UNIVERSIDAD DE LA CIUDAD DE AGUASCALIENTES

MAESTRÍA EN CIENCIA DE DATOS

BASES DE DATOS PARA CIENCIA DE DATOS

“Detección de anomalías mediante aprendizaje automático en tráfico de servidores web”

MITSIU ALEJANDRO CARREÑO SARABIA

Periodo Agosto 2024 - Diciembre 2024, Aguascalientes, Ags

Resumen

En este trabajo se aplicaron diversas técnicas y tecnologías de bases de datos, sistemas distribuidos y aprendizaje automático para generar un sistema integral de monitoreo y detección de tráfico anómalo en servidores web que permita entrenar modelos para reconocer y detectar la actividad anómala y aplicar dicho reconocimiento en datos nuevos a escala comercial. Se emplean las tecnologías Apache Spark, MinIO, y ElasticSearch que permiten procesamiento en alto rendimiento para realizar las transformaciones y agregaciones necesarias, el algoritmo principal para la detección de anomalías es el bosque de aislamiento implementado en scala acompañado de técnicas de procesamiento de lenguaje natural y regresión logística. Con estos frameworks y algoritmos se logró construir un sistema que cubre el entrenamiento de modelos y sienta las bases para un sistema productivo de detección de anomalías en tiempo real suave, permitiendo a empresas, monitorear y detectar la actividad de sus servidores de manera automática.

Palabras clave: Detección de anomalías, cómputo distribuido, aprendizaje no supervisado, big data, Bosque de aislamiento, Apache spark.

ÍNDICE DE CONTENIDO

I. PLANTEAMIENTO DEL PROBLEMA

En los últimos años hemos experimentado una impresionante expansión y adopción de servicios a través de internet, por ejemplo compras en línea, redes sociales, plataformas de entretenimiento entre otras, el tráfico total de internet ha experimentado un crecimiento dramático en las últimas décadas, en 2017 el tráfico global de internet alcanzó más de 45,000 Gb por segundo. (Cisco, 2017)

Una consecuencia de este aumento en el tráfico de internet es que los servidores web deben ser capaces de manejar un mayor número de conexiones, permitiendo el correcto intercambio de información con una mayor cantidad de clientes.

El origen de dicho tráfico puede ser generado por peticiones de usuarios reales, peticiones de bots automatizados y peticiones de usuarios con intenciones maliciosas siendo que cada uno de estos grupos se comportan de maneras específicas. Esto se traduce en un rápido crecimiento en el volumen de logs (archivos de registro). Para manejar estos volúmenes de logs de manera eficiente y efectiva, una línea de investigación se centra en desarrollar técnicas inteligentes y automatizadas de análisis de datos. (Zhu, 2023)

I.1 Problemática

El tráfico de un servidor web provee datos confiables sobre accesos, solicitudes, y procesamiento de peticiones, además permite analizar el contexto bajo el que los clientes hacen uso de los recursos, pero el volumen de información generada es tan grande que un análisis manual no es viable. Entender los usos típicos y diferenciarlos de los atípicos puede ser una herramienta poderosa que aplicada en tiempo real permitirá mejorar la calidad y resguardo de la información contenida en el servidor.

Analizar los registros de tráfico web permite no solo entender la manera en que se consume la información, sino también detectar si el uso generalizado se transforma, o si existen anomalías.

A pesar de continuos esfuerzos por regular y definir políticas y organismos enfocados en cuidar el ciberespacio como la Policía Cibernética o el Centro Nacional de Respuesta a Incidentes CIbernéticos (CERT-MX), encargado de vigilar virtualmente la infraestructura tecnológica de la nación (Dominguez, 2023). El Banco Interamericano de Desarrollo los considera insuficientes y recomienda centrarse en mejorar el despliegue de estándares de seguridad cibernética y controles técnicos, así como fomentar el desarrollo de un mercado de ciberseguridad. Resaltando que las disposiciones, organismos y regulaciones son limitadas y dejan varias lagunas, lo que dificulta la lucha contra el cibercrimen (BID, 2020)

Generar herramientas de detección de anomalías puede tener un gran impacto en México, donde el panorama de ciberseguridad es preocupante, ya que en 2018 México fue uno de los cinco países de América Latina que recibió más ataques ransomware (infección de sistemas informáticos mediante virus que bloquean una computadora) (Dominguez, 2023)

I.2 Objetivos

El objetivo de este proyecto es explorar la implementación de técnicas heurísticas, así como de aprendizaje automático para determinar si la actividad y tráfico de un servidor web es anómala, generando un sistema integral de monitoreo y detección de tráfico anómalo que sea capaz de analizar grandes cantidades de datos de manera automática, y a la vez permitir la constante actualización de patrones, ajustando el concepto de comportamiento normal y detectando nuevas anomalías.

I.2.1 Objetivos específicos

Para poder realizar un sistema integral de monitoreo es necesario cumplir los siguientes objetivos:

- Desarrollar o implementar un algoritmo que permita la detección de anomalías que sea tolerante a grandes cantidades de datos y ofrezca resultados de calidad en un tiempo manejable.
- Desarrollar una infraestructura que permita el entrenamiento y alojamiento de múltiples modelos, dando flexibilidad a la temporalidad del análisis.
- Desarrollar una infraestructura que permita alojar múltiples clientes, posibilitando la escalabilidad horizontal.

II. METODOLOGÍA Y PLAN DE ACCIÓN

Para la realización de este trabajo, se consideró la arquitectura, tecnología y especificaciones de la empresa Designa, la cuál ofrece desarrollo de software a la medida, alojamiento y administración de cómputo en la nube e implementación de tecnología eficiente e innovadora, derivado de ello se producen y alojan múltiples plataformas y sitios web.

Para el contexto de este trabajo, se cuenta con acceso a un servidor web el cuál implementa un proxy inverso en NGINX el cuál recibe y redirige todo el tráfico que llega al servidor. A la vez llena una serie de archivos de registro, tanto para tráfico correctamente manejado (access.log) como tráfico que no se puede manejar (error.log), para este estudio se consideró únicamente los archivos access.log ya que representan una superficie más amplia para encontrar vulnerabilidades debido a que en el tráfico correctamente manejado es donde existe un intercambio de información y el servidor envía respuestas.

En términos generales durante este trabajo se realizó el desarrollo de los siguientes procesos:

1. Estructurar datos de texto semi-estructurados obtenidos de access.log

2. Entrenar y aplicar un modelo de regresión logística para clasificar el dominio de tráfico desconocido.
3. Entrenar y aplicar un modelo de bosque de aislamiento para detectar anomalías en el tráfico de un dominio específico.
4. Aplicar herramientas de visualización para estudiar los resultados.

Este proceso así como sus distintos flujos se pueden apreciar con máyor detalle en la Fig. 1.

