Detección de Inyecciones SQL mediante técnicas de procesamiento de lenguaje natural y aprendizaje automático

Trabajo Final - Bases de Datos Masivas
Universidad Nacional de Luján

Gonzalo Benito
Legajo: 182885

Diciembre 2025

Resumen

Este trabajo propone desarrollar un *pipeline* de detección de ataques de Inyección SQL (SQLi) utilizando técnicas de procesamiento de lenguaje natural (NLP) y algoritmos de aprendizaje automático (ML). Se utiliza un dataset público obtenido de **Kaggle** [1] que contiene sentencias SQL etiquetadas como peligrosas o benignas. La metodología se centra en comparar dos formas de convertir el texto de las sentencias en características: (1) una representación clásica basada en n-gramas de caracteres (TF, *Term Frequency*) y (2) vectores densos (*sentence embeddings*) producidos por un modelo de lenguaje público obtenido de **Hugging Face**. Con ambas representaciones se prueban técnicas de reducción de dimensionalidad y clasificadores tradicionales, en conjunto con la aplicación de `SMOTE` para sobremuestreo del conjunto de entrenamiento. El objetivo es determinar qué combinaciones de modelos y transformaciones separan mejor las clases mediante la evaluación y comparación del desempeño, y demostrar si es posible la detección de ataques de inyección SQL mediante técnicas de ML.

1. Introducción

Las inyecciones SQL representan una de las vulnerabilidades más críticas en el desarrollo de aplicaciones, sea en entornos web, mobile o desktop, permitiendo a atacantes manipular consultas a bases de datos y comprometer la confidencialidad, disponibilidad e integridad de la información. La detección automatizada de estos ataques es fundamental para la seguridad de los sistemas de información más allá de las buenas prácticas de desarrollo.

Se pueden encontrar varios trabajos que intentan abordar este tipo de problemas mediante la aplicación de técnicas de NLP y ML, cada uno con sus diferencias en las distintas fases del *pipeline*. Este estudio se centra en construir un sistema de detección de SQLi con el uso de estas técnicas. Es importante notar que, en la detección de SQLi, el preprocesamiento debe realizarse correctamente y con cuidado, ya que los caracteres especiales, puntuaciones y dígitos son elementos clave que caracterizan los ataques y no deben eliminarse.

Se siguen dos estrategias principales para detectar la clase de las sentencias del dataset, ambas basadas en representaciones vectoriales:

Estrategia 1: Vectores "ralos" (TF-IDF). Convierte el texto mediante la generación de características con el algoritmo `TfidfVectorizer` incluido en la librería `scikit-learn` [2]. Este genera características a partir de los n-gramas que se pueden extraer de las sentencias. En este caso, las características fueron generadas a partir de la tokenización en n-gramas de

tres caracteres. La ventaja de esta estrategia es que captura patrones sintácticos y símbolos típicos de ataques de SQLi (comillas, comentarios, palabras reservadas del lenguaje SQL y del motor de base de datos atacado), y además es fácil de interpretar. La desventaja principal es que la cantidad de características generadas puede crecer muy rápidamente debido a la variedad de n-gramas, lo que deriva en una mayor dimensionalidad y coste computacional.

Estrategia 2: Vectores densos (*Embeddings*). Representa las sentencias utilizando el modelo de lenguaje (LM) `all-MiniLM-L6-v2` [3, 4], basado en la arquitectura *Transformer*, para la transformación de sentencias a vectores de 384 componentes. Estos intentan resumir información semántica y capturar patrones más amplios o generales en vectores de dimensión fija. La principal ventaja de este enfoque es que la matriz resultante tendrá una dimensión fija determinada por el modelo de lenguaje, condensando los patrones generales encontrados. Además, la cantidad de recursos necesarios para almacenar y procesar esta matriz será menor a la matriz rala del enfoque anterior. Se espera que los modelos ML puedan procesarla con mayor velocidad y seguir obteniendo buenos resultados.

Ambas representaciones permiten luego usar técnicas de reducción dimensional. En el primer enfoque se emplea **SVD** para la matriz rala. Debido a la naturaleza no lineal del problema, no se esperan buenos resultados, ya que este método busca principalmente patrones lineales en los datos. Aún así, al ser un método recomendado para matrices ralas, se utiliza para intentar reducir considerablemente el número de dimensiones y simplificar la entrada para los algoritmos de clasificación.

Por otro lado, para la matriz densa (de dimensionalidad fijada por el LM y considerablemente menor a la anterior), se aplican **PCA** y **t-SNE**. El primero también se centra en capturar la varianza de patrones lineales, por lo que requirió un número elevado de componentes para explicar el comportamiento de los datos; mientras que t-SNE, en conjunto con PCA (aplicado previamente por recomendación de la librería), logró representar de manera más clara la separación no lineal entre sentencias benignas y maliciosas [5].

Varios trabajos recientes aplican estos enfoques y reportan buenos resultados, aunque la comparación depende mucho del dataset usado y de las técnicas de ingeniería de características aplicadas. Esta transformación de las sentencias es necesaria para la aplicación de algoritmos de clasificación como *Support Vector Machines* (SVM), *Random Forest* y *Regresión Logística*, que se usan en los últimos procesos del *pipeline*.

Después de la etapa de ingeniería de características y de la separación en conjuntos de entrenamiento y prueba, se aplica la técnica `SMOTE` [6] de sobremuestreo sobre el primero para igualar a la clase mayoritaria, que en este caso es la clase benigna.

Por último, se evalúa el rendimiento de los algoritmos de clasificación mencionados para identificar patrones maliciosos mediante métricas de evaluación estándar como *accuracy*, *precision* o *F1-Score*, y métodos gráficos como matrices de confusión y curvas ROC.

1.1. Trabajos relacionados

Existen investigaciones previas en la detección de payloads maliciosos utilizados en variedad de ataques (ataques web en su mayoría), como XSS (Cross-site-scripting) o SQLi, usando técnicas de aprendizaje automático. Esto se explica en gran parte porque la vulnerabilidad SQLi ha sido catalogada consistentemente por la **Open Web Application Security Project Foundation** [7] como una de las amenazas más graves para las aplicaciones web. Los enfoques tradicionales basados en reglas estáticas demostraron ser insuficientes frente a la evolución constante de las técnicas de ataque, y en consecuencia surgieron estas nuevas líneas de investigación que buscan encontrar una solución, en el campo del NLP y ML.

