FACULTAD DE INGENIERÍA, ARQUITECTURA Y URBANISMO

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

Análisis comparativo de algoritmos de aprendizaje automático para detectar ataques de SQLI en microservicios web

PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO DE SISTEMAS

Autor(es):
Peralta Garcia Edwin Jahir
https://orcid.org/0000-0002-0182-4840
Quevedo Monsalbe Juan Carlos
https://orcid.org/0009-0003-2343-7539

Asesor:
Mg. Juan Carlos Arcila Diaz
https://orcid.org/0000-0002-7788-951X

Línea de investigación:
Ciencias de la información como herramientas multidisciplinares y estratégicas en el contexto industrial y de organizaciones

Sublínea de Investigación:
Informática y transformación digital en el contexto industrial y Organizacional

Pimentel – Perú
2024

Análisis comparativo de algoritmos de aprendizaje automático para detectar ataques de SQLI en microservicios web

Aprobación del jurado

DR. ATALAYA URRUTIA CARLOS WILLIAM

Presidente del Jurado de Tesis

MG. ASENJO CARRANZA ENRIQUE DAVID

Secretario del Jurado de Tesis

MG. GUEVARA ALBURQUEQUE LAURITA BELEN

Vocal del Jurado de Tesis

DECLARACIÓN JURADA DE ORIGINALIDAD

Quienes suscriben la DECLARACIÓN JURADA, somos **Edwin Jahir Peralta Garcia** y **Juan Carlos Quevedo Monsalbe,** del Programa de Estudios de **Ingeniería de Sistemas** de la Universidad Señor de Sipán, declaramos bajo juramento que somos autores del trabajo titulado:

Análisis comparativo de algoritmos de aprendizaje automático para detectar ataques de SQLI en microservicios web

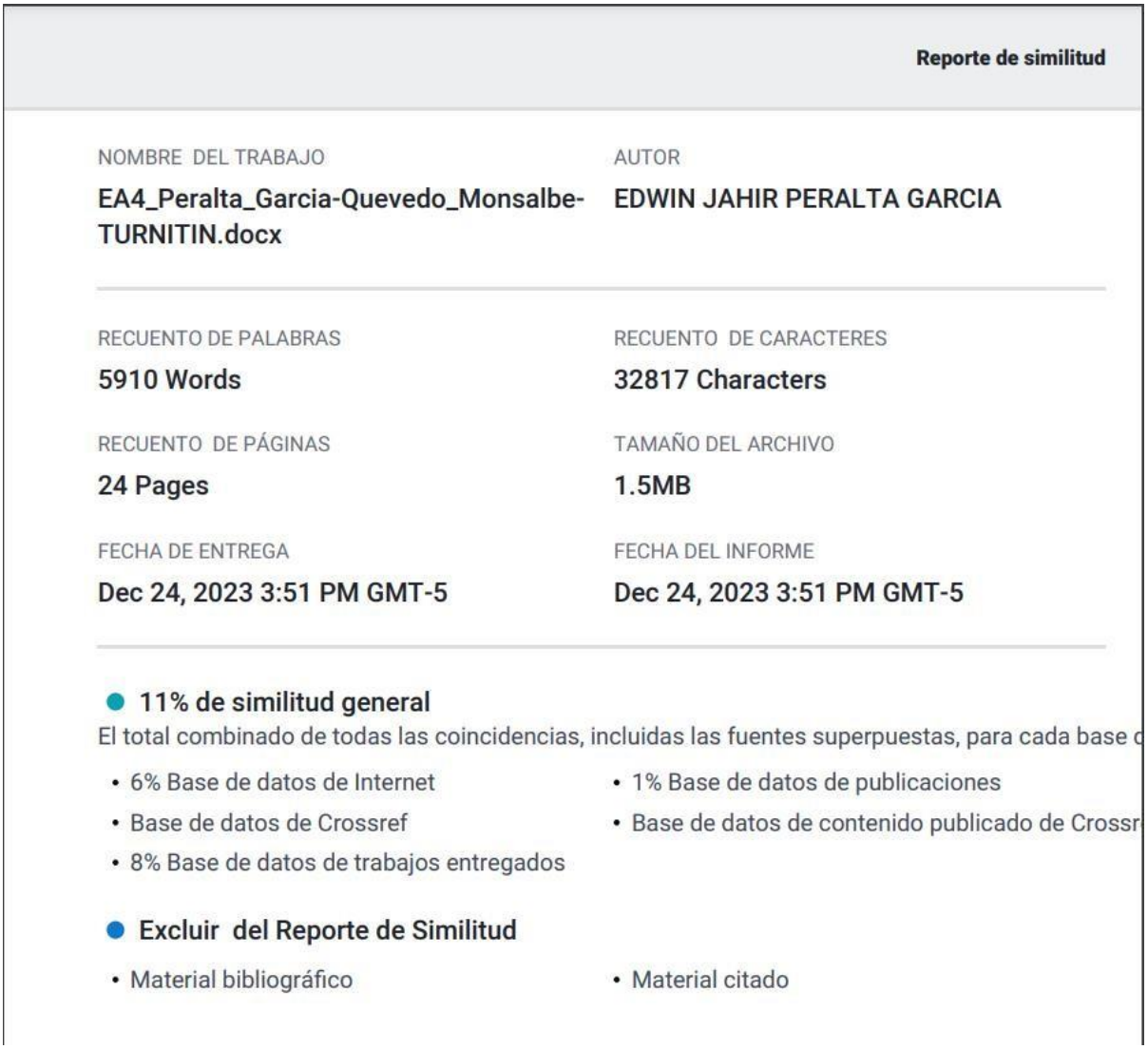
El texto de mi trabajo de investigación responde y respeta lo indicado en el Código de Ética de la Universidad Señor de Sipán, conforme a los principios y lineamientos detallados en dicho documento, en relación con las citas y referencias bibliográficas, respetando el derecho de propiedad intelectual, por lo cual informo que la investigación cumple con ser inédito, originaly autentico.

En virtud de lo antes mencionado, firman:

Peralta Garcia Edwin Jahir	72517313	
Quevedo Monsalbe Juan Carlos	75724740	

Pimentel, 15 de diciembre de 2023.

REPORTE DE SIMILUTD TURNITIN



Reporte de similitud

NOMBRE DEL TRABAJO
EA4_Peralta_Garcia-Quevedo_Monsalbe-TURNITIN.docx

AUTOR
EDWIN JAHIR PERALTA GARCIA

RECUENTO DE PALABRAS
5910 Words

RECUENTO DE CARACTERES
32817 Characters

RECUENTO DE PÁGINAS
24 Pages

TAMAÑO DEL ARCHIVO
1.5MB

FECHA DE ENTREGA
Dec 24, 2023 3:51 PM GMT-5

FECHA DEL INFORME
Dec 24, 2023 3:51 PM GMT-5

11% de similitud general
El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para cada base c

- 6% Base de datos de Internet
- 1% Base de datos de publicaciones
- Base de datos de Crossref
- Base de datos de contenido publicado de Crossr
- 8% Base de datos de trabajos entregados

Excluir del Reporte de Similitud

- Material bibliográfico
- Material citado

ÍNDICE

Resumen

Las inyecciones SQL (Structured Query Language) plantean una amenaza constante a los microservicios web, destacando la urgencia de una identificación eficiente para abordar esta vulnerabilidad. El presente estudio compara algoritmos de aprendizaje automático para la detección de inyecciones SQL en microservicios web, utilizando un dataset público de 22,749 datos. Se llevó a cabo una revisión literaria para identificar tipos de inyecciones SQL y algoritmos de aprendizaje automático. Se compararon los resultados de Random forest, Decision tree y Support Vector Machine. Los hallazgos indican que Random forest lidera con una precision y acurracy del 99%, recall del 97%, y F1-Score del 98%. En cambio, Decision tree obtiene una precision del 92%, recall del 86%, y F1-Score del 97%. SVM presenta una acurracy, precision y F1-Score del 98%, con un recall del 97%. En resumen, Random forest se destaca como óptimo para la detección de inyecciones SQL en microservicios web.

Palabras Clave: SQL injection, Aprendizaje automático, Algoritmos, Aplicaciones web, técnicas.

Abstract

SQL (Structured Query Language) injections pose a constant threat to web microservices, highlighting the urgency of efficient identification to address this vulnerability. This study compares machine learning algorithms for detecting SQL injections in web microservices using a public dataset of 22,749 data. A literature review was conducted to identify types of SQL injections and machine learning algorithms. The results of Random forest, Decision tree and Support Vector Machine were compared. The findings indicate that Random forest leads with an accuracy and acurracy of 99%, recall of 97%, and F1-Score of 98%. In contrast, Decision tree has an accuracy of 92%, recall of 86%, and F1-Score of 97%. SVM has an accuracy, precision and F1-Score of 98%, with a recall of 97%. In summary, Random forest stands out as optimal for SQL injection detection in web microservices.

Keywords: SQL injection, Machine learning, Algorithms, Web applications, techniques.

I. INTRODUCCIÓN

Los microservicios, un estilo de desarrollo de software que ha ganado importancia en los últimos años, se basa en descomponer una aplicación en servicios independientes, cada uno encargado de una función específica, brindando escalabilidad, flexibilidad y facilidad de implementación [1] . Sin embargo, presenta desafíos en cuanto a la gestión de datos, particularmente en las bases de datos, estas contienen información importante y confidencial de una organización, enfrentándose a amenazas como las inyecciones SQL [2]. Este ataque que se presenta en diferentes organizaciones, es causado por usuarios que tienen la intención de obtener información sensible relacionada con la base de datos, manipulando los datos, generando efectos perjudiciales que afectan a diversas organizaciones a nivel mundial [3]. La prevención de este tipo de ataques se ha convertido en prioridad en las organizaciones y también para las industrias de desarrollo de software, destacando la detección temprana y precisa para evitar efectos perjudiciales [3][4]. Históricamente, la prevención de este tipo de ataques se basaba en la validación de la entrada de datos, examinando que no contengan caracteres especiales asociados a ataques comunes. Aunque esta práctica aún persiste en la actualidad, ha dejado de ser eficaz para contrarrestar otros tipos de ataques [6]. Con el constante avance tecnológico, tanto organizaciones como desarrolladores buscan activamente nuevas soluciones para hacer frente a este problema, que continúa emergiendode manera constante.

En este contexto, se tiene como objetivo principal analizar comparativamente algoritmos de aprendizaje automático para la detección de inyecciones SQL en microservicios web, haciendo uso de datasets públicos de inyecciones SQL, con el propósito de evaluar la eficacia de los algoritmos de aprendizaje automático en la detección de este tipo de vulnerabilidades en microservicios web, a partir de ello surgió la necesidad de plantearse la siguiente pregunta de investigación ¿Qué algoritmo de aprendizaje automático presenta mejor desempeño para detectar inyecciones SQL en microservicios web?; por lo cual, se ha obtenido la siguiente hipótesis, el algoritmo Random forest presenta mejor desempeño para

la detección de inyecciones SQL.