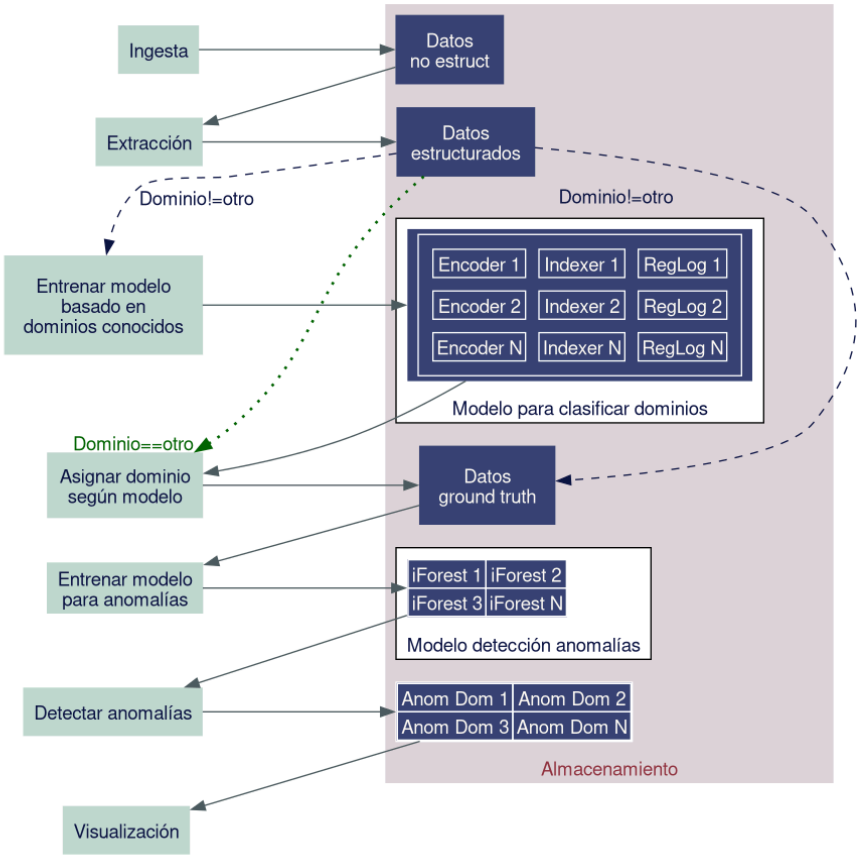

Figura 1. Flujo de procesamiento para el proceso de entrenamiento del sistema.

Las tecnologías empleadas para el desarrollo del sistema fue el protocolo SSH para la ingesta de datos, Apache Spark (en los lenguajes Python y Scala) para el procesamiento de datos a gran escala, el almacenamiento en MinIO bajo la arquitectura de almacenamiento basado en objetos, la visualización se realizó en

Kibana alimentada de datos almacenados en ElasticSearch para permitir búsquedas y consultas de alto rendimiento, la implementación tecnológica se puede apreciar en la Fig. 2.

II.1 Extracción

Cabe destacar que en la arquitectura estudiada se procesan múltiples dominios (sitios) desde un único servidor (Fig. 3), nótese que NGINX se emplea a manera de reverse proxy pero dentro del mismo servidor, esta arquitectura permite maximizar el uso de recursos del servidor, minimizando costos de infraestructura.

Una consecuencia importante de que nginx esté dentro del servidor significa que los archivos de registro contienen tráfico de todas las aplicaciones que el servidor aloja, usando la nomenclatura de la Fig. 3, los archivos de registro del servidor 1 contiene tráfico del dominio 1 (a.com), dominio 2 (b.com) y dominio 3 (c.com) combinado siguiendo como se muestra en el Código 1.

```
'$remote_addr - $remote_user - [$date_time] ''$request" $status
$body_bytes_sent "$http_referer" "$user_agent" "$gzip_ratio"
```

Código 1. Formato del contenido en archivo access.log generado por NGINX.

Fuente: NGINX, 2024

Si analizamos el fragmento del Código 2 es posible notar que existen el primer registro pertenece al dominio a.com, el segundo registro pertenece al dominio b.com en cambio el último no especifican a cuál dominio pertenece, esto se debe a que el campo en cuestión es http_referer, el cuál usualmente lo llena el navegador indicando de cuál url se generó la nueva petición, aterrizando la frase en acciones cotidianas, supongamos que se está navegando el sitio "a.com/inicio" y tiene un botón que nos lleva a "a.com/perfil", cuando el

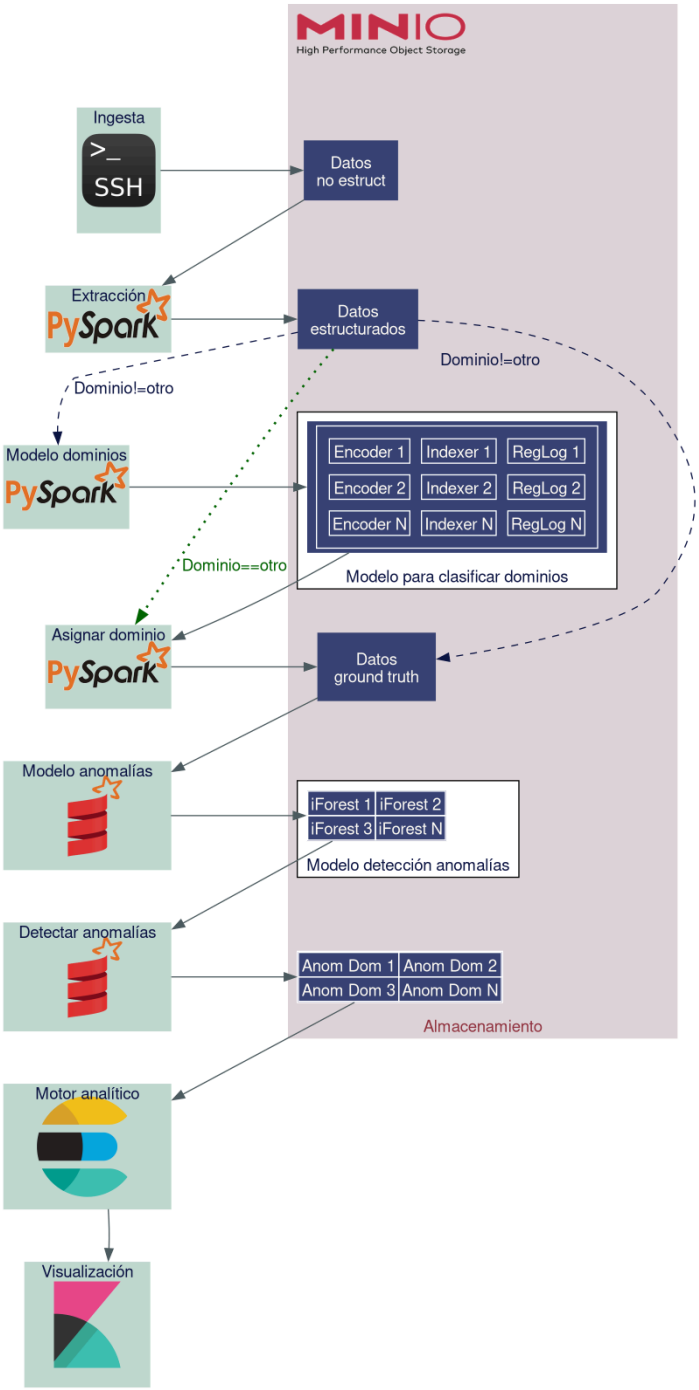

Figura 2. Implementación tecnológica del sistema desarrollado.

