de un modelo computacional que se inspira en el comportamiento de las neuronas en el cerebro humano. En el contexto de la computación, una red neuronal es un grafo cuyos nodos corresponden a las neuronas, y las aristas representan las conexiones entre neuronas.
- Híbridos: se refiere a los algoritmos que combinan dos o más algoritmos para alcanzar un fin.
- Algoritmos probabilísticos: clasificación que contempla algoritmos que basan los resultados de la resolución a algún problema en cálculos probabilísticos.
- Otros enfoques: esta es una clasificación creada para los algoritmos de aprendizaje automático que no se ajustan a otras categorías de la taxonomía propuesta por Shalev-Shwartz y Ben-Dav.

3 Trabajo relacionado

No se encontraron en la literatura trabajos previos que abordaran específicamente el uso de técnicas de aprendizaje automático para mejorar el rendimiento de aplicaciones web. Sin embargo, se encontraron trabajos que han comparado y caracterizado algoritmos de aprendizaje automático, que resultan importantes de conocer para efectos de esta investigación. Estos trabajos se describen a continuación.

El estudio de Kazhmaganbetova et al. [4] evalúa técnicas de aprendizaje automático para garantizar una buena comunicación entre clientes y servidores en términos de establecimiento de protocolos seguros, tráfico inusual recibido, y cargas transportadas de un extremo a otro. El principal aporte de este estudio fue su enfoque en la predicción de anomalías en el tamaño de los paquetes de datos que se envían entre el cliente y la aplicación web, con un rendimiento de 23,5 según el valor de Error Cuadrático Medio, métrica utilizada durante la experimentación realizada.

Algunas herramientas han sido propuestas para monitorear el rendimiento de las aplicaciones web. El estudio secundario de Pradeep y Kumar [5] hace una recopilación de tales herramientas, entre ellas: *Apache Jmeter*, *Selenium*, *Cucumber* y *Web Test*. Dicho estudio presenta herramientas y código libre en general, que permita hacer entregas de productos de *software* más depurados y libres de errores. El aporte más valioso de este estudio está en la experimentación realizada con las diferentes herramientas estudiadas. El *framework Locust*, desarrollado en *Python*, es la herramienta que reportó mejores tiempos en ejecución de casos de prueba, con un promedio de 1,6966 segundos en cinco pruebas realizadas, superando a *HULK Analyzer* que promedió 1,8158 segundos, y a *Apache Jmeter* que promedió 2,1068 segundos, al realizar las mismas cinco pruebas.

También se han comparado los resultados de realizar pruebas de rendimiento automáticas *versus* manuales, este es el caso del estudio de Arslan et al. [6] donde evalúan indicadores de rendimiento, tales como consumo de recursos, latencia, utilización de memoria, entre otros más, utilizando pruebas automatizadas y manuales. Este estudio aportó importantes conclusiones sobre la ineficiencia de las pruebas manuales en comparación con las pruebas automáticas, esta conclusión se basa en características como el tiempo de ejecución de las pruebas y lo propenso de estas a fallar debido a la gran cantidad de datos necesarios para ejercitar las técnicas de aprendizaje automático.

La contribución del presente mapeo de literatura es aportar información sintetizada y relevante en un contexto novedoso como lo es el rendimiento de las aplicaciones web. La información ofrecida abarca indicadores que miden el rendimiento de las aplicaciones, así como las capas de aplicación en donde se ha utilizado aprendizaje automático.

4 Metodología

Para cumplir con el objetivo del estudio, se desarrolló un mapeo sistemático de literatura con base en las pautas de Petersen [7] y las recomendaciones de Kitchenham [8] para la conducción de este tipo de estudios secundarios. Esta sección detalla el protocolo de investigación seguido para obtener los resultados expuestos en este estudio.

El objetivo del estudio, formulado con la plantilla de objetivos *Goal Question Metric* (GQM) [9], fue analizar la mejora de rendimiento a partir de técnicas de aprendizaje automático con el propósito de caracterizarlos con respecto a indicadores y áreas de aplicación, desde el punto de vista de un investigador en el contexto de aplicaciones web.