Las herramientas tecnológicas basadas en aprendizaje automático para detectar ataques de inyección SQL están teniendo un impacto en las industrias del desarrollo y en las organizaciones. Diferentes métodos han sido desarrollados en base a machine learning para detectar SQLI. En el trabajo [7], se realizaron pruebas utilizando herramientas de auditoría de seguridad enfocadas en la detección de inyecciones SQL con el propósito de identificar patrones comunes y clasificar vectores de ataque. A partir de los datos recopilados, se desarrollaron dos sistemas clasificadores: uno basado en Naive Bayes (NB) y otro en Decision Tree (DT). Los resultados indicaron que el primero alcanza un rendimiento del 97% con una precision del 98%. En [8], se basaron en un modelo de vectorización de textos mediante un algoritmo llamado ITFIDF el cual se mejoró y utilizó junto a Support Vector Machine (SVM), K-Nearest Neighbors (KNN) y DT, buscando así la detección de inyecciones SQL. Los resultados del primer algoritmo revelaron una precision del 99.08%, un recall del 99.34%, y un F-score del 99.21%. KNN logró una precisión del 98.22%, un recall del 98.12%, y un F-score del 98.17%, mientras que Decision Tree alcanzó una precisión del 98.18%, un recall del 97.98%, y un F-score del 98.08%. En conclusión, se señala que TFIDF + SVM es la combinación que arroja los mejores resultados. En [9], se recopilaron datos auténticos de SQLI a nivel empresarial y se juntaron con la misma cantidad de consultas normales, las cuales fueron utilizadas para entrenar a una red neuronal basada en Multi-Layer Perceptron (MLP) y Long Short Term Memory (LSTM), los resultados indicaron que el modelo basado en MLP con una capa oculta es el más óptimo, logrando alcanzar una precision de 96.24%, un recall de 100%, una acurracy de 99.67%, y una tasa de falsos positivos de 0.36%, todo ello en un tiempo de procesamiento de 84.1 milisegundos. En el estudio [10], se construyó un conjunto de datos extrayendo muestras etiquetadas de la web, y se amplió mediante la incorporación de sql-injection-payloadlist. Utilizando este conjunto de datos, se entrenó un modelo de red neuronal MLP secuencial de Keras, diseñado para explotar la cláusula WHERE en las bases de datos; este enfoque condujo a resultados notables, con una precision del

85%, un Recall del 100%, un F1-Score del 92%, y una acurracy del 94.4%. Por otro lado, Gandhi, et al. Realizó una comparación entre algoritmos de machine learning, tales como (QDA, MLP, CNN, Decisión Tree y otros) y propuso un enfoque de algoritmo hibrido el cual está basado en CNN-BiLSTM, con la finalidad de conocer su efectividad para detectar SQLI logrando obtener una acurracy de 98% un recall de 100%, un F1-score de 98.5% y una precision de 98.3% demostrando así lo óptimo de este algoritmo frente al resto de modelos [11]. En el trabajo [12], se recopilaron consultas maliciosas de SQL a través de enfoques como Bag-Of-Words (BOW) y word2vec. Estas consultas se utilizaron como conjunto de entrenamiento para los modelos de Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN) y Long Short-Term Memory (LSTM), al emplear BOW, se alcanzaron niveles de precisión del 91.0%, 95.4% y 91.9%, mientas que con word2vec, los porcentajes fueron del 76.3% 76.7% y 81.6% respectivamente. En [13], se creó un sistema denominado KubAnomaly, el cual emplea técnicas de redes neuronales y aprendizaje supervisado para detectar múltiples anomalías en los sitios web, respecto a las inyecciones SQL, los resultados revelaron una destacada precision 99.6% un recall de 99.6% y un F1-score de 99.6%.

El enfoque de aprendizaje automático fue elegido en la presente investigación por la capacidad demostrada para abordar problemas complejos de ataques en base de datos. A través de la implementación de diferentes técnicas de machine learning, se entrena un modelo capaz de poder detectar con precisión las inyecciones SQL, con el objetivo de proporcionar un algoritmo confiable y efectivo para la industria de desarrollo de software.

II. MATERIALES Y MÉTODOS

Tabla 1. Materiales de investigación

N	NOMBRE	CARACTERISTICAS
1	Conjunto de Datos (Dataset que contiene consultas SQLI y no SQLI)	Conjunto de datos que contiene consultas benignas y maliciosas de SQL, para diversas bases de datos, representándose por etiquetas (Query, Label)

2	Herramientas de Software	Entorno de desarrollo de Python (COLAB), bibliotecas de aprendizaje automático (Skelearn y Pandas)
3	Hardware	- Laptop i5 Acer, Tarjeta gráfica RTX 3050, 16gb de RAM, 512gb SSD - Laptop i7 Chrome OS, 16gb de RAM, 512gb de SSD

Fuente: Elaboración propia

Método de la investigación

Figura 1: Método de la investigación

En la figura 1, se presenta el método utilizado en la investigación. En la primera etapa se realizó la caracterización de inyecciones SQL para el conocimiento de los tipos de inyecciones existentes, la segunda etapa se seleccionó el Dataset que contiene inyecciones anómalas y consultas normales para el entrenamiento y prueba de los algoritmos de aprendizaje automático, la tercera etapa se seleccionó los algoritmos de aprendizaje automático con mejor desempeño en la detección de inyecciones SQL, en base a una revisión literaria, la cuarta etapa se realizó el desarrollo de una aplicación web bajo la arquitectura de

microservicios para las pruebas, la quinta etapa se realizó el entrenamiento de los algoritmosde aprendizaje automático seleccionados, para posteriormente implementarlos en la aplicación desarrollada y en la última etapa se evaluaron los resultados en base a indicadores.

Inicialmente, se llevó a cabo la caracterización de los ataques de inyección SQL a bases de datos con el propósito de comprender sus tipos y métodos de ejecución. Se identificaron ocho tipos de inyecciones SQL que son ampliamente utilizadas por los atacantes (ver anexo 7). Posteriormente, se procedió a analizar las características específicas de cada tipo de ataque para obtener un mayor rendimiento de su estructura y aplicación. La Tabla 1 presenta los ocho tipos de inyección, junto con sus definiciones, consultas normales y anómalas, parámetros inyectables y descripciones asociadas.

Tabla 2: Características de tipos de SQLI

Tipo de ataque	Definición	Consulta normal	Parámetro inyectable	Consulta anómala	Descripción
Tautology SQLI	Este tipo de ataque intenta utilizar un argumento de pregunta condicional para probar la validez de la consulta SQL, esto se realiza utilizando la cláusula WHERE donde el atacante inyecta la condición y la transforma en una tautología que siempre es válida [14][15].	SELECT * FROM usuarios WHERE nombre = 'usuario' AND password = 'contraseña';	OR '1'='1';	SELECT * FROM usuarios WHERE nombre = 'usuario' ANDpassword = 'contraseña'**OR '1'='1';**	Se convierte en una consulta anómala al agregar la sentencia 'OR '1'='1';' a la consulta SQL. Esto provoca que los campos de la tabla se evalúen siempre como verdaderos, resultando en la devolución de datos de la tabla, independientemente de las condiciones reales [15][16].
Illegal/Logically incorrect query SQLI	La inyección de este tipo, ocurre por errores, falta de validaciones o lasentradas son inválidas desde el punto de vista lógico, puesto que en periodo de desarrollo ayuda a los programadores a corregir sus programas [17].	SELECT * FROM products WHERE product_name = 'name';	; SELECT * FROM users; --';	SELECT * FROM products WHERE product_name = **'; SELECT * FROM users; --';**	Esta inyección intenta ejecutar dos consultas SQL al mismo tiempo, la primera se cierra prematuramente y la segunda selecciona todos los registros de la tabla users, potencialmente revelando informaciónsensible.
Union query SQLI	La inyección de este tipo se compone principalmente por la palabra UNION la cual simula una sola consulta compuesta por la cláusula SELECT. Para que se aplique esta consulta	SELECT * FROM accountTable WHERE user_login ='umar';	UNION	SELECT * FROM accountTable WHERE user_login = 'umar' **UNION** SELECT * FROM accountTable WHERE No=10232	La consulta puede referenciar una inyección SQL de un login, ya que trata de unir dos tablas mediante una sola consulta, buscando así devolver de todos los usuarios que tengan el

	es necesario que las tablas se encuentren relacionadas y se debería conocer el nombre correcto de las tablas junto a sus campos [14][18].				nombre de usuario umar o el valor 10232 en la columna No [14].
Piggy-Backed Query SQLI	La inyección de este tipo ejecuta dos consultas al mismo tiempo, la primera debería ser correcta yla segunda controladapara la extracción dedatos, adición, modificación, ejecución remota odenegación deservicio [19][17].	SELECT book_name FROM booktable WHERE book_id='book1' AND book_name = 0	;	SELECT book_name FROM booktable WHERE book_id= 'book1' AND book_name = 0**; DROP booktable;**	Esta consulta se caracteriza por poneruna consulta verdaderay se ejecuta normalmente, adicionalmente seaplica las consultas maliciosas [18].
Blind SQLI	Se compone de la formación de serie de consultas verdaderas/falsas, recopilando así datos valiosos infiriendo de las respuestas de la web consultada [14][19].	SELECT 5 WHERE username = :username	1/0 ELSE SELECT 5	admin'; IF SYSTEM_USER='sa' SELECT **1/0 ELSE SELECT 5**	La consulta se compone de dos partes, antes del punto y coma y después de este, además se tiene que si el valor es correcto va a devolver un mensaje de error o un acceso directo sin embargo si no se llega a tener la respuesta se obtendrá '5' como respuesta.
Timing SQLI	Esta inyección maneja tiempos de respuesta en las que el atacante registra las respuestas que genera la base de datos, basándose en la técnica de datos transferencia de datos entrantes y salientes [17].	SELECT precio FROM productos WHERE id = 1;	or 1=1 --'; sleep (1)	SELECT precio FROM productos WHERE id = **' or 1=1 --'; sleep (1);**	En este tipo de inyección, el éxito del ataque se evalúa mediante la entrada de un input, donde la primera sentencia corresponde al ID. Si la inyección tiene éxito, se observará el tiempo de respuesta dentro del periodo programado en la consulta maliciosa.
Store Procedure SQLI	Tipo de vulnerabilidad de base de datos que permite a un atacante inyectar código malicioso en una consulta SQL almacenada en caché. Este tipo de ataque puede dar lugar a la ejecución de comandos arbitrarios en la base de datos, lo que puede llegar a comprometer toda la BD y sus registros [15][14].	SELECT * FROM accountTable WHERE user = 'umar' AND passwd = 'farooq';	SHUTDOWN;-- ;	SELECT * FROM accountTable WHERE user login= ‘umar’ AND passwd = ‘farooq’; **SHUTDOWN;--;**	La consulta tiene la última parte que se considera inyectable, siendo esta una manera forzada de detener o apagar el sistema de gestión de base de datos, además los últimos signos indican que todo lo posterior es considerado una consulta.
Alternate Encoding SQLI	Este tipo de inyección se aplica para eludir las validaciones de caracteres especiales, utilizando así codificaciones	SELECT * FROM accountTable WHERE user = 'umar' AND pin = 'farooq';	44524f502044415441 4241534520	SELECT * FROM accountTable WHERE user = 'umar' AND pin = 'farooq'; **EXEC('44524f50204441 54414241534520');**	Esta inyección busca ocultar ciertos caracteres o palabras reservadas de las bases de datos, con el fin de hacer que estas

alternativas como hexadecimal, ASCII o Unicode [19].		**SHUTDOWN;**	pasen las validaciones y se ejecuten, en este caso se tiene una encriptación en baseHEXADECIMAL

Fuente: Elaboración propia

Además, fue esencial identificar el conjunto de datos (Dataset) con el cual se implementarán los algoritmos de aprendizaje automático. En este proceso, se llevó a cabo una búsqueda en diversas fuentes como Kaggle y Github, identificando aproximadamente 30 conjuntos de datos. Posteriormente, se procedió a la selección del más adecuado para la investigación, considerando cuidadosamente los siguientes criterios:

- Volumen de datos, donde otras investigaciones [9][11][12] sugieren que, para el entrenamiento efectivo de los algoritmos de aprendizaje automático destinados a detectar inyecciones SQL, se requiere un conjunto de datos que oscile entre 20,000 y 40,000 registros.
- Fuente confiable, evidenciando que el dataset se encuentre en repositorios conocidos.
- Etiquetado preciso, se requiere una clasificación precisa mediante la asignación de valores que indiquen si una consulta es anómala o normal.
- Inclusión de datos normales, donde se debe incluir datos diferentes a lo que se desea detectar para un mejor entrenamiento.
- Legalidad, se debe seleccionar un conjunto de datos que sea legalmente permitido para su uso en investigaciones.