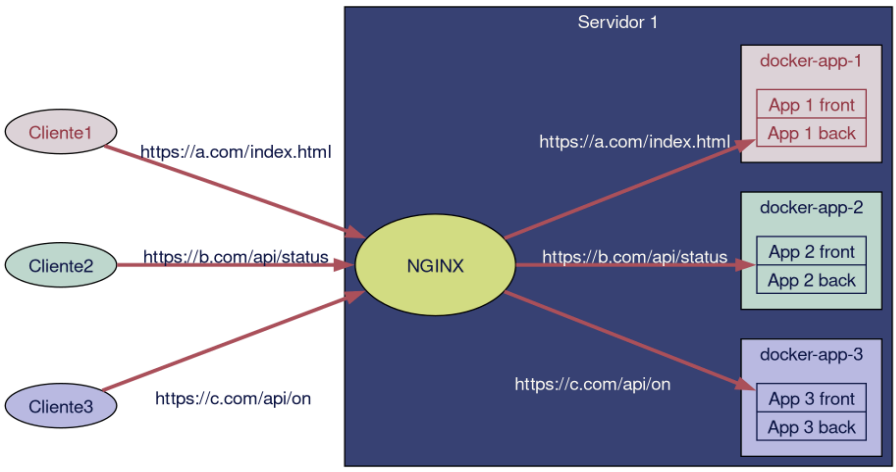

Figura 3. Diagrama de componentes en servidor.

navegador solicite "a.com/perfil" va a llenar automáticamente http_referer con el la url de la cuál venimos, en este caso "a.com/inicio", en otras palabras "a.com/inicio" nos refirió a "a.com/perfil". Pero http_referer puede no llenarse, como es el caso del registro tres del Código 2, esto sucede por ejemplo si en la barra de navegación el usuario escribe "a.com/perfil" nadie nos refiere, o si la petición viene de un dispositivo que tiene la url fija (por ejemplo un reloj checador, o un bot), durante este trabajo usaré la terminología de dominio "otro" para referirme a todo el tráfico que no tenga un http_referer explícito o válido.

```
45.166.93.223 - - [23/Aug/2024:00:00:20 +0000] "GET
/api/manual/find/?category=De%20todo%20un%20poco&searchIn=category&page=1&limit
=12&search=%7B%22searchAllStatuses%22%3Atrue%2C%22searchParam%22%3A%22De%20todo
%20un%20poco%22%7D HTTP/1.1" 304 0 "https://a.com/manual/" "Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/128.0.0.0 Safari/537.36"

201.141.19.215 - - [23/Aug/2024:00:01:05 +0000] "POST
/api/v1/courses/11083/quizzes/331373/submissions/94734/events HTTP/1.1" 204 0
"https://b.com/courses/11083/quizzes/331373/take" "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0
Safari/537.36"

177.225.161.41 - - [23/Aug/2024:00:01:10 +0000] "GET /profile HTTP/1.1" 302 105
"-" "Mozilla/5.0 (iPhone; CPU iPhone OS 17_5_1 like Mac OS X)
AppleWebKit/605.1.15 (KHTML, like Gecko) Version/17.5 Mobile/15E148
Safari/604.1"
```

Código 2. Fragmento enmascarado de archivo access.log generado por NGINX.

A los registros ejemplificados en el Código 2 se les aplicaron las transformaciones del Código 3 para estructurarlo, dando como resultado las variables de la Tabla 1.

```
def decode_uri(uri):
	return urllib.parse.unquote(uri) if uri is not None else None
def get_path(uri):
	return urllib.parse.urlparse(uri).path if uri is not None else None
def get_query_list(uri):
	if uri is None:
		return []
	query = urllib.parse.urlparse(uri).query
	return [f"{k}={v}" for k, v in urllib.parse.parse_qsl(query)]
def get_domain(referer):
	netloc = urllib.parse.urlparse(referer).netloc
	return netloc if netloc not in (None, "", "-") else "Unknown"
def parse_date(date_str):
	return (
		datetime.strptime(date_str, "%d/%b/%Y:%H:%M:%S +0000")
		if date_str is not None
		else None
	)
def to_unix_timestamp(dt):
	return time.mktime(dt.timetuple()) if dt is not None else None


domains_str = os.getenv("DOMAINS")
registered_domains = ast.literal_eval(domains_str)
df = df.withColumn(
	"domain_category",
when(col("domain").isin(*registered_domains),col("domain")).otherwise("otro"),
)
```

Código 3. Transformaciones para estructurar los datos.

Variable	Valor
remote_addr	45.166.93.223
remote_usr	(Vacío)
date_time	23/Aug/2024:00:00:20 +0000
method	GET
req_uri	/api/manual/find/?category=De%20todo%20un%20poco&searchIn=category&page=1&limit=12&search=%7B%22searchAllStatuses%22%3Atrue%2C%22searchParam%22%3A%22De%20todo%20un%20poco%22%7D
http_ver	HTTP/1.1
status	304
body_bytes_sent	0
http_referer	https://a.com/manual/
user_agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.0.0 Safari/537.36
dec_req_uri	/api/manual/find/?category=De todo un poco&searchIn=category&page=1&limit=12&search={"searchAllStatuses":true,"searchParam":"De todo un poco"}
clean_path	/api/manual/find/
clean_query_list	[“category=De todo un poco”, “searchIn=category”, “page=1”, “limit=12”, `search={"searchAllStatuses":true,"searchParam":"De todo un poco"}`]
domain	a.com
fdate_time	23/Aug/2024:00:00:20
dateunixtimest	1724392820
fabstime	0.0
day_of_week	4
domain_category	otro

Tabla 1. Datos estructurados para cada registro de NGINX.

II.2 Clasificación de dominio

Una vez que la información ha sido estructurada se puede notar en la transformación `domain_category` del Código 3 y Tabla 1 que se catalogó como "otro", esto como se mencionó anteriormente, se debe a que el dominio en el campo "http_referer" no forma parte de los dominios alojados en el servidor, sin embargo, por el hecho de estar en access.log se concluye que es tráfico correctamente manejado, es decir que a pesar de que su http_referer no corresponde a los dominios de interés, la petición misma es parte del tráfico de alguno de los dominios de interés.

Para disminuir la cantidad de tráfico catalogado como dominio "otro", se optó por desarrollar una solución que aplica técnicas de aprendizaje automático y que permite, dado el resto de contexto que se tiene de esa conexión en específico, (hora de la solicitud, método de la petición, status, cantidad de bytes en la respuesta y recurso solicitado por el cliente) para poder estimar el grado de probabilidad que esa conexión corresponda a cada dominio y poder categorizar el tráfico en uno de los dominios que procesa el servidor, es decir, poder estimar un http_referer cuando este no existe de manera natural.

Se decidió emplear un modelo de regresión logística perteneciente al grupo del aprendizaje supervisado, ya que contiene datos etiquetados (es decir una porción de los datos están clasificados) los cuales se usan para entrenar el modelo, una vez que el modelo abstrae las relaciones entre variables independientes, es capaz de clasificar datos no etiquetados.

Una distinción importante de la regresión logística (a diferencia de la regresión lineal) es que su salida o producto es binaria o dicotómica. (Hosmer, 2013) Dada esta distinción, los resultados de la regresión logística se pueden interpretar como la probabilidad de que un evento pertenezca a cierto grupo, en la Fig. 4 en la gráfica referente a la regresión logística (derecha) se puede dividir los resultados

en dos grupos (rojo y negro), entonces, la regresión logística nos dara para cada punto en el eje X (variable independiente) una probabilidad (valor del eje Y) de pertenecer al grupo negro.