4.1 Estrategia de búsqueda y proceso de selección de estudios

El protocolo de investigación se desarrolló durante el 2019, durante el primer semestre se ubicaron los artículos de donde se obtuvo la información para el mapeo, mientras que el segundo semestre se dedicó a la redacción del estudio.

El protocolo de investigación comenzó con una búsqueda exploratoria para encontrar artículos de investigación que sostuvieron el objetivo planteado, estos hallazgos se tomaron como artículos de control para guiar el resto de la búsqueda. Los temas que abordan los artículos de control seleccionados son la distribución

de peticiones de usuario entre servidores [S02], predicción web basada en las acciones de los usuarios [S03] [S04] y la optimización de aplicaciones web en dispositivos distintos a los de escritorio [S05].

A partir del conjunto de artículos de control se concibió la cadena de búsqueda basada en el modelo PICO (Población, Intervención, Comparación, Salidas) [10], la cual buscó por términos clave en el título y resumen de estudios primarios dentro de los repositorios científicos escogidos: *Scopus*, *IEEE Xplore*, y *Web of Science*. Después de un proceso iterativo de refinamiento, esta fue la cadena de búsqueda final:

("performance" OR "optim*") AND ("web") AND ("load*" OR "render*" OR "display*") AND ("machine learning" OR "artificial Intelligence")

Tal y como se muestra en la Figura 1 La cantidad de estudios primarios recuperada para cada repositorio de búsqueda fue de 474, con la siguiente distribución: 265 estudios en *Scopus*, 44 en *IEEE Xplore* y 165 en *Web of Science*. De este conjunto se eliminaron los estudios duplicados y se aplicaron los criterios de inclusión y exclusión (E/I) para hallar el conjunto final con el que se elaboraron los resultados de la investigación.

El proceso de E/I se hizo con base en el título y el resumen de los artículos. Se incluyeron solo (I1) artículos que abordaron el rendimiento de aplicaciones web mediante técnicas de aprendizaje automático y (I2) estudios primarios. Luego, se excluyeron publicaciones que (E1) estuvieran redactados en un idioma distinto al inglés, y (E2) que no tuvieran acceso al texto completo. El conjunto final de estudios con los que se trabajó luego de terminado el protocolo de selección fue de 37 artículos primarios, la lista completa con sus respectivas referencias está disponible siguiendo este hipervínculo: https://bit.ly/36eYdgV.

4.2 Evaluación de calidad

La evaluación de la calidad refleja el nivel de detalle ofrecido por cada artículo para los aspectos estudiados. Se plantearon las siguientes preguntas de calidad: ¿hubo mejora medible en el rendimiento de la aplicación web al emplear aprendizaje automático?, ¿el artículo señaló el uso de aprendizaje automático como el principal factor de la mejora de rendimiento?, ¿el artículo detalló la metodología o estrategia seguida para aplicar el aprendizaje automático dentro de la aplicación web? y ¿se hizo alusión a indicadores de rendimiento en el estudio? La puntuación se realizó con una escala de 0 a 1, donde 0 = No cumple en lo absoluto, 0,5 = Cumple parcialmente, 1 = Cumple totalmente.

Luego de evaluada la calidad se observa que el valor más alto fue alcanzado por diez artículos [S04], [S05], [S06], [S07], [S08], [S09], [S10], [S11], [S12], [S13]. Por otro lado, hubo solo tres que obtuvieron la peor calificación [S14], [S15], [S16]. El detalle de la evaluación de calidad se encuentra disponible siguiendo este hipervínculo: https://bit.ly/352VozF, en donde se asigna el puntaje a cada estudio primario de acuerdo a los criterios de calidad contemplados.