Por lo tanto, se optó por un conjunto de datos que satisface los criterios mencionados. Se seleccionó el conjunto de datos denominado “SQL Injection Detection by Machine learning” [20] para llevar a cabo este estudio, en la siguiente tabla se muestra el listado de los Dataset con los criterios a seleccionar.

Tabla 3: Selección de Dataset

Nombre	Fuente	Criterios

/Repositorio		Volumen de datos	Fuente confiable	Etiquetado preciso	Inclusión de datos normales	Legalidad
Machine learning – SQLI / (Github) [21]	https://github.com/Scott-Park/MachineLearning/blob/master/Sql-Injection/source/trainingdata/sql_querys.txt		X			X
Sql-injection-payload-list / (Github) [22]	https://github.com/payloadbox/sql-injection-payload-list		X		X	X
EP-CNN / (Github)	https://github.com/uestcer-xx/EP-CNN/blob/master/sqli_100.txt		X			X
Libinjection / (Github) [23]	https://github.com/client9/libinjection		X			X
Starter: SQL-Injection Payload e7fb1937-2 / Kaggle [24]	https://www.kaggle.com/code/kerneler/starter-sql-injection-payload-e7fb1937-2/input		X			X
sql injection Dataset / Kaggle [25]	https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset	X	X		X	X
SQL Injection Detection by Machine learning / Kaggle [20]	https://www.kaggle.com/code/sanshui123/sql-injection-detection-by-machine-learning/input?select=Modified_SQL_Dataset.csv	X	X	X	X	X

Fuente: Elaboración propia

Después de elegir el conjunto de datos, se decidió realizar un equilibrio mediante submuestreo. Este enfoque implica la eliminación de datos para lograr una proporción equitativa entre las consultas anómalas y normales. Inicialmente, el conjunto de datos seleccionado constaba de 30,905 registros, con 19,537 consultas anómalas y 11,382 normales. Para mejorar el proceso de entrenamiento de los algoritmos de aprendizaje automático, se llevó a cabo el submuestreo, resultando en un conjunto de datos final de 22,764 registros, con 11,382 datos tanto para consultas anómalas como normales. Este procedimiento garantiza resultados más robustos y equitativos durante el entrenamiento de los algoritmos.

De igual manera, para la selección de los algoritmos de aprendizaje automático, se llevó a cabo una revisión sistemática de la literatura teniendo como fuentes las bases de datos Scopus, ScienceDirect y IEEE Xplore. En este análisis, se extrajeron 320 artículos relacionados con el tema. Sin embargo, a través de un proceso de selección, aplicando criterios de inclusión y exclusión, se identificaron 32 artículos relevantes que proporcionaron

información sustancial sobre los algoritmos de aprendizaje automático utilizados para detectar inyecciones SQL (Ver anexo 8). Posteriormente, se hizo una selección de algoritmos que presenten mejor desempeño, tomando en cuenta los criterios de precision, acurracy, Recall y F1-Score, de esta manera se obtuvo como resultado que los algoritmos Decision tree, SVM y RandomForest presentan mejores resultados en cuanto a los criterios establecidos, y, además, fueron implementados por varios autores, como se detalla en la tabla 4.

Tabla 4: Top 10 de los algoritmos con mejor desempeño

Algoritmo	Accuracy	Precision	Recall	F1-Score
[18] Light Gradient Boosting Machine (LGBM)	0,9933	0,9933	0,9933	0,9933
[18] Gradient Boosting Machine (GBM)	0,9904	0,9898	0,9903	0,991
[14][19] Artificial Neural Network (ANN)	0,9893	0,9870	0,9913	0,99
[26][18] AdaBoost (AB)	0,9808	0,9559	0,9592	0,9561
[27][28][19][17] Decision Tree (DT)	0,9668	0,9315	0,88955	0,9164
[27][28][19] Random Forest (RF)	0,9634	0,9247	0,8947	0,9149
[27][28][19] Support Vector Machine (SVM)	0,9546	0,9706	0,9085	0,9395
[4] Logistic Regression (LR)	0,9503	0,9737	0,9089	0,9653
[19][29] Naive Bayes (NB)	0,9074	0,8966	0,7985	0,9010
[26] KNN (K-nearest Neighbors)	0,8920	0,9143	0,8931	0,8853

Fuente: Elaboración propia

Además, la elección de estos algoritmos se justifica por las siguientes razones. El algoritmo DT es simple de interpretar y permite identificar características relevantes para la detección de inyecciones SQL. RF al ser un ensamble de árboles de decisión, ofrece robustez y resistencia al sobreajuste al utilizar múltiples modelos para hacer predicciones, dando

facilidad en la detección de inyecciones, además puede manejar conjuntos de datos desbalanceados de manera más efectiva. Por último, SVM es eficaz en espacios de características de alta dimensión y es menos propenso al sobreajuste en comparación con otros algoritmos de aprendizaje automático, en detecciones de inyección SQL, SVM separa las consultas SQL legitimas de consultas maliciosas en un espacio de características definidopor características relevantes.

Así mismo, para la selección de los algoritmos DT, RF y SVM, se consideró fundamental la evaluación de su aplicabilidad en investigaciones científicas. De la revisión sistemática realizada se observó que el algoritmo DT fue utilizado en 9 estudios, SVM en 10, y RF en 9, lo que denota su relevancia y adopción en el ámbito científico. Por lo tanto, se optó por estos algoritmos en lugar de otros que son menos usados en la industria.

En la siguiente etapa, se implementaron tres algoritmos basados en Machine Learning, elegidos por su alto rendimiento. Estos algoritmos fueron entrenados utilizando el Dataset seleccionado con el objetivo de detectar inyecciones SQL en las peticiones dirigidas a una aplicación desarrollada bajo el enfoque de microservicios.

Los algoritmos implementados en el entorno de trabajo Google Collaboratory fueron SVM, Decision tree y Random forest utilizando el lenguaje de programación Python.

Support Vector Machine

El algoritmo SVM, tipo de aprendizaje supervisado usado para la clasificación de datos, tiene como finalidad identificar un hiperplano óptimo que maximiza la separación entre las clases [30]. Para la detección de las inyecciones SQL, el algoritmo identifica patrones de SQLI en los datos de entrada, aprendiendo a distinguir entre consultas normales y anómalas.

El objetivo del algoritmo SVM es maximizar el margen minimizando errores de clasificación, donde el margen es la distancia entre el límite de decisión (hiperplano) y puntos de datos más cercanos de cada clase [30]. Además, Support Vector tiene un impacto significativo en la precisión de la clasificación del SVM, donde son los puntos que se

encuentran más cerca del límite del hiperplano y determinan la posición y orientación de ladecisión

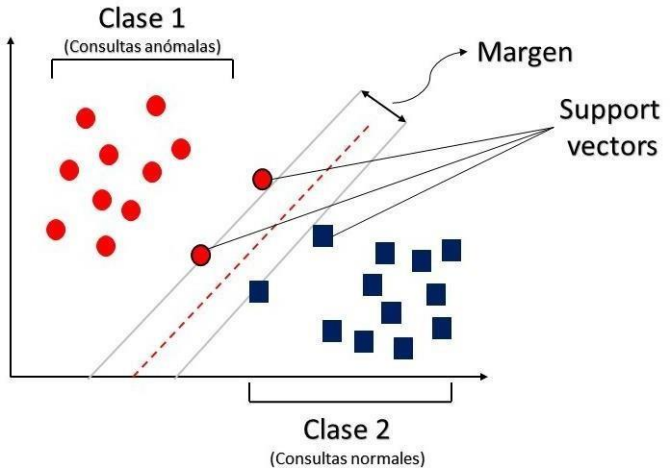


Figura 2. Arquitectura de SVM (Elaboración propia)

El procedimiento del algoritmo abarca desde la fase inicial de preparación de los datos hasta la etapa de predicción. En una primera instancia, se lleva a cabo la carga del conjunto de consultas SQL (Dataset) y la asignación de etiquetas, donde el valor "1" representa una consulta anómala, mientras que "0" indica una consulta normal, en la figura 2 esto se representa en la clase 1 y 2 respectivamente. A continuación, se realiza un preprocesamiento de los datos, eliminando las filas que contengan valores nulos. Posteriormente, se efectúa una división de datos aplicando la regla de Pareto del 80/20, destinando el 80% para el entrenamiento del algoritmo y el 20% para las pruebas. El algoritmo SVM utilizó la codificación 8-bit Unicode Transformation Format (UTF-8) para manejar textos en lenguaje natural que podrían incluir caracteres especiales. Durante el procesamiento de datos, se empleó la función dropna() para eliminar filas con valores faltantes o nulos, asegurando así la integridad del conjunto de datos. La vectorización aplicada en el algoritmo utiliza la técnica TF-IDF, que asigna pesos a las palabras según su importancia en el contenido de texto, este enfoque permite realizar un conteo ponderado de las palabras en las consultas, lo que facilita la determinación de si se trata de una inyección SQL o no. Además, se utilizado un kernel lineal ya que el objetivo es clasificar únicamente si una consulta es una inyección SQL o no, sin requerir una separación no lineal de las clases.

Decision tree

El algoritmo Decision tree, es un modelo de aprendizaje supervisado, empleado en el ámbito de la inteligencia artificial y el aprendizaje automático, su estructura se refleja en una secuencia de decisiones y sus respectivas ramificaciones [31]. Esta configuración es ajustable según la profundidad deseada para la tarea o el nivel óptimo de exploración. Decision Tree consta de un nodo raíz del cual se desprenden nodos hijos que conforman lasramas, y hojas que contienen las predicciones finales.