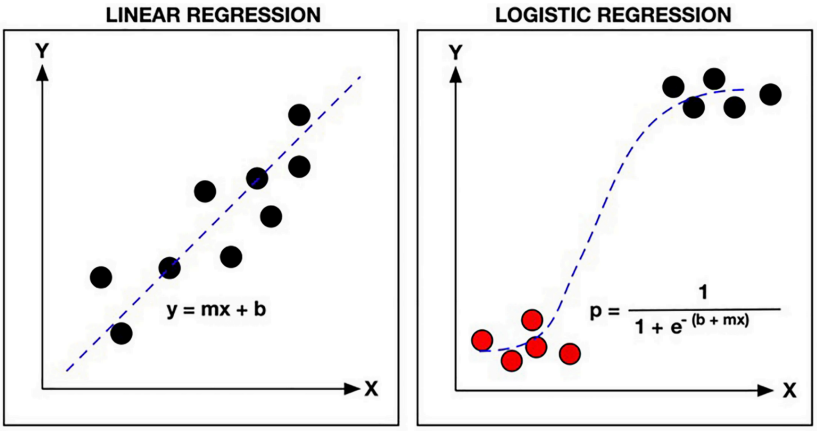

Figura 4. Gráfica y fórmula de la regresión lineal y logística.

Fuente: Rashidi, 2019

De los datos estructurados anteriores (Tabla 1) se consideraron únicamente los campos "fabstime", "day_of_week", "body_bytes_sent", "clean_path", "clean_query_list", "domain_category", "method" y "domain", se eligieron estos campos debido a que se considera son los que mejor representan de manera individual e única cada posible endpoint. Cabe resaltar que para emplear todos los campos en un algoritmo de clasificación es necesario transformar únicamente a valores numéricos.

Dado que los campos "method" y "domain" son categóricos, se aplicó una transformación one-hot (Fig. 5 de referencia) la cuál convierte las categorías en representaciones vectoriales.

En cambio los campos "clean_path" y "clean_query_list" que representan endpoints en formato texto se aplicaron técnicas de procesamiento de lenguaje natural comenzando por tokenizar "clean_path" en 9-gramas como lo muestra el Código 4, transformando un texto como "/api/v1/planner/items" en:

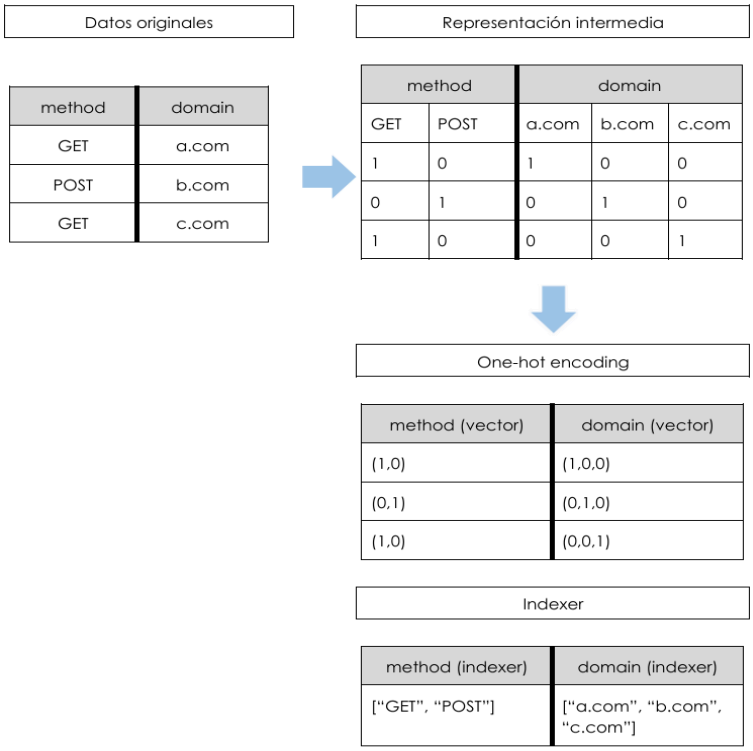

Figura 5. Ejemplo de codificación one-hot e indexer.

```
df_known = df_known.withColumn("path_characters", split(col("clean_path"), ""))

ngram = NGram(n=9, inputCol="path_characters", outputCol="path_ngrams")

df_known = ngram.transform(df_known)

================================================================================

Resultado:
[

/ a p i / v 1 / p,

a p i / v 1 / p l,

p i / v 1 / p l a,

i / v 1 / p l a n,

/ v 1 / p l a n n,

v 1 / p l a n n e,

1 / p l a n n e r,

...

]
```

Código 4. Código y ejemplo para obtener 9-gramas del campo "clean_path".

En cambio para "clean_query_list" el orden de cada dupla llave-valor es irrelevante, tomando como ejemplo el valor "`[end_date=2024-10-12T05:00:00.000Z, order=desc, per_page=1]`" en formato de url `/api/v1/planner/items?end_date=2024-10-12T05:00:00.000Z&order=desc&per_page=1` es exactamente igual a `/api/v1/planner/items?order=desc&end_date=2024-10-12T05:00:00.000Z&per_page=1`, ambas solicitan el mismo recurso con los mismos parámetros. Incluso para "clean_query_list", podemos desechar los valores y mantener únicamente las llaves (end_date, order y per_page) ya que estás son constantes y ofrecen mayor estabilidad en el reconocimiento de patrones para ello se usó el Código 5.

```
transform(col("clean_query_list"), lambda x: split(x, "=")[0])
```

Código 5. Código para obtener únicamente las llaves del campo "clean_query_list".

Finalmente se aplica una transformación HashingTF mediante el cuál se logra eficientar la transformación de ambos campos textuales a valores numéricos que se puedan ingresar al modelo, más uno de los costos es que el método hash es un proceso no reversible, es decir, es imposible regenerar "url_features" basado en el resultado de HashingTF, en este caso, esta pérdida de reversibilidad no es un problema, ya que este proceso solo se usa para entrenar el modelo. (Apache Spark, 2024)

Con los datos agrupados y codificados en un vector, se inicializó el modelo de regresión logarítmica como lo muestra el Código 6 ya que este es un problema de clasificación.

```
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(

      featuresCol="features",  labelCol="domain_index",  family="multinomial",
maxIter=100

)

train_data, test_data = df.randomSplit([0.8, 0.2], seed=123)

lr_model = lr.fit(train_data)
```

Código 6. Creación y entrenamiento de modelo de regresión logística.

Finalmente en la implementación del modelo de regresión logístico (Código 6) se definió de tipo "multinomial" ya que no solo se cuenta con dos categorías, sino con 48 categorías (cada posible dominio).

Una vez que se tiene el modelo para asignar dominios entrenado, se realizan las mismas transformaciones y preparación de datos que en el entrenamiento en los datos hasta ahora catalogados como dominio "otro". El modelo regresa un vector de dimensión igual a las posibles categorías (en este contexto los posibles dominios) similar mostrado en el Código 7:

```
Resultado:

[

2.31E-10,    4.60E-13,     8.00E-11,    2.04E-10,   6.57E-10,     5.05E-09,

6.26E-11,    2.34E-10,     1.67E-10,    2.51E-09,   1.86E-10,     9.69E-10,

0.9999999887,

1.30E-10,    6.79E-11,     7.32E-11,    1.02E-10,    3.77E-11,    1.89E-10,

1.14E-10,    6.71E-11,     3.55E-11,    2.79E-11,    3.41E-11,     5.39E-11,

1.87E-11,    1.27E-11,     5.05E-12,    3.19E-12,    4.24E-12,     2.48E-12,

2.78E-12,    1.49E-12,     1.14E-12,    1.15E-12,    1.12E-12,     1.22E-12,

8.74E-13,    9.74E-13,     8.13E-13,    8.42E-13,    8.75E-13,     6.50E-13,

7.17E-13,    4.72E-13,     6.42E-13,    4.88E-13,    1.69E-13

]
```

Código 7. Resultado de aplicar el modelo de regresión logística entrenado.