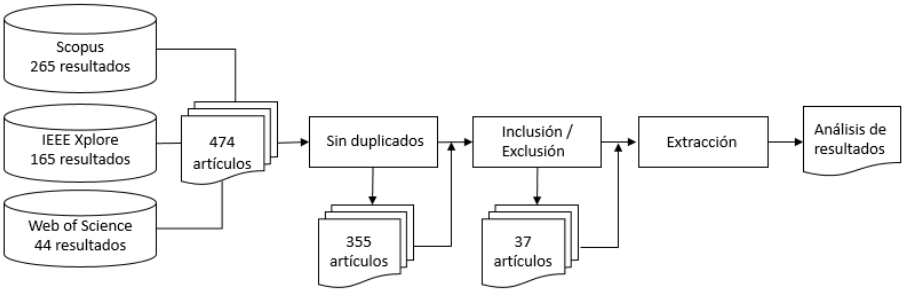

Fig. 1: Proceso para obtener los artículos del mapeo.

4.3 Amenazas a la validez

A continuación se presentan las amenazas a la validez de este mapeo de literatura, así como las estrategias para minimizarlas.

- **Construcción de la cadena de búsqueda**: la cadena de búsqueda retornó 313 estudios primarios que terminaron siendo descartados del mapeo. Aunque muchos de estos estudios abordaban temas de aprendizaje automático, sus contextos eran muy distintos al estudiado, estando la mayoría dentro del contexto de *software* para el cuidado de la salud. La estrategia seguida para mitigar esta amenaza fue hacer la lectura de todos los resúmenes de estos estudios, para así discernir si el contexto en el que se enfocaban era el de interés para este mapeo.
  El término "*artificial intelligence*" en la cadena puede crear un sesgo en los resultados hacia una tecnología en particular. Esta amenaza se mitigó al realizar varias búsquedas piloto con distintas cadenas, con esto se logró obtener un conjunto más homogéneo de tecnologías analizadas.
- **Selección de los repositorios utilizados**: elegir los repositorios de búsqueda sugiere una amenaza en la que los artículos que almacenan resulten irrelevantes en el mapeo, o que directamente no existan estudios en ellos que aporten en las respuestas de las RQs. Esta amenaza se reduce al elegir tres de los más importantes y renombrados repositorios científicos. Además, muchos de los estudios primarios seleccionados estaban en dos o tres de los repositorios, lo que sugiere su completitud de las bases de datos en cuanto al tema investigado.
- **Generalización de resultados**: si el mapeo de literatura realizado se basa en estudios que dependen de variables complejas de replicar, podría surgir la amenaza de no ser fácilmente verificable. La estrategia de mitigación radica en la elección misma de los repositorios. Al ser repositorios reconocidos y respetados se genera una alta confianza en los artículos que almacenan.
- **Inclusión y exclusión**: fueron creados criterios de Inclusión y Exclusión para apoyar en la selección de estudios primarios. Una elección inadecuada de estos criterios podría haber excluido del mapeo de literatura artículos primarios que habrían aportado en los resultados obtenidos. Se mitigó esta amenaza siguiendo un protocolo de constantes revisiones por criterios expertos de profesionales con experiencia en investigación. En particular para

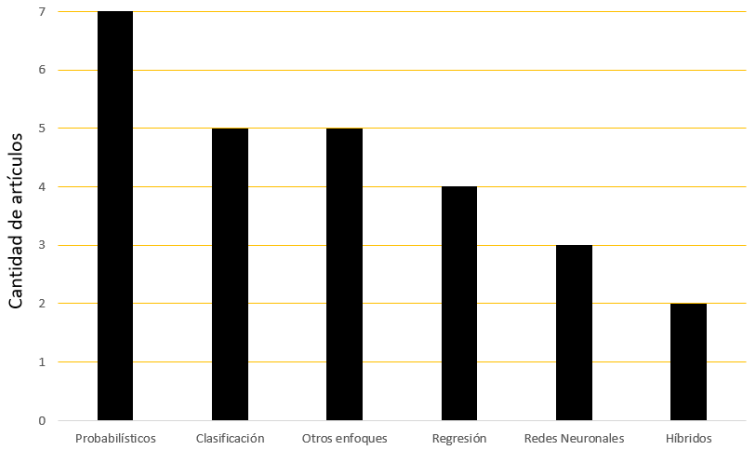

Fig. 2: Cantidad de estudios por categoría de aprendizaje automático.

el I1, se decidió ser inclusivos y también considerar estudios enfocados en rendimiento no solo en capas lógicas de las aplicaciones, sino también físicas, como la infraestructura donde estas se despliegan.