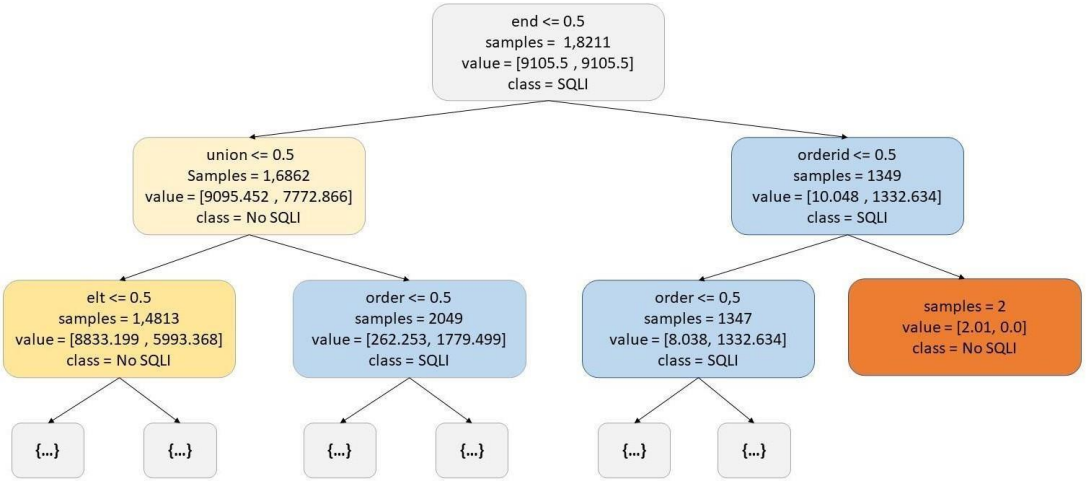


Figura 3. Arquitectura del algoritmo Decision tree (Elaboración propia)

En el contexto específico de la detección de inyecciones SQL, el algoritmo procesa un conjunto de datos etiquetados desde el nodo raíz, utilizando la ganancia de información para generar ramificaciones adicionales (figura 3). Cada bifurcación divide el conjunto de datos en subconjuntos más pequeños, creando nodos hijos. La selección de características se realiza después de cada división, repitiendo el proceso iterativamente hasta alcanzar un criterio de parada, como la profundidad máxima del árbol o la ganancia de información que no supera un umbral predeterminado. Adicionalmente, el algoritmo genera nodos terminales (hojas) que representan predicciones finales, ya sea la clase más común en el subconjunto para problemas de clasificación o la media para problemas de regresión. Este enfoque se destaca por su capacidad de adaptarse a nuevos datos y su interpretabilidad, fundamentales en aplicaciones de seguridad como la detección de inyecciones SQL. La arquitectura de Decision

tree se ajusta por la profundidad que se agrega en la misma, además las formas de las ramas y la creación de hojas se va formando de manera interna en cuanto a si se reconocen o no como inyecciones SQL los datos procesados. Este algoritmo empleó la codificación UTF-8 para asegurar la integridad en el manejo de caracteres especiales, y la función dropna() se utilizó para descartar filas con valores nulos, manteniendo así la calidad de los datos empleados en el entrenamiento del modelo. En cuanto al procesamiento de texto, se aplicaron técnicas de lematización y tokenización mediante la biblioteca Natural Language Toolkit (NLTK), lo que resultó en una normalización y limpieza efectiva de las consultas SQL, mejorando la precisión del modelo en la detección de inyecciones SQL. Se optó por el método CountVectorizer para transformar el texto preprocesado en vectores numéricos interpretables por el algoritmo de aprendizaje automático, lo que facilitó la representación de las características del texto y su utilización en el modelo de árbol de decisión. Además, se realizaron ajustes específicos al modelo, como establecer una profundidad máxima del árbol (max_depth=30) y el uso del parámetro ccp_alpha para prevenir el sobreajuste mediante la penalización y la poda del árbol, lo que favoreció una generalización efectiva del modelo en datos no vistos y mejoró su capacidad predictiva.

Random forest

El algoritmo Random forest de tipo aprendizaje supervisado, destaca por construir múltiples árboles de decisión y combinar sus resultados para mejorar la precisión y mitigar el sobreajuste. Este algoritmo crea un conjunto de árboles de decisión, cada uno entrenado en una submuestra aleatoria del conjunto de datos de entrenamiento [32]. En la detección de inyecciones SQL, RandomForest captura patrones más complejos y mejora la generalizacióndel modelo (figura 4).

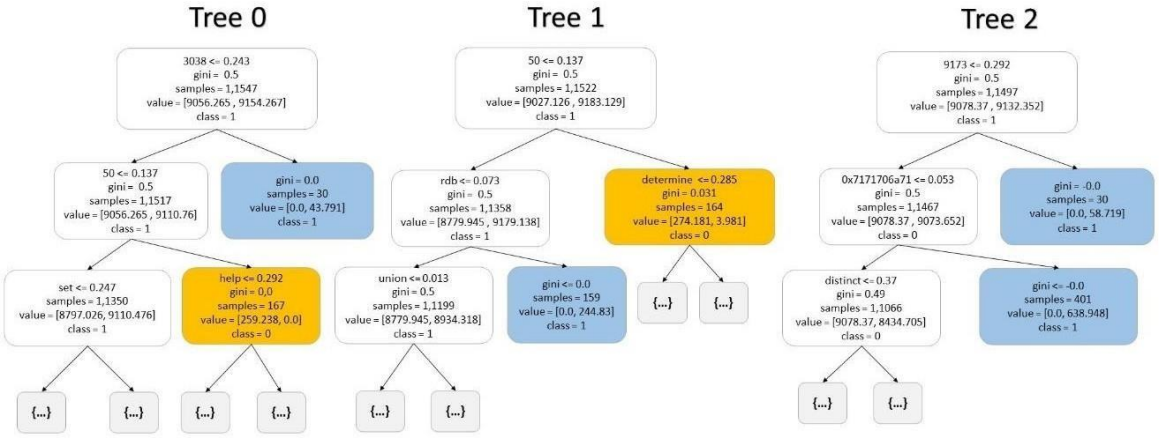


Figura 4. Arquitectura del algoritmo Random forest (Elaboración propia)

El proceso inicia con la preparación de los datos, la eliminación de valores nulos, la división de datos según la ley de Pareto siendo 80 y 20, y la vectorización de las consultas mediante TF-IDF. El entrenamiento del algoritmo se realiza aplicando criterios específicos, como el número de estimadores (n_estimators), la profundidad máxima del árbol (max_depth), el peso de las clases (class_weight) y el estado aleatorio (random_state).

En la tabla 5, se presenta los términos esenciales que son usados en la matriz de confusión, donde se utilizan fórmulas para poder evaluar el rendimiento del algoritmo. Así mismo, la detección de SQLI se puede categorizar como verdadero positivo (VP) o verdadero negativo (VN) cuando se realizan detecciones precisas en los ataques, mientras que se clasifica como falso positivo (FP) o falso negativo (FN) en caso de detecciones incorrectas.

Tabla 5. Términos de matriz de confusión

Término	Sigla
Verdaderos Positivos	VP
Verdaderos Negativos	VN
Falsos Positivos	FP
Falsos Negativos	FN

Fuente: Elaboración propia

Precision (P)

Métrica utilizada para medir la calidad de las predicciones, minimizando falsos positivos y maximizando la cantidad de verdaderos positivos correctamente clasificados.

$$P = \frac{VP}{VP + FP}$$

Recall (R)

Evalúa la eficiencia de clasificación de todos los elementos que pertenecen a una mismaclase.

$$R = \frac{VP}{VP + FN}$$

F1-Score (F)

Indicador que da un balance entre precision y Recall para poder comparar mejor elrendimiento combinado.

$$F = 2 * \frac{(P * R)}{(P + R)}$$

Accuracy (Acc)

Evalúa la predicción que el algoritmo hace correctamente y tenga una clasificación exacta.

$$Acc = \frac{VP + VN}{VP + FN + FP + VN}$$

El sistema desarrollado para la detección de ataques de inyección SQL se fundamenta en una arquitectura de software basada en microservicios, enfoque reconocido por su estructura modular y distribuida, integrada por múltiples componentes interrelacionados y escalables. La arquitectura de software desarrollada en esta investigación está compuesta de 3 servicios independientes con funcionalidades específicas y comunicados entre sí mediante Application Programming Interface (API) bien definidas.

Se desarrolló un servicio con una aplicación frontend que exponía la interfaz necesaria para que el usuario pueda realizar las peticiones en el sistema de información. Esta aplicación fue desarrollada utilizando Angular, un framework basado en TypeScript, facilitando las interacciones del usuario y enviado las peticiones al servicio donde se encuentra la lógica delnegocio.

Otro servicio que compone la arquitectura de software contiene la lógica de negocio, este servicio fue desarrollado utilizando el Spring Framework en Java. Este componente que expone su API de comunicación funciona también como intermediario entre la interfaz de usuario y el servicio de detección de SQLI, su principal responsabilidad radica en gestionar las solicitudes entrantes, validarlas y dirigirlas al servicio adecuado para su procesamiento. Se ha utilizado el sistema de gestión de bases de datos MySQL para almacenar la información relevante para el funcionamiento de la aplicación, y es el componente que recibe la consulta SQL si esta no es considerada como un ataque SQLI.

Para el procesamiento de las inyecciones de SQL, se estableció un servicio dedicado a albergar los modelos de aprendizaje automático entrenados para la detección de SQLI. Este servicio, desarrollado en Python, expone una API desarrollada con el framework FastAPI que facilita la interacción con los modelos basados en los algoritmos SVM, Decision Tree y Random Forest. Estos modelos tienen la capacidad de analizar las consultas recibidas y determinar si presentan patrones asociados con ataques de inyección SQL.

En la Figura 5, se presenta el diagrama del flujo de trabajo de la aplicación basada en la arquitectura de microservicios. El proceso se inicia cuando un usuario realiza una petición a través de la interfaz web, la cual es recibida por la lógica de negocio para una validación inicial. Posteriormente, la solicitud es enviada al servicio de detección de SQLI, donde se aplica el modelo de aprendizaje automático para analizar su contenido, si se detecta un ataque de inyección SQL, se muestra una notificación al usuario indicando la detección de una posible SQLI; de lo contrario, la solicitud sigue su curso normal de procesamiento, es enviada a la base de datos y la respuesta es procesada y presentada al usuario a través de la interfaz de usuario.

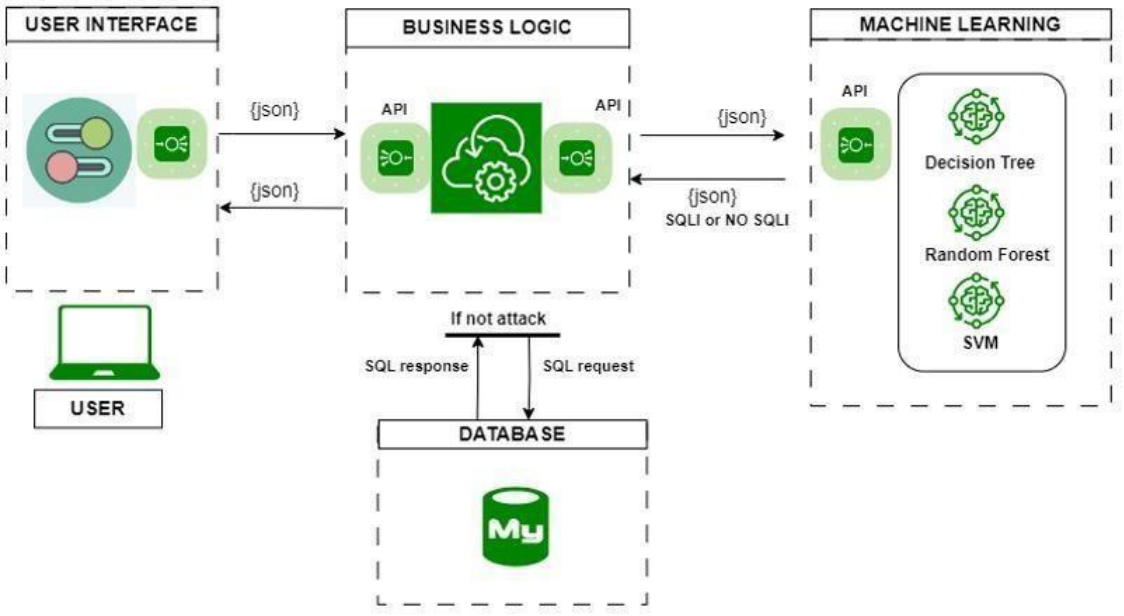


Figura 5. Arquitectura de la aplicación basada en microservicios (Elaboración propia)

III. RESULTADOS Y DISCUSIÓN

3.1. Resultados

Tabla 6. Tiempo de ejecución de los algoritmos de aprendizaje automático evaluados.