Este vector representa la probabilidad que tiene un ejemplo de pertenecer a cada uno de los dominios, en su mayoría la probabilidad es baja, excepto en uno, en el cuál el modelo calcula que tiene 99% de probabilidad de pertenecer, para asignar el nombre del dominio se recurre al indexador (véase Fig. 5 para referencia).

Si bien en este ejemplo el modelo cataloga fuertemente hacia un dominio, pueden existir casos menos definidos, por ejemplo un vector donde ningún dominio supere el 50%, para prevenir falsos positivos, se optó por aumentar la barrera de categorización de dominio, es decir, el modelo debe indicar más del 70% de probabilidad de otra manera se mantiene categorizado como "otro"

II.3 Detección de anomalías

Para la detección de anomalías se decidió emplear el modelo de bosque de aislamiento, el cuál se ajusta a contextos donde se requiere una rápida ejecución y detección. Su velocidad es en parte gracias a que a diferencia de la mayoría de los modelos de detección de anomalías que construyen un perfil de las instancias normales y después, identifican como anomalías a aquellas instancias que no encajan en dicho perfil, el bosque de aislamiento realiza una separación de una instancia del resto de las instancias. Tomando como partida que las anomalías son 'pocas y diferentes' y por lo tanto, más susceptibles a ser aisladas (Liu, 2008)

La construcción "árbol de aislamiento" corresponde a un árbol binario que representa una secuencia de divisiones anidadas en un espacio de características, donde el nodo raíz corresponde al espacio completo (los datos en su totalidad) y cada nodo hijo corresponde a un subconjunto de los datos totales al cuál se le ha aplicado "n" divisiones (Fig. 6), siendo "n" la profundidad del nodo. (Staerman, 2019)

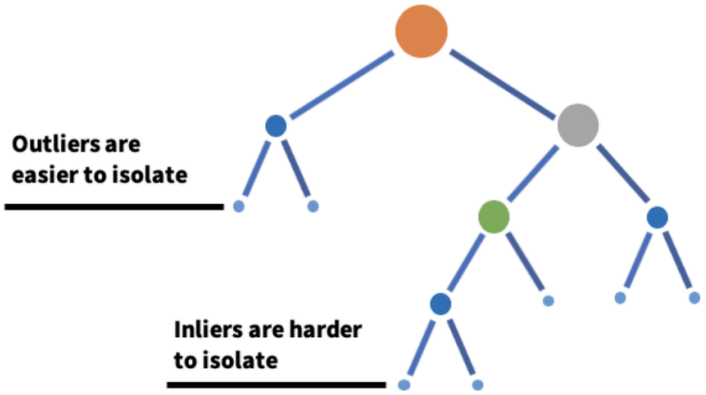

Figura 6. Ejemplo de árbol de aislamiento.

Fuente: Linkedin, 2019

Las divisiones en el grupo de datos se realizan de manera aleatoria, y este proceso iterativo de división se realiza hasta que todas las instancias (datos) sean aisladas. Esta división aleatoria produce caminos (ramas) notablemente más cortos en las anomalías (Liu, 2008) véase la comparación de la Fig 7.

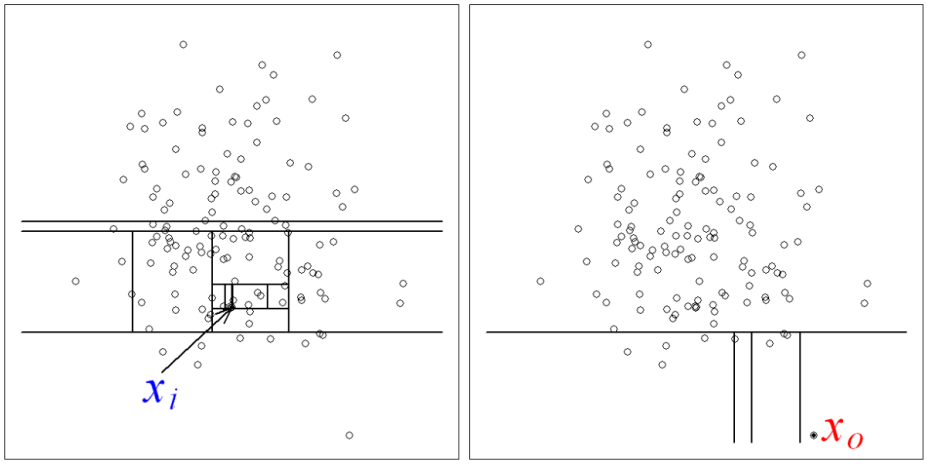

Figura 7. Comparativa de divisiones necesarias para aislar datos normales (izquierda) y datos anómalos (derecha).

Fuente: Liu, 2008

Cabe destacar la escalabilidad y performance del modelo ya que al no requerir medición de distancia o densidad para detectar anomalías, se eliminan costos computacionales, además de tener una complejidad lineal de tiempo baja así como bajos requisitos de memoria. (Liu, 2008)

Se consideró pertinente generar un modelo para cada dominio de esta manera se puede parametrizar y ajustar el modelo a las características específicas, permitiendo cada dominio generar su propia representación de anomalía sin ser influenciada por el comportamiento de otros dominios.

III. MEMORIA DE CÁLCULO Y RESULTADOS

Durante el proceso de extracción se tomaron datos textuales "crudos" no estructurados correspondientes a un periodo de 72 días consecutivos y se convirtieron en datos estructurados, ampliando de 9 variables iniciales (véase Tabla 9 para referencia) a 21 variables (véase Código 1 para referencia), sin embargo a pesar de este enriquecimiento de datos con metadatos se logró una reducción del almacenamiento de 2.0 Gb iniciales a 597 Mb en el paso de extracción, esto supone una reducción del 70% en almacenamiento. Esta característica se considera un éxito para el objetivo de crear un sistema escalable ya que permite flexibilidad para agregar más clientes, dominios y datos históricos al abaratar costos de almacenamiento. En la misma línea se considera la decisión de usar el motor de cómputo Apache Spark que permite el procesamiento de datos en paralelo en clusters de computadoras. (Chambers, 2018)

III.1 Clasificación de dominio

Después de realizar el entrenamiento del clasificador logístico se realizaron diversas pruebas para evaluar su calidad, los resultados se muestran en la Tabla 2.

Prueba	Resultado
Precisión	98.82%
Precisión ponderada	98.91%
Exhaustividad ponderada	98.97%
Puntaje F1	98.82%

Tabla 2. Métricas para evaluar el desempeño del modelo de clasificación de dominios.