– **Repetibilidad**: crear un mapeo de literatura que sea muy similar a otros existentes lleva a que no haya aporte nuevo para el estado del arte del tema estudiado. El protocolo seguido, donde se depuró una cadena de búsqueda que se ejercitó en distintos repositorios científicos, es la forma de minimizar esta amenaza, puesto que los resultados de estas búsquedas retornaron estudios secundarios que se documentan en el trabajo relacionado, pero que no son idénticos al propuesto en este mapeo.

Tabla 1: Técnicas de aprendizaje automático más citadas.

Técnica	**Estudios**	**Categoría**	**Total**
Máquina de soporte vectorial	S03, S05, S11, S13, S23, S28, S36	Clasificación	7
Red neuronal	S01, S06, S10, S12, S13, S26	Red neuronal	6
Árboles de decisión	S07, S16, S21	Clasificación	3
Bosques aleatorios	S32, S34	Clasificación	2

5 Resultados

En esta sección se presentan los resultados del mapeo de literatura, basados en los 37 estudios primarios seleccionados. El detalle de estos estudios primarios se encuentra siguiendo este hipervínculo: https://bit.ly/352VozF, donde se extraen las características más importantes de cada uno respecto a lograr el objetivo de la presente investigación.

5.1 Técnicas de aprendizaje automático para mejorar el rendimiento de aplicaciones web (RQ1)

Se reportaron 26 técnicas de aprendizaje automático para mejorar el rendimiento de aplicaciones web. La Figura 2 agrupa los estudios por categoría de aprendizaje

automático, según la taxonomía de Shalev-Shwartz y Ben-David [3]. Por su parte, la Tabla 1 presenta las técnicas más reportadas de aprendizaje automático, junto con su respectiva categoría de clasificación. La técnica más utilizada fue Máquina de soporte vectorial, con siete estudios, seguida de Redes neuronales, con seis estudios. Es importante notar que un total de cinco estudios utilizaron técnicas que no pudieron clasificarse dentro de la taxonomía, por lo que se agruparon en la columna "Otros enfoques" de la Figura 2. Estas técnicas fueron: *Target-driven parallelism* [S27], Doble matriz [S04], *Q-learning* [S35], *Fuzzy logic* [S21], y *Fuzzy reinforcement learning* [S17].

Las técnicas reportadas solo una vez fueron: Doble matriz [S04], *Gradient boosting* [S32], *Load estimation with pre-learning* [S02], *Particle swarm optimization* [S06], *Linear regression* [S19], Teoría Dempster-Shafer [S28], *Q-learning* [S35], *Ant colony optimization* [S09], *Honey bee algoritm* [S09], *Fuzzy reinforment learning* [S17], *Gradient descent* [S27], BayesNet [S36], NaiveBaes [S36], *Regression trees* [S34], *Regression splines* [S34], *TreeNet* [S34], *ID3* [S36], *Bayesian network* [S21], *Fuzzy logic* [S21], *Target driven parallelism* [S27], y *Multinomial logistic regression* [S01].

Fue común encontrar técnicas utilizadas para predecir las acciones que tomará un usuario mientras interactúa con una aplicación web. Al predecir las acciones antes de que estas se realicen, una aplicación web logra tomar ventaja de tiempo y preparar la respuesta a esta acción antes que el usuario la ejecute.

Se encontraron mejoras de rendimiento al utilizar SVM en problemas de predicción, por ejemplo: en peticiones almacenadas en caché tras la correcta predicción de la siguiente página que un usuario visitará durante su interacción con una aplicación web [S03]. Esto fue posible de modelar en una SVM en base al registro almacenado en archivos históricos para el caso particular de la NASA, *ClarkNet* y el departamento de Computación e Ingeniería de la Universidad de Washington.

También se emplearon SVM para el uso eficiente de los núcleos de un procesador en dispositivos móviles para renderizar sitios web. Esto se logró mediante entrenamiento *off-line* para dar con modelos predictivos a utilizarse en el proceso de renderización. Entonces, se seleccionaba el procesador adecuado de acuerdo a parámetros como el *network* en donde se estableció la conexión con la aplicación y los componentes web que se necesitaban renderizar [S13].