Rakha Satria Pratama, Muhamad Irsan, Rio Guntur Utomo [8] llevaron a cabo una comparación de modelos de ML (Support Vector Machine (SVM), K-Nearest Neighbor (KNN),

and Logistic Regression (LR)) utilizando un dataset de Kaggle, destacando la superioridad del modelo Support Vector Machine (SVM). De manera similar, **Alkhathami & Alzahrani** [9] encontraron que SVM era el algoritmo más eficaz en la detección de SQLi.

La conversión de las consultas SQL en características numéricas es un paso crítico en el pipeline de detección de SQLi. El presente trabajo se centra en comparar representaciones basadas en frecuencia (n-gramas de caracteres/TF) con representaciones densas (vectores sentence embeddings).

El uso de la frecuencia de términos o n-gramas para la representación de las consultas es una técnica estándar (bag of words, BOW). **Alkhathami & Alzahrani**, por ejemplo, emplearon la clase CountVectorizer para esta tarea. De manera similar, **Venkatramulu et al.** [10] evaluaron tanto CountVectorizer (unigramas) como TFIDF Vectorizer de la librería scikit-learn y reportaron buenos resultados con XGBoost.

En contraste con las representaciones ralas basadas en frecuencia, los vectores densos (embeddings) capturan mejor la semántica y el contexto de las consultas. Existen también varias investigaciones sobre estas representaciones aplicadas en la detección de inyecciones SQL. Por ejemplo, podemos mencionar el trabajo de **Ding Chen et al.** [11], quienes propusieron utilizar embeddings (Word2Vec) y redes MLP/CNN para la detección de SQLi.

1.2. Objetivos

El objetivo general de este trabajo es desarrollar un pipeline capaz de detectar inyecciones SQL (SQLi) utilizando técnicas de procesamiento de lenguaje natural (NLP) y aprendizaje automático (ML), comparando distintos enfoques de representación y clasificación.

Para lograr este objetivo, se definieron los siguientes objetivos específicos:

- Preparar el dataset de sentencias SQL etiquetadas, realizando las transformaciones necesarias para garantizar la consistencia de las etiquetas.
- Generar diferentes representaciones vectoriales de las sentencias, utilizando tanto métodos basados en frecuencia (TF-IDF con n-gramas) como embeddings.
- Entrenar y ajustar modelos de clasificación de Machine Learning para comparar su desempeño.
- Evaluar los resultados con métricas de rendimiento.

2. Dataset

El corpus de datos empleado en esta investigación es el "SQL Injection Dataset", recopilado por Syed Saqlain Hussain en la plataforma Kaggle [1]. Este conjunto consta de 30,873 instancias, cada una caracterizada por dos atributos principales: Sentence (la consulta SQL o texto asociado) y Label (indicador de inyección SQL, donde 0 representa una consulta benigna y 1 una consulta maliciosa).

Atributo	Tipo	Descripción
Sentence	texto	Consulta SQL completa o texto
Label	categórica	0 = no inyección; 1 = inyección SQL

Se identificó un desbalance entre las clases en el dataset original , que será abordado en la fase de Ingeniería de Características, específicamente después de la vectorización de las sentencias.

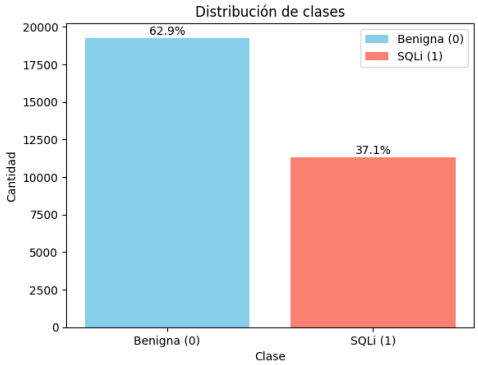

Fig. Distribución de Clases

3. Metodología

La metodología se estructuró en un pipeline de procesamiento secuencial, abarcando las etapas del proceso de KDD más comunes, como lo son el preprocesamiento, ingeniería de características, balanceo de clases, reducción de dimensionalidad, modelado y evaluación.

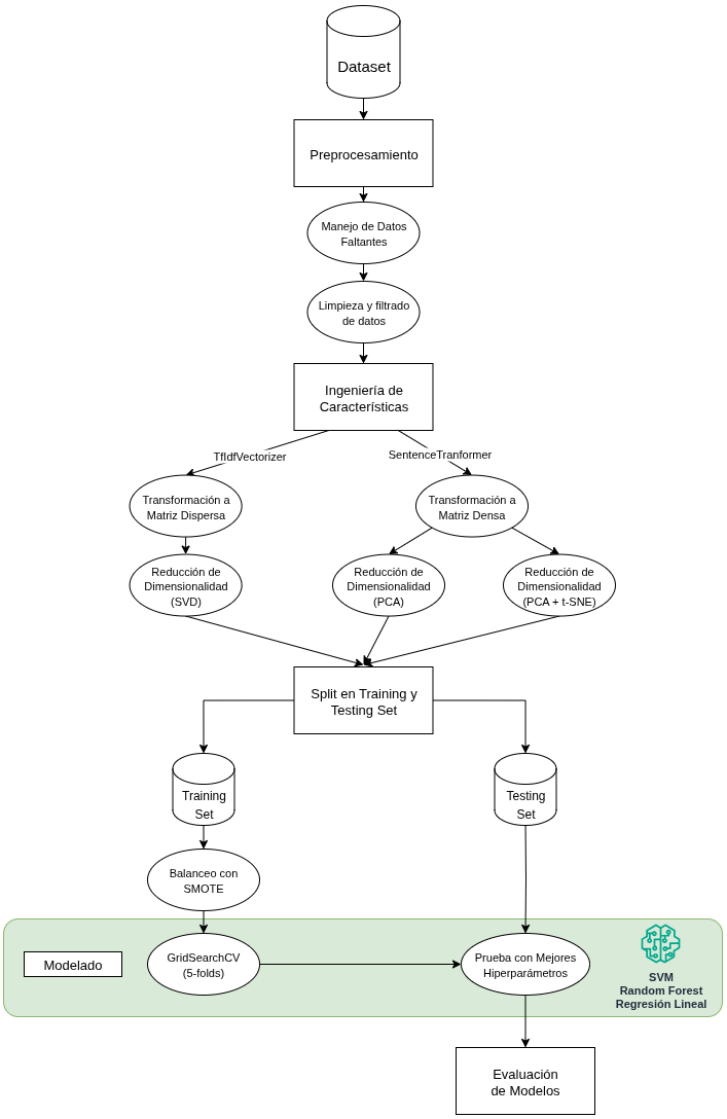

Fig. Diagrama del Pipeline de Machine Learning

3.1. Preprocesamiento

Las tareas de preprocesamiento realizadas incluyeron:

- **Eliminación de valores faltantes**: Se verificó y se procedió a la eliminación de registros con valores ausentes en la columna *Sentence*.
- **Filtrado de Clases**: Se mantuvieron únicamente las instancias correspondientes a las clases 0 (benigna) y 1 (SQLi) para el enfoque binario de la detección. Esto fue necesario debido a la presencia de valores distintos a 0 o 1, entre ellos strings pertenecientes a las sentencias.
- **Eliminación de Outliers**: Se identificaron y eliminaron outliers en la longitud de las sentencias. Específicamente, se detectó una instancia (registro 19341) con una longitud de 5370 caracteres, considerada un valor atípico y eliminada del conjunto de datos.

La distribución de las longitudes de las sentencias, mostró un comportamiento fuertemente asimétrico, caracterizado por una alta concentración de sentencias cortas y una pequeña proporción de sentencias considerablemente más largas. Este tipo de distribución presenta una cola larga, común en datasets de texto, aunque no se realizó un ajuste formal para verificar si sigue una Ley de Potencia.

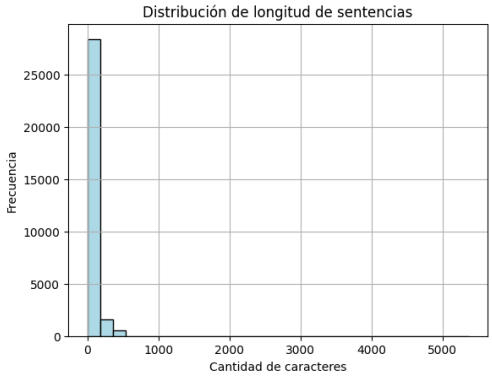

Fig. Distribución de longitudes de sentencias original

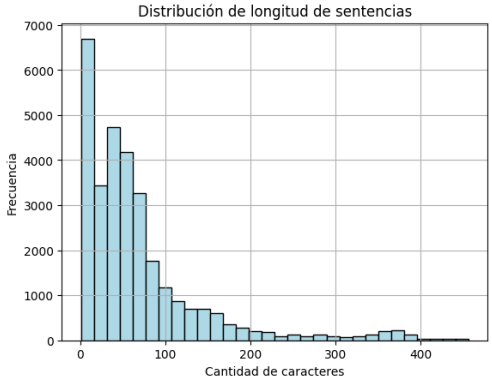

Fig. Distribución de longitudes de sentencias luego de limpieza

3.2. Ingeniería de Características

Esta fase es crucial, ya que la representación de los datos impacta directamente en el rendimiento de los modelos. El pipeline se bifurcó para explorar distintas técnicas de generación de características:

Representación TF-IDF con N-gramas de Caracteres: Se utilizó la clase `TfidfVectorizer` de scikit-learn [2] para generar una matriz TF-IDF. Se exploraron distintas configuraciones, incluyendo `ngram_range=(3,3)` (indica el rango de caracteres de los n-gramas, que en este caso son solo trigramas) y `analyzer='char_wb'` (para ignorar los espacios), junto con `min_df=3` (frecuencia mínima de un trigrama en todo el dataset) y `use_idf=False` (utiliza solo la frecuencia de los trigramas, sin ponderar por la especificidad del mismo), para capturar patrones locales en las sentencias.

Embeddings de Sentencias (Transformer): Se utilizaron modelos de lenguaje pre-entrenados, específicamente all-MiniLM-L6-v2 [3, 4], para generar representaciones vectoriales densas de las sentencias. Estos embeddings, basados en la arquitectura Transformer, capturan información semántica y contextual de las consultas.

3.3. Reducción de Dimensionalidad

Dada la alta dimensionalidad de las representaciones generadas, se aplicaron técnicas de reducción para optimizar el proceso de modelado y reducir el uso de recursos computacionales.

PCA (Principal Component Analysis): Se aplicó PCA con el objetivo de reducir la dimensionalidad de los vectores densos de 384 componentes a 50 componentes (se explica el por qué de este número en los siguientes párrafos de t-SNE). También se intentó aplicar PCA a la matriz rala, pero el rendimiento no fue óptimo, donde las primeras 100 componentes explicaron solo el 9 % de la varianza. Esto sugiere que PCA no estaba siendo efectivo para reducir la dimensionalidad mientras conserva la información relevante de los datos, derivando en una gran pérdida de información.

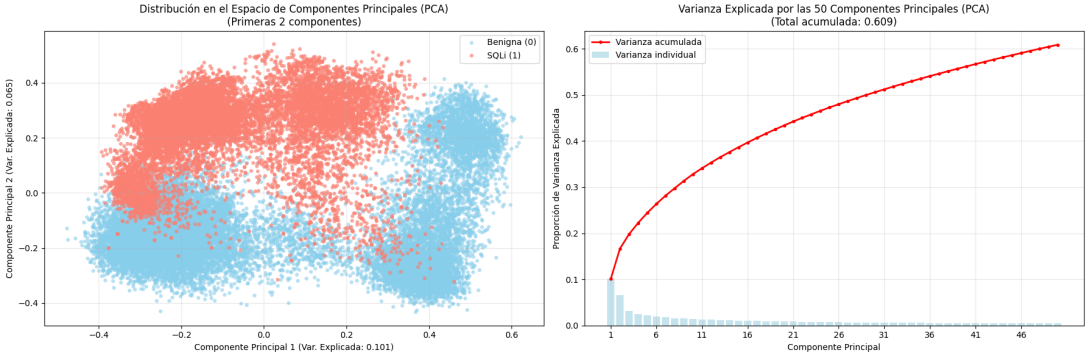

Fig. Gráfico de primeras 2 componentes PCA y varianza explicada acumulada

Truncated SVD (Singular Value Decomposition): Se aplicó SVD como una alternativa a PCA para la reducción de dimensionalidad en vectores ralos, potencialmente más adecuado dadas las características de los datos transformados. Se observó que para capturar el 50 % de la varianza, eran necesarias 112 componentes; para el 70 %, eran necesarias 609 componentes; y para capturar el 80 % de la varianza se requerían aproximadamente 1200 componentes. La varianza explicada por la componente 1 era de 0.0696, y la varianza explicada por la componente 2 era de 0.0712, muy superior a los resultados de PCA. Sin embargo, estos valores no son óptimos y demuestran ineficacia de la técnica para este problema.