Una vez aplicado el modelo de clasificación a los datos con dominio "otro" se logró reducir de 18.9% (1,543,472 registros) a 3.6% (294,075 registros) quedando distribuidos como lo indica la Fig. 8

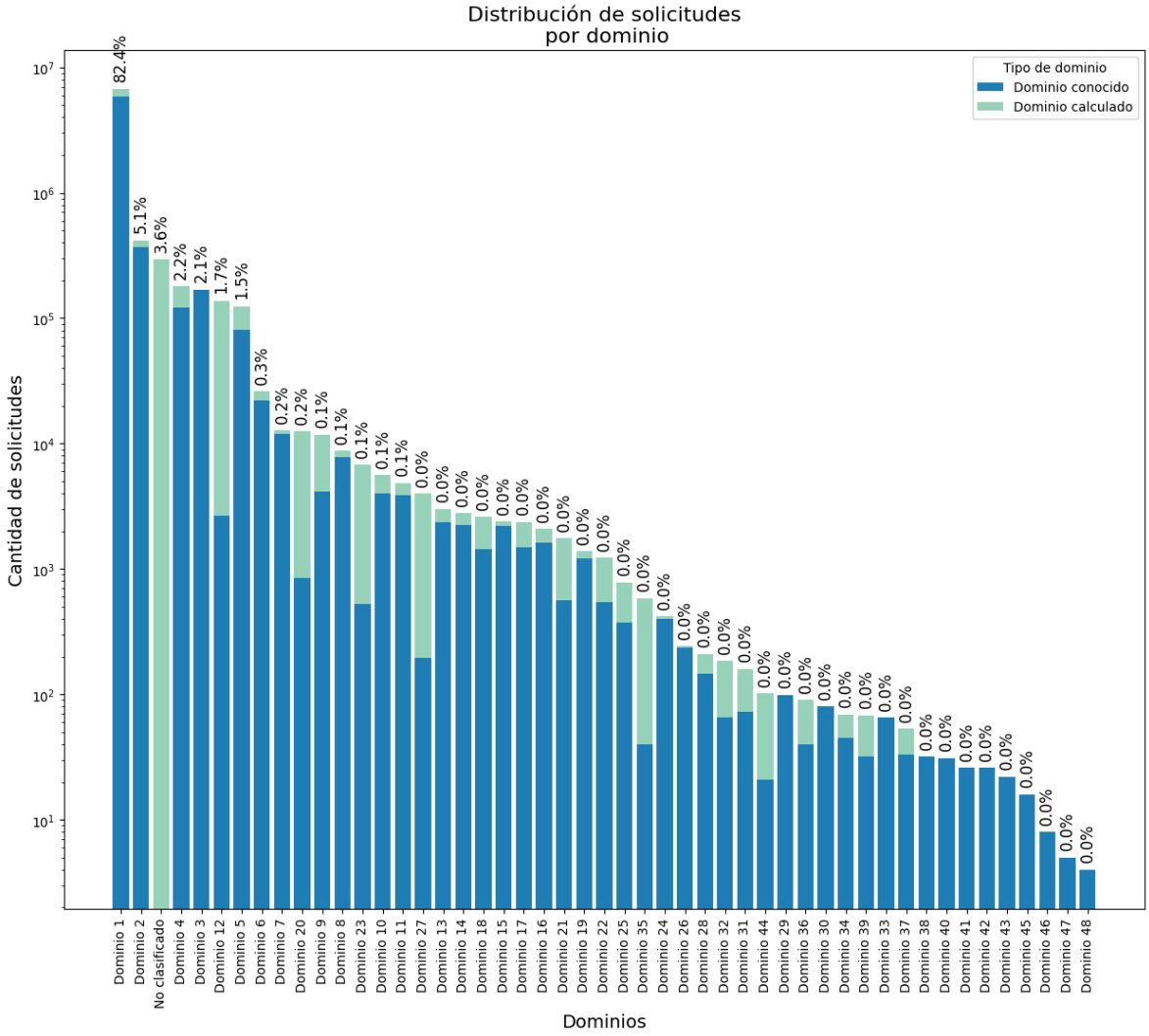

Figura 8. Distribución de registros por dominio después de reclasificar el dominio "otro".

A pesar de que los datos indican que no fue posible categorizar todos los registros de dominio "otro", se debe tener en cuenta que nunca se fijó como objetivo reducir a cero los registros de dominio "otro", ya que esto es virtualmente imposible, como se mencionó en la subsección II.1, hay sistemas o fracciones de

sistemas que por naturaleza nunca van a tener http_referer, un ejemplo es un sistema de relojes checadores, o un sistema de scraping o una api pública, ya que estas peticiones al no ser generadas por humanos o no ser generadas desde un navegador web, pueden prescindir del campo http_referer el cuál se usa en la clasificación inicial del dominio (campo "domain" en la Tabla 1) bajo esta aclaración, se considera que las técnicas empleadas fueron correctas para reducir la cantidad de registros con dominio "otro", donde se prefirió reducir los falsos positivos (haber catalogado un registro en el dominio incorrecto) con tal de aumentar los verdaderos positivos.

III.2 Detección de anomalías

El entrenamiento del bosque de aislamiento presentó un desempeño notable, en el que a pesar de contar con millones de registros y cerca de dos mil características por registro, el tiempo de procesamiento es notable y se presenta en la Tabla 3

Cantidad de registros por dominio	Tiempo de entrenamiento
6, 731, 624 registros	50 minutos
178, 674 registros	2 minutos
169, 464 registros	11 minutos
138, 263 registros	3 minutos

Tabla 3. Métricas para evaluar el desempeño del modelo de clasificación de dominios.

Explorando los datos categorizados como anomalías se seleccionaron los siguientes casos notables en el Código 8.

```
/__debugging_center_utils___.php?log=;echo  ljyabmwesqxkknnjecooewtpopjvuxsk  |
id

/__debugging_center_utils___.php?log=;echo  ljyabmwesqxkknnjecooewtpopjvuxsk  |
ipconfig

/services/auth/config/aws_credentials.json

/plus/recommend.php?action=&aid=1&_FILES[type][tmp_name]=\x5C'  or  mid=@`\x5C'`
/*!50000union*//*!50000select*/1,2,3,md5(871702),5,6,7,8,9#@`\x5C'`+&_FILES[typ
e][name]=1.jpg&_FILES[type][type]=application/octet-stream&_FILES[type][size]=4
294
```

Código 8. Ejemplos de tráfico categorizado como anómalo.

Los primeros dos renglones son similares e intentan invocar el endpoint `__debugging_center_utils___.php` (el cuál no existe) e intenta ejecutar código (payload) vía query list params `echo` para imprimir un mensaje en salida estándar y los comandos `id` y `ipconfig` los cuales intentan obtener información sobre el entorno de ejecución del servidor, esté es un intento de aprovechar la vulnerabilidad CVE-2016-5674 que afecta a equipo de videovigilancia (NVD, 2024). El tercer renglón intenta aprovechar una posible configuración incorrecta del servidor en la que expone datos de autenticación para servicios de Amazon Web Services. El cuarto renglón es un intento de inyección de código sql y sobrepasar un posible web application firewall.

Como se puede observar, los ataques son diversos y emplean técnicas y frameworks o lenguajes distintos.

III.3 Visualización

La manera en que este sistema opera actualmente, procesa y detecta anomalías en las peticiones a nivel unitario, es decir, una petición a la vez, lo cuál permite el análisis detallado de peticiones pero pierde un poco el contexto de porque la petición es anómala, por ello se integraron herramientas de visualización que permiten recontextualizar los datos, para ello se generó un

dashboard de visualización que contiene las gráficas expuestas en las Fig 9, 10 y 11.