Las redes neuronales también se utilizaron en el proceso de renderizado de aplicaciones web [S13]. La forma en la que se aprovechó esta técnica fue, en primer lugar, predecir el tiempo de carga que tomaría a un sitio web desplegarse completamente, y por otro lado, estimar el consumo de energía que le tomaría al dispositivo este proceso de despliegue. Con esta información se toman decisiones respecto al procesador del dispositivo del cliente que debe renderizar la aplicación.

El algoritmo de doble matriz es utilizado, junto con la técnica de *pre-fetching*, para aprovechar la memoria caché de las aplicaciones web. Su objetivo es predecir la siguiente página que un usuario necesitará y pre-cargarla en caché [S04]. La doble matriz es entrenada con patrones de peticiones realizadas por los usuarios

Tabla 2: Indicadores de rendimiento para las aplicaciones web.

Indicador	Estudios	Capa	Total
Uso de caché	S03, S04, S06, S07, S08, S17, S18	*front-end*, *back-end*	7
Tiempo del primer *byte*	S07, S08, S11, S19, S20, S21	*back-end*	6
Tiempo total de carga	S01, S06, S05, S22, S23, S25	*front-end*, *back-end*	6
Latencia	S15, S18, S25, S26, S27	infraestructura	5
Carga de trabajo de la red	S05, S22, S28, S29	infraestructura, *back-end*	4
Tamaño del DOM	S01, S13	*front-end*	2
Tiempo de respuesta del servicio	S19, S30	*back-end*	2
Consumo de energía	S05, S23	infraestructura	2
Uso del CPU	S11, S31	*back-end*	2
Utilización de memoria	S11, S31	*back-end*	2
Índice de velocidad	S24, S32	*front-end*	2
Carga del servidor y ancho de banda	S26, S31	infraestructura, *back-end*	2

para que logre producir como salida la siguiente página que se visitará. Esta técnica es denominada doble matriz al contar con una matriz dedicada a almacenar la probabilidad de una página para ser elegida como siguiente página a pre-cargar, además de una segunda matriz con información sobre pesos máximos para reducir el tiempo que toma el algoritmo para ubicar la entrada más pesada por cada fila de la primer matriz.

El *target-driven parallelism* (TPC) se utilizó para corregir dinámicamente peticiones a un servidor que fueron predichas incorrectamente como peticiones de corta duración [S27]. Al corregirse dinámicamente, fue posible mover estas peticiones a una lógica de paralelismo que reduce su tiempo de completitud. Esta solución se basa en la existencia de un predictor en el servidor encargado de clasificar peticiones como de larga o corta duración. TPC corrige posibles fallas del predictor y cambia la forma en que se computan las peticiones para que tomen ventaja de la programación en paralelo.

5.2 Indicadores que permiten medir el rendimiento de las aplicaciones web (RQ2)

Se identificaron 24 indicadores para medir el rendimiento de las aplicaciones web. Este conjunto de indicadores se muestra en la Tabla 2. El indicador más citado en la literatura fue el Uso de caché, siendo citado en siete estudios, seguido por el Tiempo del primer *byte*, y el Tiempo total de carga, ambos reportados en seis estudios cada uno.

Los indicadores reportados en solo un estudio fueron: *Script size* [S06], *Average image number* [S07], *Average search response time* [S07], *Load utilization ratio* [S08], *Query to connectivity ratio* [S08], Utilización de recursos [S09], Número de conexiones por segundo [S09], Número de peticiones por segundo [S09], *Service level agreement* [S19], *Perceptual speed index* [S32], Elasticidad [S35], *Throughput* [S21], *Router roundtrip time* [S37], Tamaño de la sesión [S16].