Algoritmos de aprendizaje automático	Tiempo de ejecución (Segundos)	Uso de Memoria RAM
Random forest	24	1.5GB
SVM	33	1.7GB
Decisión tree	6	1.4GB

Fuente: Elaboración propia

La tabla 6 evidencia los tiempos de ejecución (en segundos) de los algoritmos utilizados, Random forest, SVM y Decision tree en un mismo entorno de ejecución. Los tiempos varían en función de la estructura y la arquitectura específicas de cada algoritmo. En el caso de Random forest, utiliza un conjunto de 200 árboles de decisión con 15 ramas cada uno, alcanzando un tiempo de entrenamiento estimado de 24 segundos. Por otro lado, el modelo SVM requiere un tiempo de entrenamiento mayor, llegando a los 33 segundos debido a la complejidad de su arquitectura y al procesamiento necesario para abordar las inyecciones. Asimismo, Decision tree, un algoritmo compuesto por un solo árbol para generar predicciones,

presenta un tiempo de entrenamiento de 6 segundos.

Para entrenar los algoritmos de detección de inyecciones SQL se utilizó un Dataset compuesto por 22,764 registros, con 11,382 registros con consultas normales y la misma cantidad de consultas anómalas. Se siguió una estrategia basada en la ley de Pareto, asignando el 80% de los datos para el entrenamiento y el 20% es decir 4553 registros se usaron para las pruebas.

SVM

La figura 6 muestra la matriz de confusión del algoritmo SVM para la detección de inyecciones SQL, indicando que hubo 2298 casos de verdaderos positivos (VP), 2176 casos de verdaderos negativos (VN), 57 casos de falsos positivos (FP) y 22 casos de falsos negativos (FN). En comparación, la tasa de falsos negativos fue del 0.48%, lo que indica que el 0.48% de los casos fueron incorrectamente identificados como negativos.

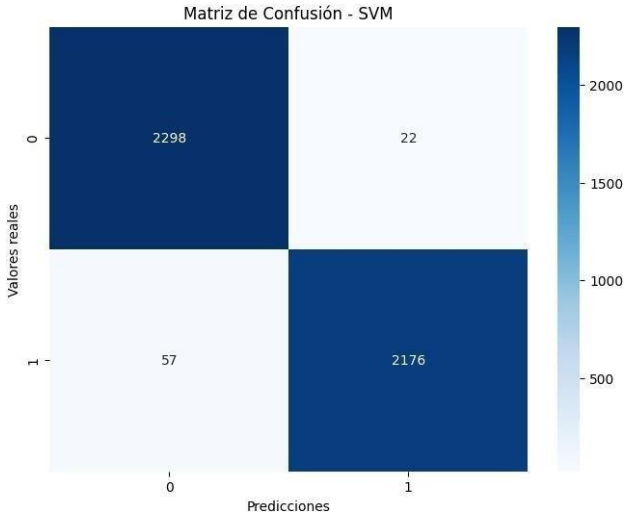


Figura 6. Matriz de confusión del SVM

Random forest

La figura 7 presenta la matriz de confusión del algoritmo Random forest. Se puede apreciar que 2313 casos de verdaderos positivos (VP), 2168 casos de verdaderos negativos (VN), 65 casos de falsos positivos (FP) y 7 casos de falsos negativos (FN). En comparación, la tasa de falsos negativos fue del 0.15%, este sería el porcentaje de los casos fueron incorrectamente identificados como negativos.

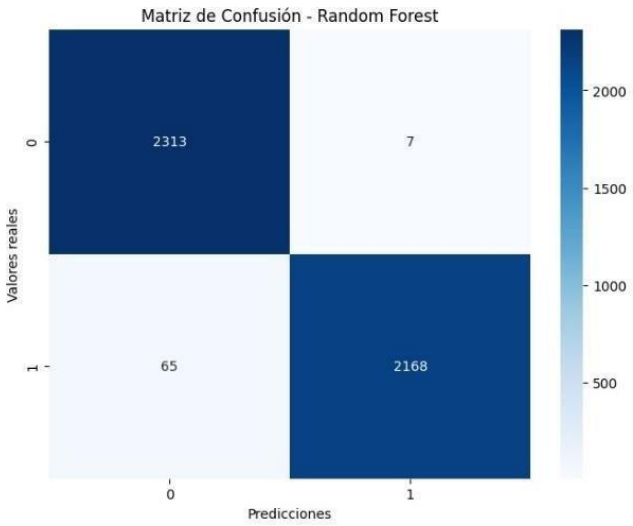


Figura 7. Matriz de confusión de Random forest

Decision tree

La figura 8 muestra la matriz de confusión del algoritmo Decision para detectar inyecciones SQL, indicando que hubo 2276 casos de verdaderos positivos (TP), 1932 casos de verdaderos negativos (TN), 301 casos de falsos positivos (FP) y 44 casos de falsos negativos (FN). En comparación, la tasa de falsos negativos fue del 0.48%, lo que indica que el 0.97% de los casos fueron incorrectamente identificados como negativos.

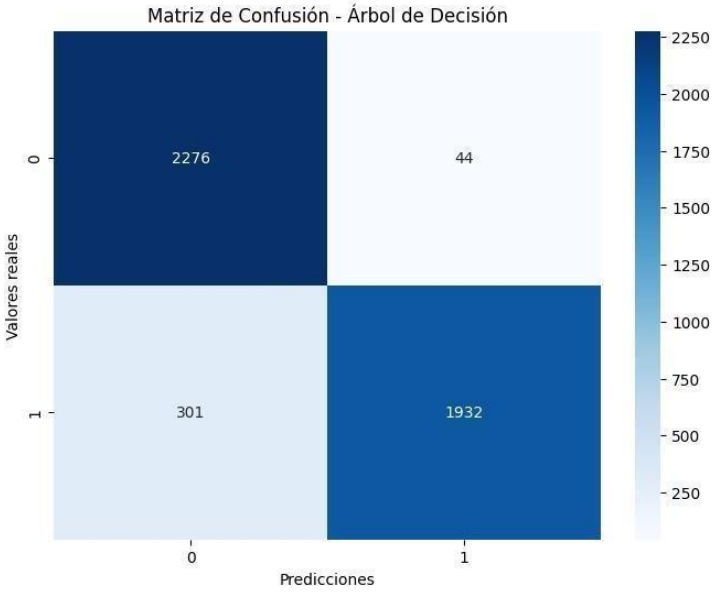


Figura 8. Matriz de confusión de Decision tree

Por lo tanto, en la tabla 4 se aprecia los indicadores de desempeño de los tres algoritmos seleccionados durante la fase de entrenamiento para la detección de SQLI en el microservicio web. En este contexto, los algoritmos fueron entrenados utilizando diversos

modelos de tokenización y vectorización, permitiendo así el procesamiento efectivo de las cadenas que conforman tanto las inyecciones como las no inyecciones. Los resultados revelan que el algoritmo Random forest destaca por su excelente desempeño en la detección de SQLI, logrando una precision y accuracy del 99%, un Recall del 97%, y un 98% de F1- Score. Así mismo, el algoritmo SVM exhibe un rendimiento destacado con una precision, accuracy y F1-Score del 98%, un Recall del 97%. Por el contrario, el algoritmo Decision tree no muestra resultados óptimos para la detección de SQLI, presentando una precision del 97%, un Recall del 86%, un 91% en F1-Score y un 92% en Accuracy.

Tabla 7. Medidas de desempeño de los tres algoritmos de aprendizaje automático

Algoritmos de aprendizaje automático	Precision	Recall	F1-Score	Accuracy
Random Forest	99%	97%	98%	99%
SVM	98%	97%	98%	98%
Decision tree	97%	86%	91%	92%

Fuente: Elaboración propia

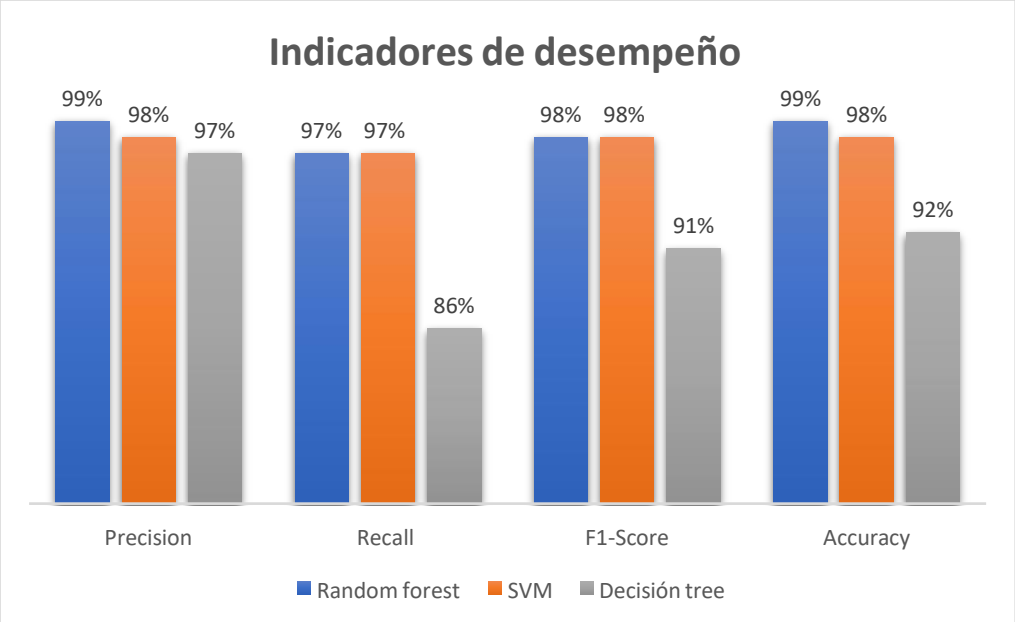


Figura 9. Indicadores de desempeño de los algoritmos seleccionados

3.2. Discusión

Los resultados obtenidos en este estudio, centrado en la comparativa de algoritmos de aprendizaje automático para la detección de diversos tipos de inyecciones SQL en microservicios web, mediante un conjunto de datos clasificados como inyecciones y no inyecciones, poseen implicaciones significativas para el ámbito de la ciberseguridad. Es importante aclarar que, a diferencia de los estudios previos que informan sobre la detección de SQLI en aplicaciones web basadas en arquitecturas monolíticas, la presente investigación se centra en el contexto específico de microservicios web [33], donde la literatura existente aún no se ha profundizado. La arquitectura de software basada en el enfoque de microservicios está compuesta de 3 servicios independientes con funcionalidades específicas, un servicio que expone una interfaz de usuario permitiendo realizar las peticiones SQL, que pueden ser peticiones normales o ataques; un servicio que recibe estas peticiones y las envía al servicio de procesamiento que evalúa la petición utilizando los modelos entrenados con 3 algoritmosde machine learning (SVM, RF y DT).

En los algoritmos SVM y RF se ha utilizado la vectorización mediante TF-IDF para detectar patrones anómalos y manejar cargas de consultas elevadas, aprovechando la versatilidad del vectorizador. Sin embargo, en el tercer algoritmo, se ha utilizado el vectorizador CountVectorizer, una elección que difiere de los enfoques presentados en los estudios previos [34] countVectorizer genera una matriz de recuento de términos, donde cada fila almacena un término único extraído del dataset, lo que facilita la identificación efectiva de inyecciones SQL.

Además, es crucial destacar que en el algoritmo RF se ha implementado Gradient Boosting (GB) [35], esta estrategia no fue abordada en los estudios previos, se emplea para mejorar el rendimiento del modelo y lograr una detección más precisa de inyecciones SQL.