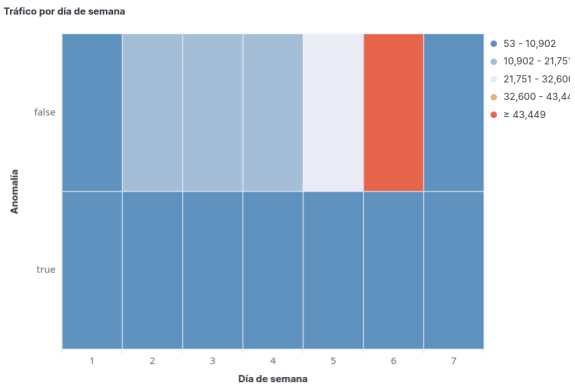

Figura 9. Cantidad de anomalías por dia de semana

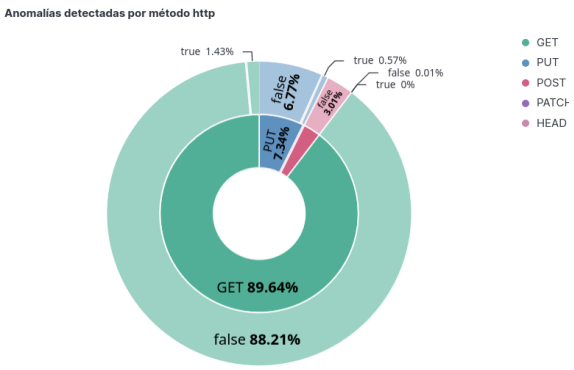

Figura 10. Distribución de métodos y anomalías

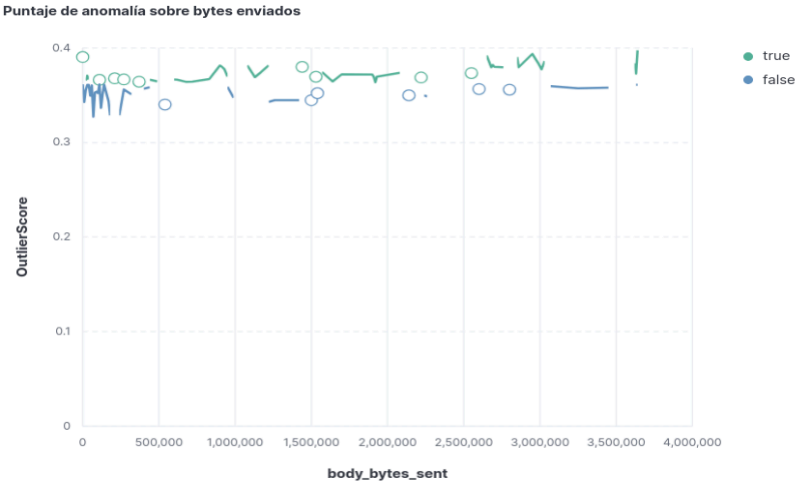

Figura 11. Grado de anormalidad según bytes contestados

IV. CONCLUSIONES Y RECOMENDACIONES

En términos generales se considera que se lograron establecer las bases tecnológicas y algorítmicas para generar un sistema que exitosamente pueda analizar y obtener conocimiento de grandes volúmenes de información, que a pesar de haber numerosos falsos positivos en las anomalías detectadas es necesario recordar el contexto de estos ataques, es una centena o menos de peticiones maliciosas entre miles o millones que procesa el servidor, por ello se

resalta la necesidad y utilidad de aplicar técnicas de aprendizaje automático. Con la arquitectura y configuración actual se consiguió ofrecer una plataforma y código escalable, tolerante a grandes volúmenes de información.

Tarea	Tiempo y volumen de procesamiento
Extracción	35 minutos en procesar y expandir 8,170,910 registros
Entrenamiento de clasificador	3 hora, 6 minutos en entrenar el modelo basado en 6,636,438 registros
Predicción de modelo	1 minuto y medio en aplicar modelo entrenado a 1,543,472 de registros
Generar ground truth	1 minuto en combinar 8,169,584 registros de dominios conocidos con dominios asignados
Detección de anomalías	Aproximadamente 10 minutos por dominio

Tabla 4. Tiempo y volumen de datos procesados.

El avance logrado estableció las bases fundamentales para producir un sistema productivo y comercial debido a sus capacidades técnicas y madurez en prácticas de ingeniería de software.

IV.1 Ambiente de Producción

El trabajo desarrollado se centra en la implementación de modelos y algoritmos para el aprendizaje de patrones, una vez que se completa esta etapa y se tienen los modelos entrenados, se propone el flujo de la Fig. 12 en el que el tráfico es dirigido más eficazmente a los algoritmos de detección para obtener respuestas de anomalía en tiempo real. Además se propone la implementación tecnológica (Fig. 13) que si bien mantiene el núcleo de procesamiento en Apache Spark, aprovecha tecnología de flujos de datos como Apache Kafka para procesamiento paralelo, y Spark Streaming.

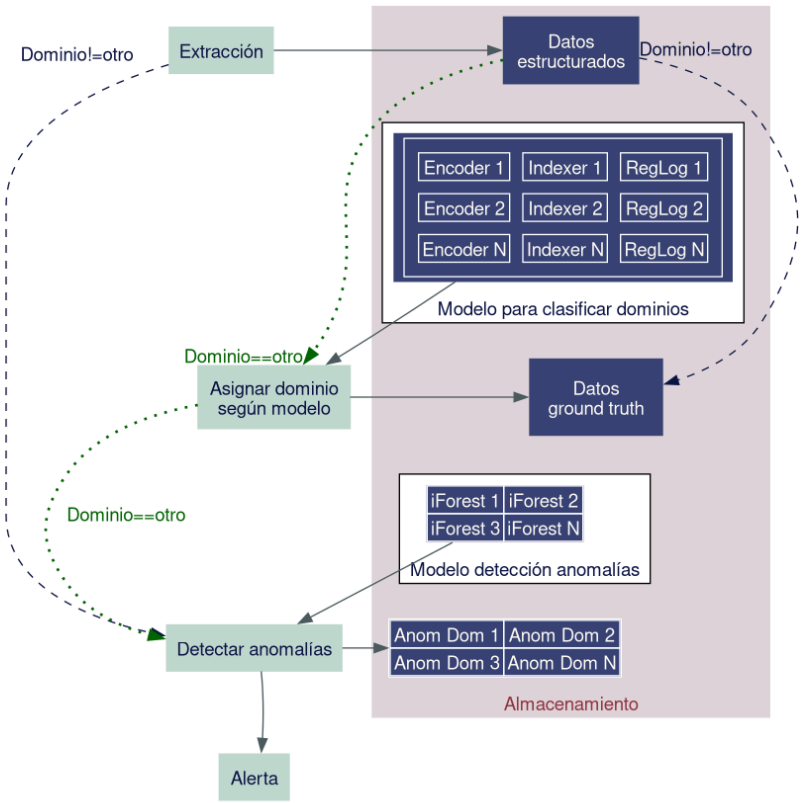

Figura 12. Flujo de procesamiento para el proceso de detección en tiempo real.

Finalmente se hace hincapié en que en las Fig 1, 2, 12 y 13 se representan múltiples clasificadores de dominio (RegLog 1, RegLog2, RegLog n) así como múltiples modelos de detección de anomalías (iForest 1, iForest 2, iForest 3, iForestN) esto se debe a que la arquitectura empleada permite entrenar múltiples modelos (Fig. 14), dando flexibilidad de realizar análisis paralelos sobre el mismo registro, por ejemplo un modelo que esté entrenado con las tendencias del último mes (primeros cuatro renglones de la Fig. 14), un modelo entrenado con las tendencias del último bimestre (quinto y sexto renglón de la Fig. 14), o del último cuatrimestre (último renglón de la Fig. 14). De esta manera se puede analizar un mismo registro en diversos contextos históricos lo que amplía los conceptos de normalidad y anormalidad.