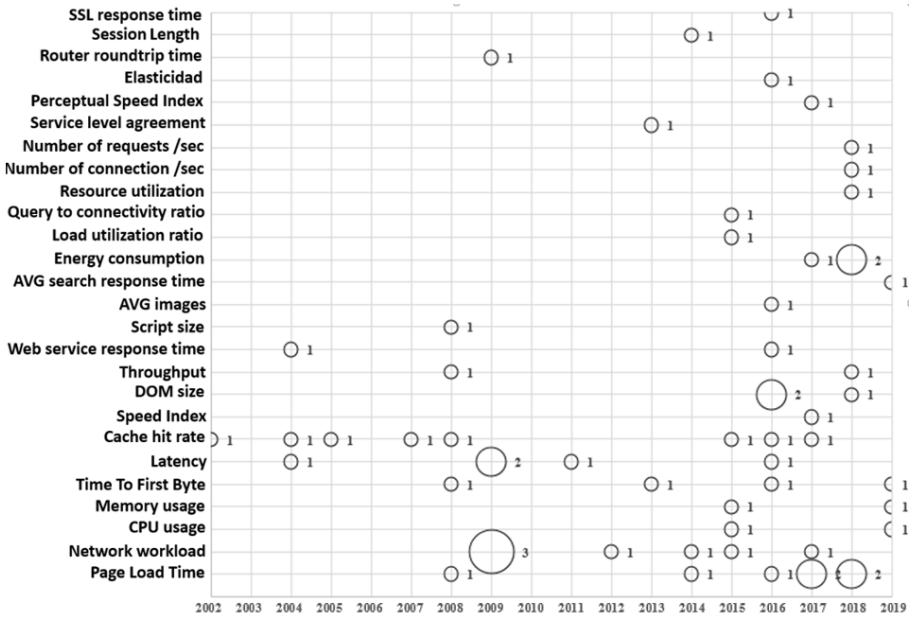

Fig. 3: Uso de diversos indicadores de rendimiento en el tiempo.

La Figura 3 muestra el uso de indicadores de rendimiento a través de los años, desde el 2002 hasta el 2019. Se puede notar que el Uso de caché ha sido reportado de forma relativamente constante a través del tiempo (con excepción del periodo 2009-2014), mientras que el Tiempo total de carga ha tenido un repunte a partir del año 2014.

Para los dos primeros lustros, que comprenden del año 2000 al 2010, el uso del caché para obtener recursos no modificados de una manera mucho más rápida fue el indicador más citado, con cinco reportes por lustro. Luego, para el lustro que comprende del año 2011 al 2015, hubo una tendencia hacia el indicador *Network workload*, con tres referencias en la literatura. Este indicador tiene que ver con la carga de trabajo de un servidor en los momentos específicos donde se toma una medición y por un periodo de tiempo dado [S22]. Finalmente, desde el año 2016 y hasta la actualidad, se reportan cinco estudios que se enfocan en mejorar el *Page load time* de los sitios en Internet. Este indicador se refiere al momento en el que transcurren dos segundos y no hay más actividad de red para cargar recursos de la aplicación [11].

5.3 Capas de aplicación web donde se ha mejorado el rendimiento con el uso de aprendizaje automático (RQ3)

Los esfuerzos para mejorar el rendimiento de aplicaciones web pueden ser clasificados según la capa de aplicación donde se utilicen: *front-end*, *back-end*, e infraestrutura de despliegue. Siguiendo esta clasificación, la Tabla 3 muestra las capas de aplicación donde se han implementado técnicas de aprendizaje automático, así como las áreas (por capa) donde se han realizado las mejoras de

Tabla 3: Capas y áreas de aplicación donde se usa aprendizaje automático.

Capa	**Área**	**Estudios**	**Total**
Back-end	Caché del servidor	S03, S08, S10, S17, S18, S26, S34, S36	8
	Análisis de archivos históricos	S03, S07, S16, S28	4
	Uso del CPU	S13, S27	2
Front-end	*Pre-fetching*	S04	1
	Renderizado del DOM	S05	1
Infraestructura	Balanceadores de carga	S09, S12, S14, S17, S27, S30, S36, S37	8
	Infraestructura como servicio	S11, S35	2
	Secure sockets layer	S02	1
	Procesadores de dispositivos móviles	S05	1
	Precision time protocol v2	S22	1