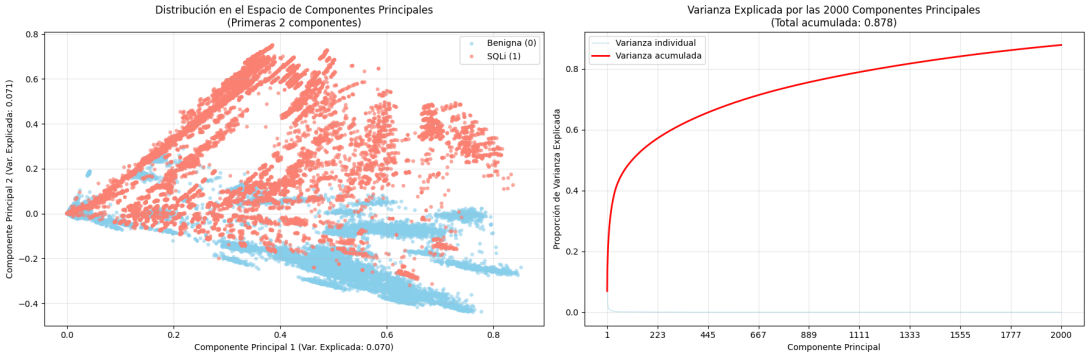

Fig. Gráfico de primeras 2 componentes SVD y varianza explicada acumulada

Como el resultado fue muy malo, necesitando alrededor de 1200 componentes para explicar el 80 % de la varianza de los datos, se exploraron otras técnicas, como la transformación a vectores densos en lugar de ralos, lo que ampliaba a su vez el abanico de posibilidades para la reducción de dimensionalidad. Aún así, se continuó el flujo del pipeline con la representación rala reducida mediante SVD.

PCA + t-SNE (t-Distributed Stochastic Neighbor Embedding): Se exploró la combinación de PCA y t-SNE para la reducción de dimensionalidad, buscando preservar la estructura de los datos en un espacio de menor dimensión. Primero se aplicó PCA para reducir ruido y bajar la dimensionalidad a un tamaño manejable (50 componentes), como recomendaba la librería `scikit-learn` [2], y luego se ejecutó t-SNE sobre esa salida. Esto es necesario, porque t-SNE tiene una complejidad computacional de $O(n^2)$, lo que significa que el tiempo de ejecución crece cuadráticamente con el número de muestras (n) y el número de dimensiones de entrada.

t-SNE es particularmente útil para capturar relaciones no lineales entre instancias, y esto se ve en los gráficos, donde la separación entre sentencias benignas y maliciosas se ve más clara que con PCA solo. Se probaron valores distintos para dos parámetros ajustables. El primero, la "perplejidad", que, de manera resumida, indica cómo equilibrar la atención entre los aspectos locales y globales de los datos. Cuando la perplejidad es baja, intenta agrupar pocos puntos, y cuando es alta, intenta mantener muchos puntos juntos (se aprecia en los siguientes gráficos). Y el segundo, la cantidad de iteraciones que realizará el algoritmo. En ambos casos, no hay un valor ideal conocido, sino que hay recomendaciones de cómo probar distintas configuraciones, hasta que se alcance un estado con algún patrón visible que sea de utilidad.

Para este trabajo, se utilizó t-SNE más como una herramienta para resumir los datos en menos características, y así mejorar la velocidad de ajuste de los modelos, en lugar de usarlo para detectar algún tipo de patrón visible en los gráficos [5].

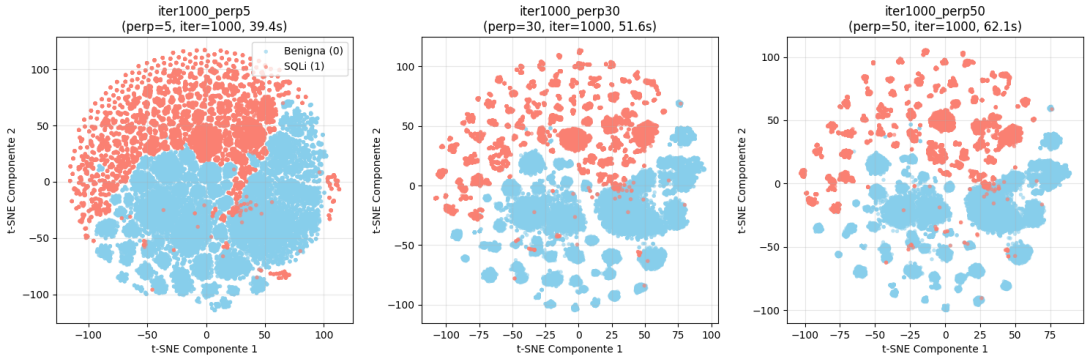

Fig. Comparación de t-SNE con 1000 iteraciones

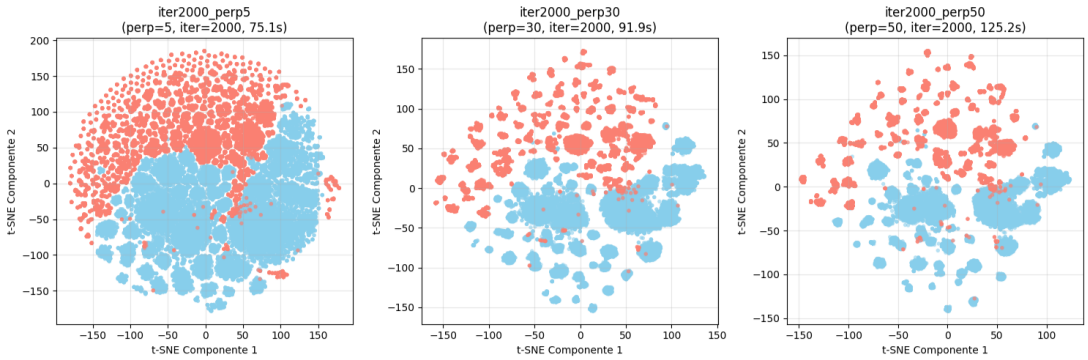

Fig. Comparación de t-SNE con 2000 iteraciones

3.4. Separación del Conjunto de Entrenamiento y de Pruebas

Se separó el corpus de datos ya transformado con la función `train_test_split()` proporcionada por la librería `Sklearn`. Se separó en dos conjuntos: uno de entrenamiento con el 80 % de las muestras, y uno de pruebas con el 20 % restante. Se utilizó el parámetro `stratify=y` sobre la variable objetivo (y) para preservar la proporción de sentencias benignas y maliciosas en ambos conjuntos. Es decir, fuerza que mantengan aproximadamente la misma distribución de clases que el dataset original.