No obstante, durante el desarrollo de este estudio, se identificaron algunas limitaciones. En particular, se destacó la falta de información detallada sobre los ataques a los microservicios web, una arquitectura que las empresas están actualmente valorando y considerando para mejorar sus sistemas de información debido a sus beneficios en términos de redundancia, escalabilidad y flexibilidad, esto puede ser debido a que actualmente este tipo de ataques se mitiga ya que la capa de acceso a datos de los sistemas web se desarrollan utilizando frameworks mapeadores objeto-relacional (ORM), sin embargo las base de datos trabajan recibiendo peticiones SQL, así que todavía es posible realizar este tipo de ataques. Adicionalmente, se encontró que el procesamiento de consultas de SQL y SQLI presentó desafíos considerables, esto se debe a la composición de estas consultas, que involucran aspectos de Procesamiento de Lenguaje Natural (NLP), caracteres especiales, signos, letras y otros elementos. La complejidad inherente a estos elementos ha añadido dificultades al manejo eficiente de las consultas y, por ende, representa una consideración importante para futuras investigaciones y mejoras en la implementación de soluciones, por lo que sugerimos utilizar consultas parametrizadas y desarrollar técnicas más avanzadas para manejar la complejidad inherente de estas consultas.

Las métricas evaluadas, que incluyen Precision, Recall, F1-Score y Accuracy, reflejan resultados alentadores en cuanto a la efectividad para detectar SQLI en los microservicios web. Entre los modelos de Machine Learning aplicados, el análisis revela que el algoritmo Random forest con un promedio de desempeño del 98% en base a las métricas evaluadas, posicionándose como la opción más factible y eficiente para la detección de inyecciones SQL en microservicios web, su uso podría representar una contribución significativa al campo de la ciberseguridad. El algoritmo Random forest ha demostrado obtener métricas superiores gracias a las características y estructura que posee para la clasificación de inyecciones, permitiendo identificar patrones específicos y

realizar predicciones precisas, consolidándose como una herramienta clave en la prevención y detección de amenazas de seguridad en entornos de microservicios web.

Al comparar los resultados con otros estudios, se observa que se centran en la detección de SQLI en aplicaciones web en base a arquitecturas monolíticas, en este contexto, la presente investigación se distingue al aplicar algoritmos de aprendizaje automático específicamente a aplicaciones web basadas en arquitecturas compuestas por servicios independientes, enfoque conocido como microservicios. Así pues, en la investigación de [36], se observa que se realizaron conversiones de datos de entrada (dataset), convirtiendo a valores numéricos y se hizo la verificación de datos para evitar columnas o filas vacías en el entrenamiento, llegando a entrenar seis algoritmos, de los cuales destaca Regresión Lineal y Perceptron+SGD obteniendo a una precision superior a 96%, resultados un tanto similares a los valores obtenidos en los modelos que se han empleado en este estudio. Por otro lado en [37], se propone un modelo de detección de ataques de inyección SQL basado en aprendizaje semántico y aprendizaje profundo, aplicados mediante técnicas de procesamiento de lenguaje natural NLP (TF-IDF y Word2Vec), técnicas también aplicadas en este estudio con el fin de procesar las consultas de SQLI a lenguaje entendible para el algoritmo, los resultados obtenidos en este estudio revelan que el modelo synBERT logró un 99.74% de Accuracy, 99.68% de Precision, un recall de 99.52% y un F1-Score de 99.60%. Por otro lado, se tiene que en
[38] se propone el codificador automático RNN el cual logra obtener una Precision de 95%, Accuracy de 94%, Recall de 90% y F1-Score de 92%.

Los resultados obtenidos sugieren que los algoritmos de aprendizaje automático podrían ser altamente eficaces en la detección de inyecciones SQL en aplicaciones basadas en microservicios web. aumentando así el nivel de seguridad y permitiendo la detección temprana de posibles ataques informáticos. Esta perspectiva apunta a mejorar significativamente el nivel de seguridad, posibilitando la identificación precoz de posibles ataques informáticos. Sin embargo, es importante mencionar que la posibilidad de aplicar

los algoritmos en microservicios web, es un desafío debido a la complejidad de la arquitectura y los componentes de solicitud de datos. La efectividad demostrada del algoritmo Random forest, puede contribuir significativamente al ámbito de la ciberseguridad, abriendo oportunidades para futuras investigaciones y desarrollos en la detección de SQLI en entornos de microservicios web.

IV. CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

- La caracterización de los tipos de ataques de SQL ha posibilitado la identificación de los ocho tipos más comúnmente empleados por los atacantes. Este análisis ha proporcionado una comprensión detallada de sus definiciones y la estructura de las consultas anómalas correspondientes a cada tipo, lo que a su vez ha permitido adoptar medidas defensivas específicas frente a estos ataques.
- La selección del dataset proveniente de repositorios públicos reconocidos como Kaggle y Github, y la aplicación de submuestreo para el balanceado de datos, ha permitido obtener un dataset optimizado para fortalecer el entrenamiento y ejecución de los algoritmos de aprendizaje automático para la detección de SQLI en microservicios web.
- Para la selección de los algoritmos de aprendizaje automático, se llevó a cabo una revisión sistemática de la literatura, de esta manera se obtuvo como resultado que los algoritmos Decision tree, SVM y RandomForest presentaron mejores resultados considerando los criterios de precisión, exactitud, Recall yF1-Score.
- Se ha desarrollado toda la arquitectura de software basado en el enfoque de microservicios, considerando 3 servicios, un servicio que expone la interfaz de usuario para capturar las peticiones SQL, un servicio que recibe las peticiones web y las traslada al servicio de evaluación de la petición para determinar si es un ataque. Además, se ha utilizado el sistema de gestión de bases de datos

MySQL para almacenar la información relevante para el funcionamiento de la aplicación, y es el componente que recibe la consulta SQL si esta no es considerada como un ataque SQLI.

- Para entrenar los algoritmos seleccionados, se llevó a cabo el proceso utilizando Python en el entorno virtual de Google Collaboratory. El dataset estuvo compuesto por 22,764 registros, con 11,382 registros con consultas normales y la misma cantidad de consultas anómalas, se aplicó la ley de Pareto para dividir el conjunto de datos en un 80% destinado al entrenamientoy un 20% para las pruebas.

- El análisis de los resultados confirma que el algoritmo con mejor desempeño para detectar SQLI en microservicios web es el algoritmo Random forest con una Precision y Accuracy del 99%, Recall del 97% y 98% de F1-Score.

4.2. Recomendaciones

- Para conocer sobre algoritmos de aprendizaje automático y su aplicación, es recomendable realizar una revisión bibliográfica en fuentes conocidas como Scopus, ScienceDirect, Taylor & Francis, para un mayor conocimiento en el funcionamiento y la implementación en cualquier tema que se está revisando.

- El uso de las bibliotecas que proporciona Python como pandas, numpy, sklearn, permite evaluar a los algoritmos de aprendizaje automático, así como también para su implementación.

REFERENCIAS

[1] J. Arcila-Diaz and C. Valdivia, "A Microservice-based Software Architecture for Improving the Availability of Dental Health Records," *International Journal of Computing*, vol. 21, no. 4, pp. 475–481, 2022, doi: 10.47839/ijc.21.4.2783.

[2] L. MOCEAN and M.-P. VLAD, "DATABASE SECURITY IN RDF TERMS.," *Buletin Stiintific*, vol. 28, no. 1, pp. 55–65, 2023, [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=164270519&lang=es&site=ehost-live

[3] W. YI-CHUAN, Z. GUI-LING, and Z. YA-LING, "Analysis of SQL Injection Based on Petri Net in Wireless Network.," *Journal of Information Science & Engineering*, vol. 39,no. 1, pp. 167–181, 2023, [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=iih&AN=161511501&lang=es&site=ehost-live

[4] S. M. Shagari, D. Gabi, N. M. Dankolo, and N. N. Gana, "Countermeasure to Structured Query Language Injection Attack for Web Applications using Hybrid Logistic Regression Technique," *Journal of the Nigerian Society of Physical Sciences*, vol. 4, no. 4, Nov. 2022, doi: 10.46481/jnsps.2022.832.

[5] H. Furhad, R. K. Chakrabortty, M. J. Ryan, J. Uddin, and I. H. Sarker, "A hybrid framework for detecting structured query language injection attacks in web-based applications," *International Journal of Electrical and Computer Engineering*, vol. 12, no. 5, pp. 5405–5414, Oct. 2022, doi: 10.11591/ijece.v12i5.pp5405-5414.

[6] A. W. Marashdih, Z. F. Zaaba, and K. Suwais, "Predicting input validation vulnerabilities based on minimal SSA features and machine learning," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, pp. 9311–9331, Nov. 2022, doi: 10.1016/j.jksuci.2022.09.010.

[7] M. Lodeiro-Santiago, C. Caballero-Gil, and P. Caballero-Gil, "Collaborative SQL- injections detection system with machine learning," Sep. 2022, [Online]. Available:

http://arxiv.org/abs/2209.06553

[8] Y. Li and B. Zhang, “Detection of SQL Injection Attacks Based on Improved TFIDF Algorithm,” in *Journal of Physics: Conference Series*, Institute of Physics Publishing, Nov. 2019. doi: 10.1088/1742-6596/1395/1/012013.

[9] P. Tang, W. Qiu, Z. Huang, H. Lian, and G. Liu, “Detection of SQL injection based on artificial neural network,” *Knowl Based Syst*, vol. 190, Feb. 2020, doi: 10.1016/j.knosys.2020.105528.

[10] B. A. Meharaj Begum and M. Arock, “Efficient Detection of SQL Injection Attack(SQLIA) Using Pattern-based Neural Network Model,” in *Proceedings - IEEE 2021 International Conference on Computing, Communication, and Intelligent Systems, ICCCIS 2021*, Institute of Electrical and Electronics Engineers Inc., Feb. 2021, pp. 343–347. doi: 10.1109/ICCCIS51004.2021.9397066.

[11] N. Gandhi, J. Patel, R. Sisodiya, N. Doshi, and S. Mishra, “A CNN-BiLSTM based Approach for Detection of SQL Injection Attacks,” in *Proceedings of 2nd IEEE International Conference on Computational Intelligence and Knowledge Economy, ICCIKE 2021*, Institute of Electrical and Electronics Engineers Inc., Mar. 2021, pp. 378– 383. doi: 10.1109/ICCIKE51210.2021.9410675.

[12] K. Zhang, “A machine learning based approach to identify SQL injection vulnerabilities,” in *Proceedings - 2019 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019*, Institute of Electrical and Electronics Engineers Inc., Nov. 2019, pp. 1286–1288. doi: 10.1109/ASE.2019.00164.

[13] C. W. Tien, T. Y. Huang, C. W. Tien, T. C. Huang, and S. Y. Kuo, “KubAnomaly: Anomaly detection for the Docker orchestration platform with neural network approaches,” *Engineering Reports*, vol. 1, no. 5, Dec. 2019, doi: 10.1002/eng2.12080.

[14] F. G. Deriba, A. O. Salau, S. H. Mohammed, T. M. Kassa, and W. B. Demilie, “Development of a Compressive Framework Using Machine Learning Approaches for SQL Injection Attacks,” *Przeglad Elektrotechniczny*, vol. 98, no. 7, pp. 181–187, 2022, doi: 10.15199/48.2022.07.30.

[15] Ö. Kasim, “An ensemble classification-based approach to detect attack level of SQL injections,” *Journal of Information Security and Applications*, vol. 59, Jun. 2021, doi: 10.1016/j.jisa.2021.102852.

[16] N. Joshi Padma, N. Ravishankar, M. B. Raju, and N. C. Ravi, “SURGICAL STRIKING SQL INJECTION ATTACKS USING LSTM,” *Indian Journal of Computer Science and Engineering*, vol. 13, no. 1, pp. 208–220, Jan. 2022, doi: 10.21817/indjcse/2022/v13i1/221301182.