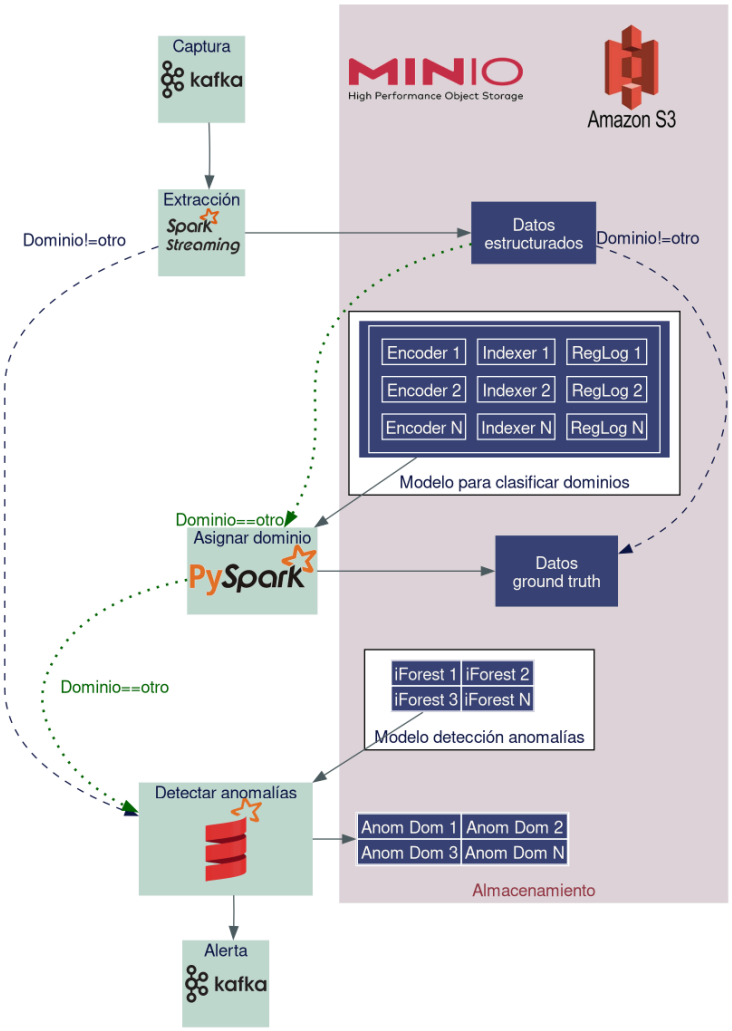

Figura 13. Implementación tecnológica del sistema de detección en tiempo real.

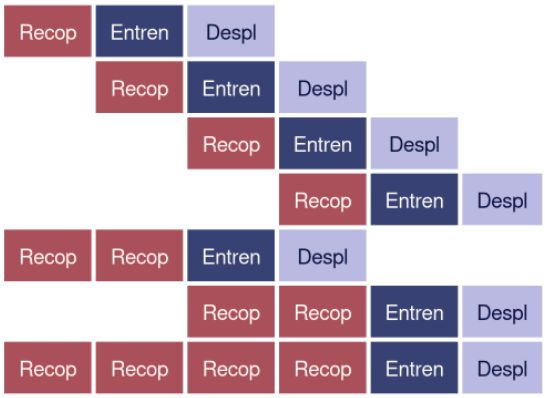

Figura 14. Propuesta de flujo de entrenamiento paralelizado

IV.2 Trabajos futuros

Finalmente se reconocen áreas de oportunidad en el procesamiento necesario para detectar anomalías, a pesar de que el servidor maneja 48 dominios distintos, no significa que sean 48 proyectos distintos, es común ofrecer el mismo software a múltiples clientes, aplicar técnicas de clusterización puede ayudar a expandir el ground truth de los datos. Esto puede ayudar a capturar el comportamiento normal de un proyecto repartido entre múltiples clientes.

Otro punto de mejora es la implementación de un algoritmo Minhash mediante el cuál posibilite agrupar endpoints similares, por ejemplo, supongamos que tenemos una ruta `/api/estudiante/1` donde 1 representa el id del primer estudiante, el estudiante 200 entonces solicitaría `/api/estudiante/200` a pesar de que son rutas distintas, ambas tienen muchos elementos en común porque en realidad son la misma ruta con un parámetro variable, el algoritmo Minhash permite detectar estas similitudes y realizar una clusterización a nivel ruta, esto disminuye la cantidad de información que se debe procesar y puede mejorar la calidad y capacidad de detección de anomalías.

Un tercer y último punto de mejora detectado es la implementación de análisis de tipo series de tiempo, actualmente como se mencionó anteriormente, el análisis se realiza a nivel unitario donde cada registro se compara con un grupo, pero este análisis deja de lado tendencias contextuales, por ejemplo fluctuaciones en el tráfico por temporadas, o fluctuaciones por semana, o un incremento paulatino pero constante, la implementación de múltiples modelos de la Fig. 14 pretende cubrir esta omisión, pero en general se considera que aplicar un tratamiento de serie de tiempo puede ofrecer ventajas notables aunado a la arquitectura de múltiples modelos propuesta.

V. REFERENCIAS

Apache Spark (2024), HashingTF, https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.HashingTF.html

BID, (2020) Ciberseguridad Riesgos, avances y el camino a seguir en América Latina y el Caribe, Reporte Ciberseguridad https://publications.iadb.org/es/publications/spanish/viewer/Reporte-Ciberseguridad-2020-riesgos-avances-y-el-camino-a-seguir-en-America-Latina-y-el-Caribe.pdf

Chambers et al, (2018), Spark: The Definitive Guide, Big Data Processing Made Simple, (1st Ed), O'Reilly Media Inc.

Cisco, (2017), Cisco Visual Networking Index: Forecast and Trends, 2017–2022. https://futuretimeline.net/data-trends/pdfs/cisco-2017-2022.pdf

Dominguez, (2023), Políticas de información en ciberseguridad en México: atención y tratamiento a conductas disvalorativas en red, Políticas de información: de lo instrumental a lo informacional. Instituto de Investigaciones Bibliotecológicas y de la Información / unam https://ru.iibi.unam.mx/jspui/bitstream/IIBI_UNAM/805/1/01_politicas_informacion_rosa_dominguez%20.pdf

Hosmer et al, (2013). Applied logistic regression. John Wiley & Sons. 2nd Edition, https://www.researchgate.net/profile/Andrew-Cucchiara/publication/261659875_Applied_Logistic_Regression/links/542c7eff0cf277d58e8c811e/Applied-Logistic-Regression.pdf

Jieming Zhu et al, (2023), Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics, IEEE 34th International Symposium on Software Reliability Engineering (ISSRE) https://ieeexplore.ieee.org/abstract/document/10301257

Linkedin, (2019) Detecting and preventing abuse on LinkedIn using isolation forests, Engineering Blog, Data Management, https://www.linkedin.com/blog/engineering/data-management/isolation-forest

Liu et al, (2008). Isolation forest. In 2008 eighth ieee international conference on data mining (pp. 413-422). IEEE. https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf?q=isolation-forest

Nginx, (2024), Configuring Logging, https://docs.nginx.com/nginx/admin-guide/monitoring/logging/

NVD, (2024), CVE-2016-5674 Detail https://nvd.nist.gov/vuln/detail/CVE-2016-5674

Rashidi et al, (2019) Artificial Intelligence and Machine Learning in Pathology: The Present Landscape of Supervised Methods. Academic Pathology. 2019;6. doi:10.1177/2374289519873088 https://journals.sagepub.com/doi/full/10.1177/2374289519873088

Staerman, et al, (2019). Functional isolation forest. In Asian Conference on Machine Learning (pp. 332-347). PMLR. https://proceedings.mlr.press/v101/staerman19a/staerman19a.pdf