3.5. Balanceo de Clases

Como observamos anteriormente, el dataset presenta un desbalance significativo entre las clases. Para mejorar el rendimiento de los modelos y evitar sesgos hacia la clase mayoritaria, aplicaremos la técnica SMOTE [6], que genera muestras sintéticas de la clase minoritaria creando ejemplos interpolados entre muestras existentes. Se optó por SMOTE por sobre RandomOverSampler, para evitar duplicar ejemplos existentes y fomentar la creación de nuevas instancias que reflejen mejor la distribución de la clase minoritaria.

SMOTE no genera nuevas sentencias de texto, sino que crea nuevos vectores en el espacio de características, ya sean densos o ralos. Una instancia sintética es simplemente un vector intermedio entre dos ejemplos reales de la clase minoritaria, calculado como una interpolación de ambos.

El procedimiento consistió en dividir primero los datos de forma estratificada en conjuntos de entrenamiento, validación y prueba, y luego aplicar SMOTE únicamente en el de entrenamiento. Por otro lado, el conjunto de pruebas no fue alterado con ninguna técnica de sobremuestreo, evitando modificar la distribución real y agregar sesgo.

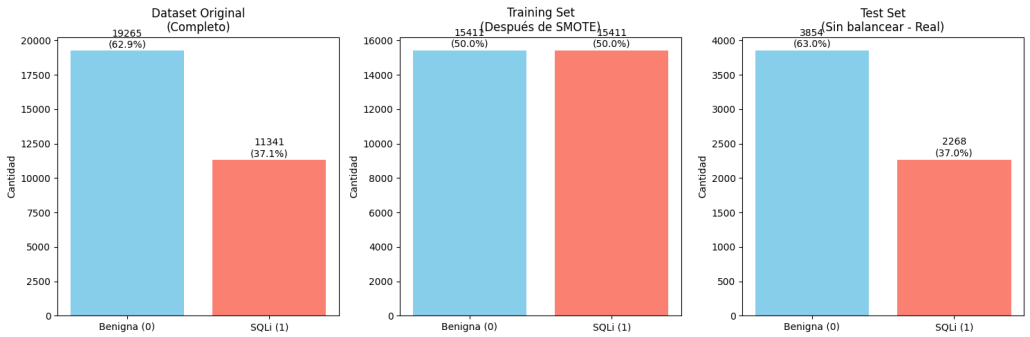

Fig. Comparación de las distribuciones de clases de los datasets

3.6. Modelado

Luego del balanceo del conjunto de pruebas, se realizó la selección de modelos. Se compararon Random Forest, Logistic Regression y SVM con kernel lineal, cuyos hiperparámetros se ajustaron mediante GridSearchCV con validación cruzada estratificada de 5 folds, priorizando métricas que penalizaran falsos negativos, como recall sobre la clase SQLi. El espacio de búsqueda fue razonablemente acotado para SVM por motivos de coste computacional y de tiempo, que superaba con creces a los de los otros modelos.

Se espera que Random Forest destaque por sobre los demás modelos, debido a que suele adaptarse bien a representaciones de alta dimensionalidad y datos con relaciones no lineales. Al estar basado en un conjunto de árboles de decisión, el modelo es capaz capturar relaciones complejas entre características y reducir el impacto de atributos irrelevantes que pueden agregar ruido [12].

4. Resultados

Se utilizaron las métricas de precisión, recall y F1-score, centradas en la clase de inyección SQL (positiva), que es la clase crítica y la que interesa clasificar correctamente, puesto que un falso negativo podría suponer un riesgo para la seguridad del sistema que implemente el modelo como medida de seguridad. En detección de inyecciones SQL, el recall de la clase positiva (inyecciones) es crítico, ya que un falso negativo significa que un ataque pasó desapercibido. Además, se evaluó el rendimiento con matrices de confusión y el método gráfico de ROC-AUC, utilizados para problemas de clasificación binaria.

4.1. Validación cruzada estratificada

Se realizó una búsqueda de los mejores hiperparámetros con validación cruzada de 5 folds, mediante la clase GridSearchCV de Sklearn, utilizando la métrica F1-Score como estimador de referencia. Los mejores hiperparámetros de cada modelo se utilizarán en la prueba final con el conjunto de pruebas. Se adjuntan los resultados obtenidos:

4.1.1. Vectores Ralos con SVD

Modelo	F1-Score	Hiperparámetros
Random Forest	0.9952	{ 'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 50 }
Regresión Logística	0.9929	{ 'C': 10.0, 'solver': 'liblinear' }
SVM (lineal)	0.9908	{ 'C': 1.0, 'cache_size': 200 }

4.1.2. Embeddings con PCA

Modelo	F1-Score	Hiperparámetros
Random Forest	0.9950	{ 'max_depth': 20, 'min_samples_split': 5, 'n_estimators': 50 }
Regresión Logística	0.9903	{ 'C': 10.0, 'solver': 'liblinear' }
SVM (lineal)	0.9905	{ 'C': 1.0, 'cache_size': 200 }

4.1.3. Embeddings con PCA y t-SNE

Modelo	F1-Score	Hiperparámetros
Random Forest	0.9974	{ 'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 50 }
Regresión Logística	0.9029	{ 'C': 0.1, 'solver': 'lbfgs' }
SVM (lineal)	0.9055	{ 'C': 1.0, 'cache_size': 200 }

4.2. Evaluación en el conjunto de pruebas

Se detallan en forma tabular los resultados obtenidos luego de correr los modelos con el conjunto de testing, compuesto por datos nunca vistos por los modelos. Además, se adjuntan las matrices de confusión y curva ROC de cada modelo con los mejores hiperparámetros encontrados.