[17] J. M. Alkhathami and S. M. Alzahrani, “DETECTION OF SQL INJECTION ATTACKS USING MACHINE LEARNING IN CLOUD COMPUTING PLATFORM,” *J Theor Appl Inf Technol*, vol. 15, no. 15, 2022, [Online]. Available: www.jatit.org

[18] U. Farooq, “Ensemble Machine Learning Approaches for Detection of SQL Injection Attack,” *Tehnički glasnik*, vol. 15, no. 1, pp. 112–120, Mar. 2021, doi: 10.31803/tg-20210205101347.

[19] W. B. Demilie and F. G. Deriba, “Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques,” *J Big Data*, vol. 9, no. 1, Dec. 2022, doi: 10.1186/s40537-022-00678-0.

[20] Sanshui, “SQL Injection Detection by Machine learning.” Accessed: Dec. 20, 2023. [Online]. Available: https://www.kaggle.com/code/sanshui123/sql-injection-detection- by-machine-learning/input?select=Modified_SQL_Dataset.csv

[21] S. Park, “MachineLearning.” Accessed: Dec. 23, 2023. [Online]. Available: https://github.com/Scott-Park/MachineLearning

[22] I. Tasdelen, “sql-injection-payload-list.” Accessed: Dec. 23, 2023. [Online]. Available: https://github.com/payloadbox/sql-injection-payload-list

[23] D. Galbreath, “libinjection.” Accessed: Dec. 23, 2023. [Online]. Available: https://github.com/client9/libinjection

[24] KAGGLE KERNELER, “Starter: SQL-Injection Payload e7fb1937-2.” Accessed: Dec. 23, 2023. [Online]. Available: https://www.kaggle.com/code/kerneler/starter-sql-injection-payload-e7fb1937-2/input

[25] S. Hussain, “sql injection dataset.” Accessed: Dec. 23, 2023. [Online]. Available: https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset

[26] V. Devalla, S. Srinivasa Raghavan, S. Maste, J. D. Kotian, and Dr. D. Annapurna, “mURLi: A Tool for Detection of Malicious URLs and Injection Attacks,” *Procedia Comput Sci*, vol. 215, pp. 662–676, 2022, doi: 10.1016/j.procs.2022.12.068.

[27] A. A. Ashlam, A. Badii, and F. Stahl, “A Novel Approach Exploiting Machine Learning to Detect SQLi Attacks,” in *Proceedings of the 2022 5th International Conference on Advanced Systems and Emergent Technologies, IC_ASET 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 513–517. doi: 10.1109/IC_ASET53395.2022.9765948.

[28] H. Fu, C. Guo, C. Jiang, Y. Ping, and X. Lv, “SDSIOT: An SQL Injection Attack Detection and Stage Identification Method Based on Outbound Traffic,” *Electronics (Switzerland)*, vol. 12, no. 11, 2023, doi: 10.3390/electronics12112472.

[29] C. Zhao, S. Si, T. Tu, Y. Shi, and S. Qin, “Deep-Learning Based Injection Attacks Detection Method for HTTP,” *Mathematics*, vol. 10, no. 16, 2022, doi: 10.3390/math10162914.

[30] T. Pankaj, “Support Vector Machines (SVM): An Intuitive Explanation.” Accessed: Dec. 23, 2023. [Online]. Available: https://medium.com/low-code-for-advanced-data-science/support-vector-machines-svm-an-intuitive-explanation-b084d6238106

[31] K. Dissanayake, “Machine Learning Algorithms(8) — Decision Tree Algorithm.” Accessed: Dec. 23, 2023. [Online]. Available: https://medium.com/towardsdev/machine-learning-algorithms-8-decision-tree-algorithm-533b6926ddbb

[32] K. Dissanayake, “Machine Learning Algorithms(9) - Ensemble techniques (Bagging - Random Forest Classifier and Regression).” Accessed: Dec. 23, 2023. [Online]. Available: https://medium.com/towardsdev/machine-learning-algorithms-9-ensemble- techniques-bagging-random-forest-classifier-and-5d3747c7a953

[33] S. Baškarada, V. Nguyen, and A. Koronios, “Architecting Microservices: Practical

Opportunities and Challenges," *Journal of Computer Information Systems*, vol. 60, no. 5, pp. 428–436, Sep. 2020, doi: 10.1080/08874417.2018.1520056.

[34] A. A. Ashlam, A. Badii, and F. Stahl, "A Novel Approach Exploiting Machine Learning to Detect SQLi Attacks," in *Proceedings of the 2022 5th International Conference on Advanced Systems and Emergent Technologies, IC_ASET 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 513–517. doi: 10.1109/IC_ASET53395.2022.9765948.

[35] A. Callens, D. Morichon, S. Abadie, M. Delpey, and B. Liquet, "Using Random forest and Gradient boosting trees to improve wave forecast at a specific location," *Applied Ocean Research*, vol. 104, Nov. 2020, doi: 10.1016/j.apor.2020.102339.

[36] I. S. Crespo-Martínez, A. Campazas-Vega, Á. M. Guerrero-Higueras, V. Riego- DelCastillo, C. Álvarez-Aparicio, and C. Fernández-Llamas, "SQL injection attack detection in network flow data," *Comput Secur*, vol. 127, 2023, doi: 10.1016/j.cose.2023.103093.

[37] D. Lu, J. Fei, and L. Liu, "A Semantic Learning-Based SQL Injection Attack Detection Technology," *Electronics (Basel)*, vol. 12, no. 6, p. 1344, Mar. 2023, doi: 10.3390/electronics12061344.

[38] M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Deep Learning Architecture for Detecting SQL Injection Attacks Based on RNN Autoencoder Model," *Mathematics*, vol. 11, no. 15, 2023, doi: 10.3390/math11153286.

ANEXOS

Anexo 1: Acta de revisión de similitud de la investigación

Universidad Señor de Sipán

Acta de revisión de similitud de la investigación

Yo **Víctor Alexci Tuesta Monteza** docente del curso de **Investigación II** del Programa de Estudios de **Ingeniería de Sistemas** y revisor de la investigación del (los) estudiante(s), **Peralta García Edwin Jahir, Quevedo Monsalbe Juan Carlos**, titulada:

ANÁLISIS COMPARATIVO DE ALGORITMOS DE APRENDIZAJE AUTOMÁTICO PARA DETECTAR ATAQUES DE SQLI EN MICROSERVICIOS WEB

Se deja constancia que la investigación antes indicada tiene un índice de similitud del 11%, verificable en el reporte final del análisis de originalidad mediante el software de similitud TURNITIN. Por lo que se concluye que cada una de las coincidencias detectadas no constituyen plagio y cumple con lo establecido en la Directiva sobre índice de similitud de los productos académicos y de investigación en la Universidad Señor de Sipán S.A.C., aprobada mediante Resolución de Directorio N° 145-2022/PD-USS.

En virtud de lo antes mencionado, firma:

Tuesta Monteza Victor Alexci	42722929	

Pimentel, 22 de diciembre de 2023.

Anexo 2. Acta de aprobación del asesor

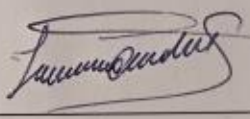
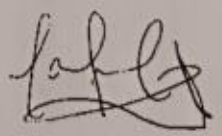
ANEXO 2: ACTA DE APROBACIÓN DEL ASESOR

Universidad Señor de Sipán

ACTA DE APROBACIÓN DEL ASESOR

Yo **Arcila Díaz Juan Carlos** quien suscribe como asesor designado mediante Resolución de Facultad **N° 0426-2023/FIAU-USS**, del proyecto de investigación titulado **ANÁLISIS COMPARATIVO DE ALGORITMOS DE APRENDIZAJE AUTOMÁTICO PARA DETECTAR ATAQUES DE SQLI EN MICROSERVICIOS WEB**, desarrollado por los estudiantes: **PERALTA GARCÍA EDWIN JAHIR, QUEVEDO MONSALBE JUAN CARLOS**, del programa de estudios de **Ingeniería de Sistemas**, acredito haber revisado, realizado observaciones y recomendaciones pertinentes, encontrándose expedito para su revisión por parte del docente del curso.

En virtud de lo antes mencionado, firman:

Arcila Díaz Juan Carlos **(Asesor)**	47715777	
Peralta García Edwin Jahir **(Autor 1)**	72517313	
Quevedo Monsalbe Juan Carlos **(Autor 2)**	75724740	

Pimentel, 10 de julio del 2024

Anexo 3. Versión primigenia enviada a la revista en español

Enlace: https://drive.google.com/drive/folders/1lVDR38jeaTcdrCwSBTTDVi1PW4-jtIPX?usp=sharing

Anexo 4. Cartas de respuesta a las observaciones de los revisores

Enlace:

https://drive.google.com/drive/folders/1X3rUmHxISqvkUqexIteWzdatUkO7iADK?usp=sharing

Anexo 5. Carta de aceptación del Articulo

informatics

an Open Access Journal by MDPI

CITESCORE 4.8

IMPACT FACTOR 3.1

CERTIFICATE OF ACCEPTANCE

The certificate of acceptance for the manuscript (**informatics-2848755**) titled:
Detection of SQL injections in web microservices using machine learning

Authored by:
Edwin Peralta-Garcia; Juan Quevedo-Monsalbe; Victor Tuesta-Monteza; Juan Arcila-Diaz

was accepted in *Informatics* (ISSN 2227-9709) on 21 March 2024

MDPI Academic Open Access Publishing since 1996

Basel, March 2024

Prof. Dr. Antony Bryant
Editor-in-Chief

Anexo 5. Tabla de matriz de consistencia lógica

Titulo	**Análisis comparativo de algoritmos de aprendizaje automático para detectar ataques de SQLI en microservicios web**				
Problema	**Hipótesis**	**Objetivo General**	**Objetivos Específicos**	**Tipo de Investigación**	**Diseño de Investigación**
¿Qué algoritmo de Machine Learning presenta mejor desempeño para detectar SQLI en microservicios web?	El algoritmo Random Forest presenta mejor desempeñopara la detección de inyecciones SQL en microservicios web.	Analizar comparativamente los algoritmos de aprendizaje automático para detectar ataques de SQLI en microservicios web	**a.** Clasificar los ataques de SQL injection en base de datos. **b.** Determinar el Dataset para entrenamiento de aprendizaje automático. c**.** Seleccionar los algoritmos de aprendizaje automático para detectar SQL injection. d**.** Desarrollar un microservicio web. e**.** Implementar los algoritmos seleccionados. f. Analizar los resultados	Tecnológica aplicada	Cuasi-experimental

Anexo 6. Tabla de matriz de operacionalización

Variables de estudio	Definición conceptual	Definición operacional	Dimensiones	Indicadores	Ítem	Instrumento	Valores Finales	Tipo de variable	Escala de medición
Variable independiente Algoritmos de aprendizaje automático	Los algoritmos de aprendizaje automático son una rama de la IA,el cual permite desarrollar técnicas que permitan que las computadoras puedan aprender y procesar cantidad de datos, para devolver una respuesta rápida y eficiente.	Para conocer el desempeño de los algoritmos se debe seleccionar un Dataset para su entrenamiento. Se debe ajustar los parámetros del algoritmo para poder obtener un desempeño eficiente. Se evalúa a los algoritmos usando diferentes métricas como precision, acurracy, Recall, entre otros más. Se debe comparar los resultados con	Consumo de recursos	Tiempo de Respuesta	$TR = \sum_{j}^{n} \frac{tf_j - tf_i}{n}$	Ficha de observación	Minutos (Min)	El tipo de variable será numérica	Proporcional (0% – 100%)
				Consumo de Memoria	$CM = \sum_{i}^{n} \frac{cm_i}{n}$		Gigabyte (GB)		Razón (0GB – 15GB)

		diferentes algoritmos paraconocer cual presenta mejor desempeño.							
Variable dependiente Detección de ataques de SQLI	Los ataques de SQLI son consultas maliciosas que se realizan en base de datos para poder acceder a la información privada de una empresa.	Se tiene que clasificar los tipos de ataques de SQLI que existen, para que después se puedan clasificar y entrenar a los algoritmos. Se construye un Dataset de SQLI para luego realizar pruebas de detección con los algoritmos.	Rendimiento	Precision	$P = \frac{VP}{VP + FP}$	Ficha de observación	Unidad de medida en porcentaje (%)	Numérica	0% - 100%
				Acurracy	$E = \frac{VP + VN}{VP + FN + FP + VN}$				
				Recall	$R = \frac{VP}{VP + FN}$				
				F-Score	$F = 2 * \frac{P * R}{R + R}$				