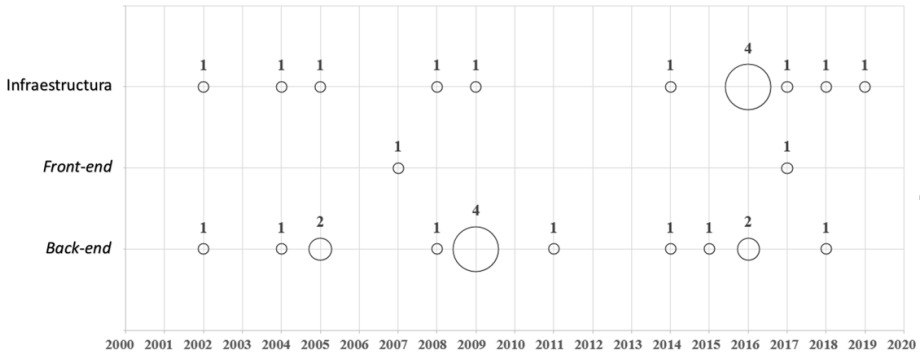

Fig. 4: Uso de aprendizaje automático por capa de aplicación, a través del tiempo.

rendimiento. A continuación se señalan las mayores incidencias por cada capa de aplicación:

- ***Front-end***: esta capa reportó mejoras al aprovechar el *pre-fetching* en los navegadores web para obtener recursos necesarios por una aplicación web antes que el usuario los requiera. Por otro lado, también se estudió un proceso de renderizado del DOM más eficiente, tomando como factor la capacidad de procesamiento en el dispositivo donde se despliega la aplicación web.
- ***Back-end***: la capa de *Back-end* tuvo un gran énfasis en el almacenamiento en caché de recursos y peticiones que los usuarios necesitan más frecuentemente. Además, reportó un gran esfuerzo en almacenar constantemente información detallada de archivos históricos sobre las incidencias de lo sucedido en los procesamientos en el servidor. Estos archivos son las principales entradas que luego se utilizan para los procesos de entrenamiento en las técnicas de aprendizaje automático. El desempeño del CPU mejoró con el uso de aprendizaje automático en características como la cantidad de ciclos de reloj ejecutadas por unidad de tiempo, los cambios de contexto necesarios para realizar una tarea y la velocidad de ejecución de procesos.

– **Infraestructura**: la capa de infraestructura reportó grandes esfuerzos por mejorar el desempeño de los balanceadores de carga. Ubicar el servidor más libre para atender una petición de usuario, un manejo de caché compartido entre servidores, y la homogeneización de peticiones de usuario para facilitar su almacenamiento en caché a través de *proxys*, son algunos de estos esfuerzos. Además, otros enfoques como el despliegue en la nube de aplicaciones web reportaron mejoras en factores como la elasticidad para encender y apagar nodos que resuelvan peticiones de usuario según sea el tráfico que recibe la aplicación en un momento dado.

Finalmente, la Figura 4 presenta la tendencia en el tiempo (del año 2000 al 2019) de los estudios realizados para mejorar el rendimiento por capa de aplicación De esta figura se observa que tanto la capa de *back-end* como la de infraestructura han tenido una cantidad constante de estudios a través del tiempo.

6 Conclusiones

Este artículo presentó un mapeo sistemático de literatura sobre el uso de aprendizaje automático para mejorar el rendimiento de las aplicaciones web. Se analizaron 37 estudios primarios, obteniendo como principales hallazgos: 26 técnicas utilizadas para la mejora del rendimiento, 24 indicadores para evaluar el rendimiento de las aplicaciones web, y 3 capas de aplicación y 10 áreas donde se aplicaron las mejoras.

Los indicadores mapeados en este estudio evalúan la actuación de una aplicación web tanto en sus tiempos del *Back-end* como del *Front-end*. El total de 24 indicadores de rendimiento ubicados abordan temas muy heterogéneos entre sí, estos van desde una efectiva utilización de caché, hasta una infraestructura óptima de servidores para recibir y responder eficientemente las peticiones que recibe una aplicación web.