4.2.1. Vectores Ralos con SVD

Modelo	Accuracy	Precision (pos)	Recall (pos)	F1 (pos)	AUC
Random Forest	0.9948	0.9979	0.9859	0.9929	0.9994
Regresión Logística	0.9962	0.9969	0.9929	0.9949	0.9993
SVM (lineal)	0.9990	0.9982	0.9925	0.9954	0.9990

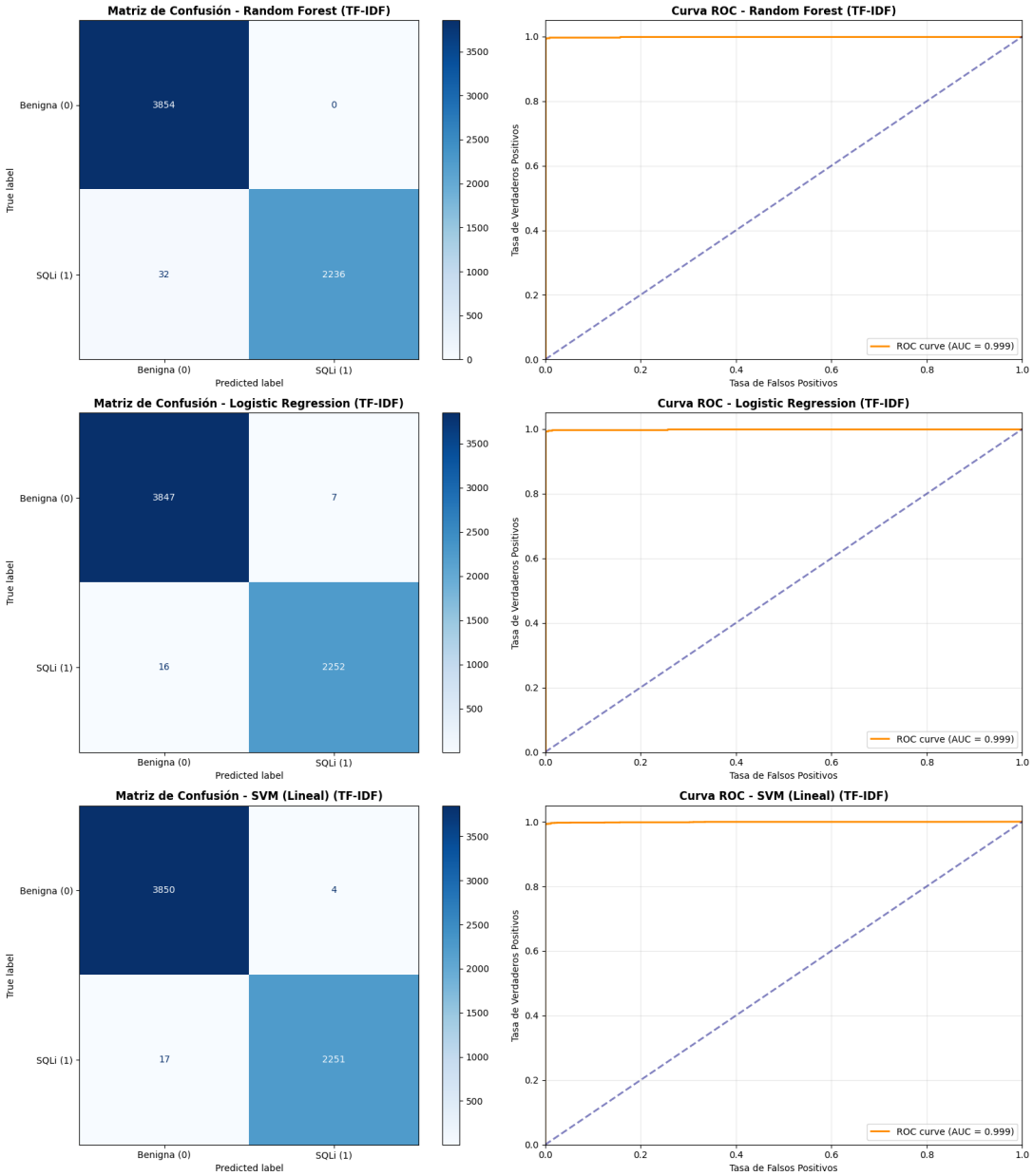

Fig. Matriz de confusión y curva ROC para los mejores modelos con matriz rala y SVD

4.2.2. Embeddings con PCA

Modelo	Accuracy	Precision (pos)	Recall (pos)	F1 (pos)	AUC
Random Forest	0.9951	0.9960	0.9907	0.9934	0.9997
Regresión Logística	0.9915	0.9881	0.9890	0.9885	0.9990
SVM (lineal)	0.9920	0.9907	0.9877	0.9892	0.9986