Anexo 7: Tipos de SQLIA

Types of SQLIA	
Tautology SQLI	
Ilegal/Logically incorrect query SQLI	
Union query SQLI	
Piggy-Backend Query SQLI	
Inference SQLI:	Blind SQLI
	Timing SQLI
Store Procedure SQLI	
Alternate Encoding SQLI	

Anexo 8: Listado de artículos científicos seleccionados

Título	Algoritmos	Criterios			
		Acurracy	Precision	Recall	F1-Score
Countermeasure to Structured Query Language Injection Attack for Web Applications using Hybrid Logistic Regression Technique	Modelo ITFIDF basado en el algoritmo supervisado LR (Regresión logística)	0,99781	0,99782	0,99781	0,99781
DETECTION OF SQL INJECTION ATTACKS USING MACHINE LEARNING IN CLOUD COMPUTING PLATFORM	SVM (Support Vector Machine), KNN (K-nearest Neighbors), Decision Tree, MNB (Multinomial Naive Bayes)	-	SVM= 0,9942 Decision= 0,994 MNB= 0,9709 KNN= 0,9245	SVM= 0,99 Decision= 0,99 MNB= 0,96 KNN= 0,92	SVM= 0,99 Decision= 0,99 MNB= 0,96 KNN= 0,91
Predicting input validation vulnerabilities based on minimal SSA features and machine learning	Método basado con el algoritmo **LSTM**	0,9867	0,979	0,981	
Development of a Compressive Framework Using Machine Learning Approaches for SQL Injection Attacks	Naive Bayes Decision Tree SVM ANN (Artificial Neural Network)	SVM= 0,9843 Decision= 0,9526 Naive= 0,87301 ANN= 0,9916	SVM= 0,9681 Decision= 0,9413 Naive= 0,872 ANN= 0,9852	-	-
SURGICAL STRIKING SQL INJECTION ATTACKS USING LSTM	**LSTM**	0,9372	0,90	0,903	0,9066
DETECTING COMMON WEB ATTACKS BASED ON SUPERVISED MACHINE LEARNING USING WEB LOGS	SVM Random Forest Decision Tree	SVM= 0,9855 Random Forest = 0,9968 Decision Tree = 0,9907	-	-	SVM= 0,9810 Random Forest= 0,9957 Decision Tree= 0,9876

Machine Learning-Based Technique to Detect SQL Injection Attack	Boyer's Moore	0,931	0,888	1	0,888
Detection of SQL injection based on artificial neural network	MLP (Multilayer perceptron) (01, 02, 03 CAPAS OCULTAS) LSTM	MLP1= 0,9967 MLP2= 0,9955 MLP3= 0,9975 LSTM= 0,9917	MLP1= 0,9624 MLP2= 0,9575 MLP3= 0,9727 LSTM= 0,9110	MLP1= 1 MLP2= 0,9902 MLP3= 0,9988 LSTM= 0,9988	-
LSTM-Based SQL Injection Detection Method for Intelligent Transportation System	LSTM	0,9153	0,9132	0,9089	0,9110
SQL Injection Detection for Web Applications Based on Elastic-Pooling CNN	Elastic-Poling CNN ()	0,9856	0,9821	0,984	0,9832
A SQL Injection Detection Method Based on Adaptive Deep Forest	ADF con AdaBoost	0,98	0,97	0,98	0,98
Detecting SQL Injection Attacks in Cloud SaaS using Machine Learning	Random Forest AdaBoost Decision Tree	Random Forest= 0,998 AdaBoost= 0,995 Decision Tree= 0,995	Random Forest= 0,985 AdaBoost= 0,989 Decision Tree= 0,998	Random Forest= 0,994 AdaBoost= 0,996 Decision Tree= 0,994	Random Forest= 0,995 AdaBoost= 0,986 Decision Tree= 0,983
Detecting SQL Injection Attacks Using Grammar Pattern Recognition and Access Behavior Mining	K-means, NavieBayes, SVM, RandomForest				
A Machine Learning based Approach to Identify SQL Injection Vulnerabilities	Decision Tree, RandomForest, SVM, LSTM	Decision Tree= 0,934 RandomForest= 0,936 SVM= 0,954 LSTM= 0,952	Decision Tree= 0,766 RandomForest= 0,774 SVM= 0,986 LSTM= 0,919	Decision Tree= 0,565 RandomForest= 0,577 SVM= 0,583 LSTM= 0,614	Decision Tree= 0,650 RandomForest = 0,660 SVM= 0,732 LSTM= 0,734
A CNN-BiLSTM based Approach for Detection of SQL Injection Attacks	CNN-BiLSTM basada con **CNN** y **LSTM**	0,987	0,98	0,99	0,987
A Novel Approach Exploiting Machine Learning to Detect SQLi Attacks	SVM, Decision Tree, K-Nearest Neighbors, AdaBoost, RandomForest		SVM= 0,94 DT= 0,93 KNN= 0,92 AdaBoost= 0,92 RF= 0,90	SVM= 0,91 DT= 0,89 KNN= 0,91 AdaBoost= 0,89 RF= 0,91	SVM= 0,94 DT= 0,91 KNN= 0,91 AdaBoost= 0,90 RF= 0,90
Detection of SQL Injection Attacks: A Machine Learning Approach	Ensemble Boosted Trees, Ensemble Bagged Trees, Linear Discriminat	Boosted= 0,938 Bagged= 0,938 Linear= 0,937	-	-	-
A Semantic Learning-Based SQL Injection Attack Detection Technology	synBERT	0,985	0,986	0,975	0,987
Deep Learning Architecture for Detecting SQL Injection	Método basado en el algoritmo **RNN**	0,94	0,95	0,9	0,92

Attacks Based on RNN Autoencoder Model					
SDSIOT: An SQL Injection Attack Detection and Stage Identification Method Based on Outbound Traffic	SVM, KNN, DT, RF	SVM= 0,9807 KNN= 0,9890 DT= 0,9841 RF= 0,9799	SVM= 0,9712 KNN= 0,9844 DT= 0,9742 RF= 0,9713	SVM= 0,9921 KNN= 0,9936 DT= 0,9889 RF= 0,9894	SVM= 0,9816 KNN= 0,9890 DT= 0,9814 RF= 0,9802
Analyzing SQL payloads using logistic regression in a big data environment	Enfoque basado con **regresión logística (RL)**	0,9802	0,9805	0,9802	0,9802
An Improved LSTM-PCA Ensemble Classifier for SQL Injection and XSS Attack Detection	Sistema propuesto con **LSTM**	Propuesta= 0,9685 LSTM= 0,9050	Propuesta= 0,9700 LSTM= 0,9530	Propuesta= 0,9700 LSTM= 0,9050	Propuesta= 0,9700 LSTM= 0,9120
Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques	Navies Bayes(NB), Decision Tree (DT), SVM, Random Forest (RF), ANN (Artificial Neural Network)	NB = 0,8720 DT= 0,9480 SVM= 0,9730 RF= 0,9340 ANN= 0,9870	NB = 0,8753 DT= 0,9172 SVM= 0,9640 RF= 0,9435 ANN= 0,9887	NB = 0,8637 DT= 0,9084 SVM= 0,9561 RF= 0,9306 ANN= 0,9913	NB = 0,8695 DT= 0,9128 SVM= 0,96 RF= 0,9370 ANN= 0,99
WEB ATTACK PREDICTION USING STEPWISE CONDITIONAL PARAMETER TUNING IN MACHINE LEARNING ALGORITHMS WITH USAGE DATA	KNN, SVM, DT	KNN= 0,904 SVM= 0,835 DT= 0,963	-	-	-
Deep-Learning Based Injection Attacks Detection Method for HTTP	Propuesta con DL (Deep Learning), SVM, Logistic, Naive Bayes	Propuesta= 0,9939 RL= 0,7981 NB= 0,8513 SVM= 0,9279 RF= 0,9726	Propuesta= 0,9951 RL= 0,976 NB= 0,9327 SVM= 0,9503 RF= 0,9651	Propuesta= 0,9937 RL= 0,6937 NB= 0,5716 SVM= 0,9347 RF= 0,9364	-
mURLi: A Tool for Detection of Malicious URLs and Injection Attacks	RF, XGBoost, AdaBoost, KNN	XGBoost= 0,9690 AdaBoost= 0,9571 KNN= 0,9571	XGBoost= 0,9565 AdaBoost= 0,9098 KNN= 0,9237	XGBoost= 0,9322 AdaBoost= 0,9407 KNN= 0,9237	RF= 0,9505 XGBoost= 0,9442 AdaBoost= 0,9250 KNN= 0,9237
Machine Learning for Securing Traffic in Computer Networks	RF, LR, CN2, NN (Neural Networks)	RF= 0,99 LR= 0,98 CN2= 0,94 NN= 0,50	RF= 0,98 LR= 0,94 CN2= 0,82 NN= 0,27	RF= 0,98 LR= 0,90 CN2= 0,71 NN= 0,28	RF= 0,98 LR= 0,91 CN2= 0,63 NN= 0,27
SQL Injection Attack Detection Using Machine Learning Algorithm	LR, NB, SMO, J48	LR= 0,9726 NB= 0,9823 SMO= 0,9923 J48= 0,9723	-	-	-
Ensemble Machine Learning Approaches for Detection of SQL Injection Attack	GBM, AdaBoost, XGBM, LGBM	GBM= 0,9904 AdaBoost= 0,9910 XGBoost= 0,9922 LGBM= 0,9933	GBM= 0,9898 AdaBoost= 0,9906 XGBoost= 0,9914 LGBM= 0,9933	GBM= 0,9903 AdaBoost= 0,9891 XGBoost= 0,9905 LGBM= 0,9933	GBM= 0,9910 AdaBoost= 0,9893 XGBoost= 0,9922 LGBM= 0,9933
Detection of SQL Injection Attacks Based on Improved TFIDF Algorithm	SVM		0,9908	0,9934	0,9921

SQL injection attack detection in network flow data	LR, LSVC, RF, KNN	LR= 0,973 LSVC= 0,834 RF= 0,840 KNN= 0,718	LR= 0,974 LSVC= 0,873 RF= 0,879 KNN= 0,819	LR=973 LSVC= 834 RF= 840 KNN= 718	LR= 973 LSVC= 829 RF= 836 KNN= 694
Multi-Phase Algorithmic Framework to Prevent SQL Injection Attacks using Improved Machine learning and Deep learning to Enhance Database security in Real-time	CNN, NB	CNN= 0,9773 NB= 0,9583	CNN= 0,9298 NB= 0,9063	CNN= 0,9941 NB= 0,9603	CNN= 0,9636 NB= 0,9325