Se ubicaron dos grandes conjuntos de técnicas de aprendizaje automático que se clasifican de acuerdo al tipo de problema que resuelven. Por ejemplo, problemas de clasificación son abordados primordialmente por algoritmos como las Máquinas de soporte vectorial, mientras que para problemas de decisión se prefieren algoritmos como los Arboles de decisión.

En total se ubicaron 26 algoritmos de aprendizaje atomatizado empleados para mejorar el desempeño de aplicaciones en Internet. Estos algoritmos son empleados en distintas de las capas que componen estas aplicaciones. La forma en la que se midió la mejora al emplear estos algoritmos fue mediante el uso de métricas de rendimiento, también reportadas en este estudio. La capa de infraestructura, junto con la capa del *back-end*, fueron las que recibieron más esfuerzos para la mejora de rendimiento. Es por esto que en los equipos de desarrollo de *software* siempre se debe tomar en cuenta tanto la velocidad de los procesos ejecutados, como la infraestructura donde corren estos procesos, para así obtener el máximo desempeño de una aplicación web a partir de acciones que la potencien desde el momento que esta es liberada al ambiente de producción.

Las principal limitación enfrentada fue clasificar las métricas de evaluación reportadas en los estudios primarios. La forma en la que algunos estudios reportaron las métricas no tuvo el detalle necesario para reconocerla como una métrica que aún no se reportaba, o si era la reiteración de una métrica ya contabilizada en el estudio. Por ello es que en varias oportunidades hubo un análisis más contextual alrededor de la métrica para saber cómo documentarla adecuadamente, lo cuál hizo que se demorara un poco más de tiempo para esta labor.

Como trabajo a futuro se plantea una pregunta de investigación que estudie herramientas para evaluar desempeño de las aplicaciones web. Durante este mapeo se vio la utilidad de herramientas como *Web Page Test*, *Lighthouse* o *SpeedCurve*, para el proceso de toma de métricas en una aplicación web. Sería de mucho provecho ahondar en las herramientas existentes, su respectiva comparación a nivel general, así como un análisis a detalle de cada una de ellas.

Finalmente, es grato identificar esfuerzos que se están llevando a cabo para mejorar el rendimiento en aplicaciones web, los cuales se traducen en mejores productos lanzados a Internet y mayor acogida de estos por parte de sus visitantes.

Referencias

1. A. Bleicher. (2011) A memory of webs past. [Online]. Available: https://spectrum.ieee.org/tech-history/cyberspace/a-memory-of-webs-past
2. J. Chadwick, T. Snyder, and H. Panda, *Programming ASP.NET MVC 4: Developing Real-World Web Applications with ASP.NET MVC*, 1st ed. O'Reilly Media, Inc., 2012.
3. S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, 2014.
4. Z. Kazhmaganbetova, S. Imangaliyev, and A. Sharipbay, "Machine learning for the communication optimization in distributed systems," *International Journal of Engineering and Technology(UAE)*, vol. 7, pp. 47–50, 09 2018.
5. S. Pradeep and Y. K. Sharma, "A pragmatic evaluation of stress and performance testing technologies for web based applications," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, Feb 2019, pp. 399–403.
6. M. Arslan, U. Qamar, S. Hassan, and S. Ayub, "Automatic performance analysis of cloud based load testing of web-application amp; its comparison with traditional load testing," in *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Sep. 2015, pp. 140–144.
7. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Planning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 89–116. [Online]. Available: https://doi.org/10.1007/978-3-642-29044-2_8
8. K. BA and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," vol. 2, 01 2007.
9. J. J. Marciniak, Ed., *Encyclopedia of Software Engineering*. New York, NY, USA: Wiley-Interscience, 1994.
10. M. Pai, M. McCulloch, J. D. Gorman, N. P. Pai, W. T. A. Enanoria, G. E. Kennedy, P. Tharyan, and J. M. Colford, "Systematic reviews and meta-analyses: an illustrated, step-by-step guide." *The National medical journal of India*, vol. 17 2, pp. 86–95, 2004.
11. R. Viscomi, A. Davies, and M. Duran, *Using WebPageTest: Web Performance Testing for Novices and Power Users*, 1st ed. O'Reilly Media, Inc., 2015.