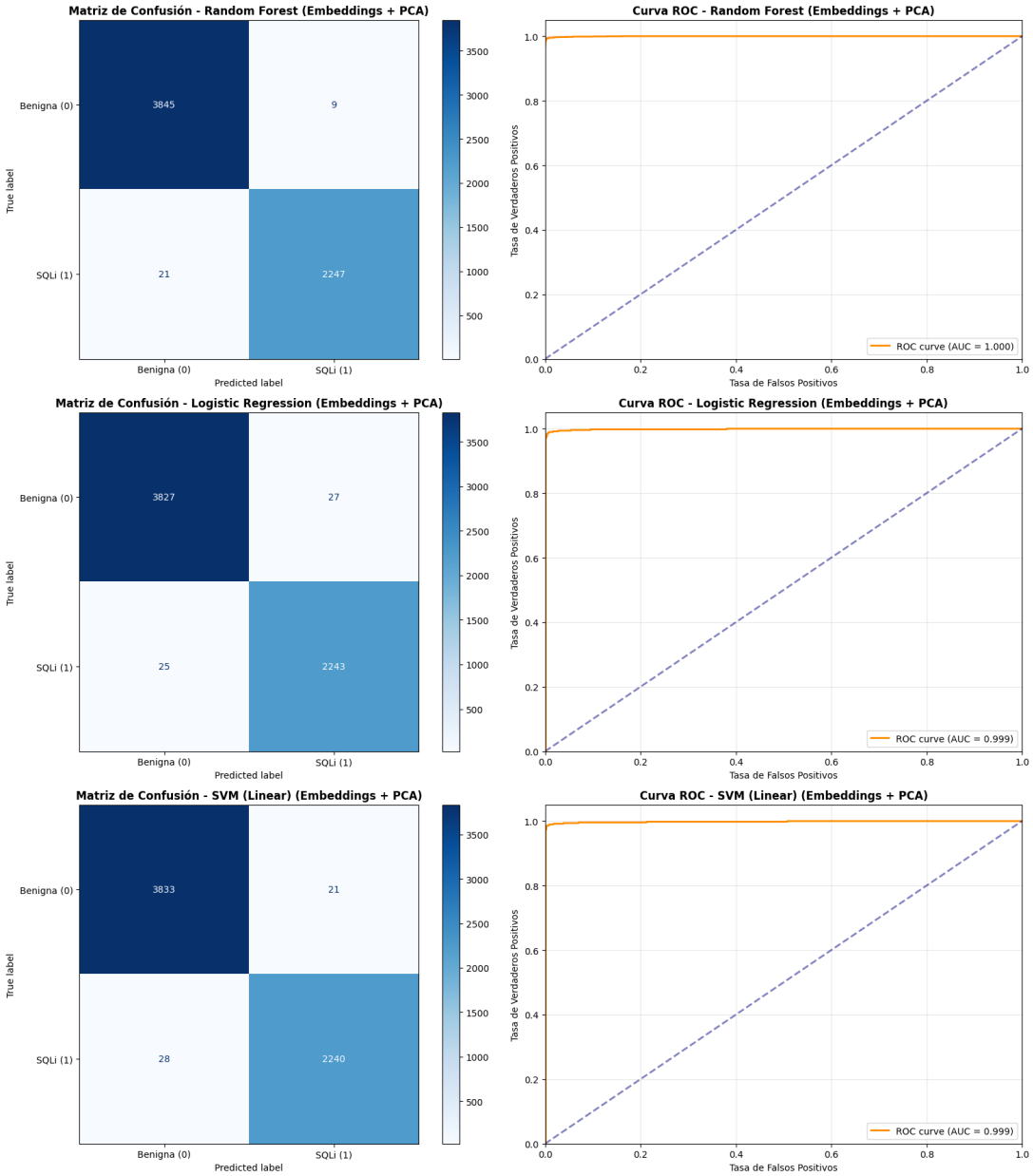

Fig. Matriz de confusión y curva ROC para los mejores modelos con matriz densa y PCA

4.2.3. Embeddings con PCA y t-SNE

Modelo	Accuracy	Precision (pos)	Recall (pos)	F1 (pos)	AUC
Random Forest	0.9979	0.9991	0.9951	0.9971	0.9987
Regresión Logística	0.8973	0.8333	0.9034	0.8669	0.9725
SVM (lineal)	0.8956	0.8223	0.9162	0.8667	0.9725

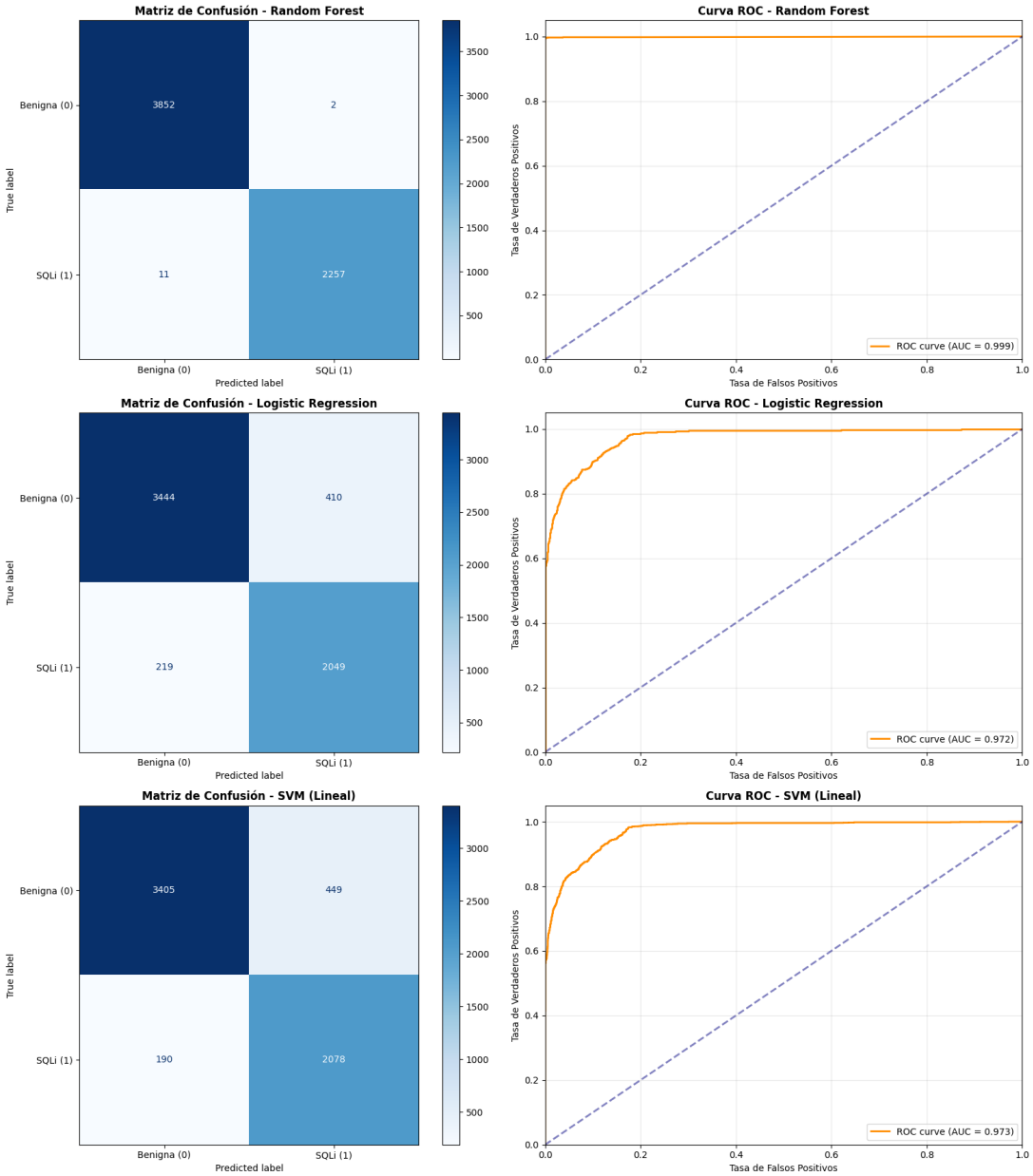

Fig. Matriz de confusión y curva ROC para los mejores modelos con matriz densa y PCA con t-SNE

5. Comparación con trabajos relacionados

En esta sección se comparan los resultados obtenidos en comparación con investigaciones recientes mencionadas anteriormente, que utilizan técnicas similares. Se han incluido los trabajos de **Rakha S. Pratama et al.**, **Alkhathami & Alzahrani**, **Venkatramulu et al.** y **Chen et al.** para la comparación de modelos basados en representaciones ralas y densas.

Trabajo	Dataset / Fuente	Modelo destacado y representación usada	Métrica (Reportada)
Rakha S. Pratama et al. [8]	Kaggle (Syed S. Hussain) [1]	SVM (TF-IDF / Rala)	Accuracy: 99.82 %
Alkhathami & Alzahrani [9]	Kaggle (Syed S. Hussain)	SVM (CountVectorizer / Rala)	Accuracy: 99.42 %
Venkatramulu et al. [10]	Kaggle (Syed S. Hussain)	XGBoost (Unigram CountVectorizer / Rala)	Accuracy: 99.40 %
Chen et al. [11]	GitHub & exploit-db	MLP (Word2Vec / Densa)	Accuracy: 98.57 %
Este trabajo	Kaggle (Syed S. Hussain)	Random Forest (SentenceTransformer+PCA+t-SNE)	Accuracy: 99.79 %

Cuadro 1: Comparación de métricas entre trabajos relevantes y el presente estudio

Se observa que los resultados obtenidos en este trabajo son competitivos y consistentes con la literatura. Si bien el modelo de Pratama et al. reporta una precisión levemente superior (+0.03 %), el enfoque del presente trabajo logra superar el desempeño reportado por Venkatramulu et al., Alkhathami & Alzahrani y Chen et al.. En primer lugar, SVM con vectores ralos reducidos con SVM también obtuvo las mejores métricas, al igual que los trabajos mencionados. Luego, Random Forest, junto con una representación densa mediante embeddings generados por un modelo de lenguaje preentrenado y una reducción de dimensionalidad adecuada, alcanza un rendimiento elevado (99.79 %) y estable, comparable con los mejores resultados reportados en la literatura.

6. Limitaciones del Estudio

Se debe considerar que la reproducción de los experimentos se vió limitada por la falta recursos computacionales para poder ejecutar los modelos con representaciones de los datos con menor pérdida de información, y mejores parámetros. Sin embargo, esto no se vió reflejado en los resultados finales.

Además, el modelo está entrenado con un dataset específico que puede no representar todos los tipos de inyecciones SQL existentes. Se pueden realizar pruebas exhaustivas con más datasets para validar la capacidad de generalización del modelo.

7. Conclusiones

Después de realizar el análisis con validación cruzada, y la evaluación con los datos de prueba, que no habían sido vistos por los modelos, se concluyó que la representación de sentencias SQL mediante la estrategia de matriz rala, generada a partir de la tokenización de ellas en trigramas de caracteres, con reducción de dimensionalidad mediante SVD, demostró ser efectiva. A su vez, la estrategia de matriz densa, con reducción de dimensionalidad mediante PCA, también demostró ser efectiva. Sin embargo, la estrategia de matriz densa, con reducción dimensional con PCA combinada con t-SNE, mostró un rendimiento inferior al de las otras dos.

Luego de la ejecución y evaluación de los modelos, Random Forest obtuvo una precisión perfecta para detectar la clase benigna usando como entrada la representación de vectores ralos reducidos con SVD, con una tasa de 0 falsos negativos, y también se destacó por sobre los otros modelos en la representación densa reducida con t-SNE, siendo el único con un accuracy de 0.99. Sin embargo, Regresión Logística y SVM demostraron ser menos efectivos con esta última representación, obteniendo un accuracy de 0.89. Esto es entendible debido a la pérdida de información que se produjo durante la reducción a dos componentes. Sin embargo, el mejor resultado fue de SVM con la representación rala reducida con SVD, que obtuvo el mejor accuracy entre todos los modelos (0.999).

Este trabajo confirma la hipótesis inicial de que **es posible detectar ataques de inyección SQL mediante técnicas de Machine Learning con alta precisión** (>99 % accuracy en conjunto de test en la mayoría de los modelos), utilizando representaciones ralas como TF-IDF de n-gramas de caracteres, o representaciones densas obtenidas de Sentence Transformers, en combinación con algoritmos de clasificación clásicos.

En función de los resultados obtenidos y del balance entre rendimiento y costo computacional, para un pipeline en producción se recomendaría emplear un modelo como Random Forest (que demostró en todas las representaciones un rendimiento estable) junto con una representación basada en frecuencias de n-gramas de caracteres. Esta combinación demostró una alta precisión junto con un menor requerimiento de recursos en comparación con el uso de embeddings, lo que facilita su despliegue en en sistemas que requieren procesamiento en tiempo real. No obstante, si se dispone de mayores recursos y se prioriza la captura de patrones más complejos, el uso de embeddings también sigue siendo una alternativa válida, a costa de un mayor tiempo en la fase de transformación o vectorización.

Referencias

[1] Syed Saqlain Hussain. Sql injection dataset. Kaggle, 2020. https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset.

[2] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[3] Hugging Face. all-minilm-l6-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2, 2021.

[4] Nils Reimers and Iryna Gurevych. Pretrained models — sentence transformers documentation. https://sbert.net/docs/sentence_transformer/pretrained_models.html, 2019.

[5] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively, 2016.

[6] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[7] OWASP Foundation. Owasp top 10:2025 rc1. https://owasp.org/Top10/, 2025.

[8] Rakha Satria Pratama, Muhamad Irsan, and Rio Guntur Utomo. Analyzing comparison performance model of machine learning through detection sql injection attack. *Jurnal Ilmiah Pendidikan Informatika (JIPI)*, 9(4):2064–2073, 2024.

[9] Jamilah M Alkhathami and Sabah M Alzahrani. Detection of sql injection attacks using machine learning in cloud computing platform. *Journal of Theoretical and Applied Information Technology*, 100(15):5446–5460, 2022.

[10] S. Venkatramulu, Md. Sharfuddin Waseem, Arshiya Taneem, Sri Yashaswini Thoutam, Snigdha Apuri, and Nachiketh. Research on sql injection attacks using word embedding techniques and machine learning. *Journal of Sensors, IoT & Health Sciences (JSIHS)*, 2(1):55–66, 2024.

[11] Ding Chen, Qiseng Yan, Chunwang Wu, and Jun Zhao. Sql injection attack detection and prevention techniques using deep learning. *Journal of Physics: Conference Series*, 1757(1):012055, 2021.

[12] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.