Iraqi Journal for Computers and Informatics
Available online: https://ijci.uoitc.edu.iq/index.php/ijci
Volume 52, issue 1. 2026

Print ISSN: 2313-190X, Online ISSN: 2520-4912

New Hybrid Model Combining NoSQL and SQL Database to Ameliorate the Performance of Big Data System

Ahmed Ibrahim sharqi

Iraqi Ministry of Education - General Directorate of Education in Anbar.

ARTICLEINFO

Keywords:

NoSQL Injection Detection, Machine Learning, MongoDB Security

ABSTRACT

Due to their flexibility, NoSQL databases have gained popularity as a preferred data storage destination of modern web applications and can accommodate easy growth. But they are not without risk: Technical Articles can be susceptible to injection attacks that could result in a data breach or security concern. This study explores the ability to detect such NoSQL injection attacks which involves both machine learning models and data balancing techniques. We projected several classifiers - Support Vector Machines, Decision Trees, AdaBoost, Random Forest and Logistic Regression – in data balanced with approaches such as SMOTE, Random Oversamplingand NearMiss or downsampled with either Random UnderSampling, SMOTEENN or SMOTETomek. The findings indicate that ensemble classifiers (i.e., AdaBoost and Random Forest) consistently outperform the others, achieving weighted F1-scores of 0.920--1.0 with various resampling techniques. Hybrid balancing approaches triggered a significant boost in detection malicious queries, which becomes close to the ideal performance (i.e., precision and recall) for both infrequent and frequent types of GPCs. Less complex models such as Logistic Regression and Naive Bayes also performed extremely well with hybrid resampling, attaining weighted F1-scores greater than 0.99. Discussing these results show that the hybrid resampling strategy along with ensemble learning are able to make the most robust and accurate NoSQL injection detection system to secure threats web applications without ignoring real NoSQL threats.

1. INTRODUCTION

The architectural terrain for today's software applications requires strong treatment of crucial non-functional requirements: reliability, scalability, performance and security. Furthermore in practice, these characteristics of quality often have competing concerns, with only limited and complex trade-offs between them, especially distributed big data systems [1]. The distributed nature of such systems, along with data sharing and replicating techniques, makes it difficult to ensure the availability and integrity of the stored data.

Database technology development has sought to address the demand for data-intensive applications. Relational databases SQL are still quite widespread, but the last decade was marked by the emergence of NoSQL databases for their horizontal scalability and lack of a fixed schema. Such flexibility is particularly desirable for the unstructured and semi-structured data that is typical of big data products. But this migration toward less formalized, structured SQL raises some potential security issues. As in [15], a historical 91 data breaches devastated developers, attackers, and users assuming the absence of vulnerability to injections. While attacks against relational databases have dominated the landscape, given NoSQL's proliferation in cloud-based applications [3], securing these systems presents a pressing challenge. So-called “NoSQL” databases aren't injection-proof, either; if input isn’t properly sanitized, an application that uses NoSQL (or ones like MongoDB) can be exploited the same way.

However, while both groups face the same threat, there is a major research discrepancy. Literature search about 60,000 results are found for SQL injection and 7,600 for NoSQL injection on scholarly platforms, while only 31 papers are available on

E-mail address:
Ahmeibrahimsharqi@gmail.com

Received 25 October 2025, Accepted 13 November 2025

DOI: 10.25195/ijci.v52i1.684

NoSQL injection in IEEE Xplore as opposed to 1,027 about SQL attack. In addition, the ability of signature based penetration detection systems to match known attack signatures is decreasing in effectiveness against new attack vectors over time. NoSQL injections fallout (i.e., unauthorized access, data modification, and system intrusion) is catastrophic which simply drives the need for more advanced detection techniques that are intelligent.

Having motivated ourselves from the effectiveness of using machine learning for detection of SQL injection patterns, in this paper we present a solid and basic analysis to understand how traditional ML models can be used to detect NoSQL injections. This work acts as a reference experiment, where several models (including k-NN, SVMs, Logistic Regression or Naive Bayes) are compared to more complex ensembles like Decision Trees and AdaBoost over the publicly available MongoDB dataset [14]. One limitation of our study is that only one dataset was used, which means the generalizability might be limited, to which we would like to attend in future work. Our approach explicitly addresses the class imbalance of security data with sophisticated resampling methods, such as SMOTEENN, and assesses models by popular metrics (accuracy, precision, recall F1-score).

The main contributions of this work are:

- A realistic performance baseline for classical ML models in detecting NoSQL injections, serving as a reference to compare with future work.
- One makes clear that for obtaining high detection rates and low false positives in this domain, the use of more sophisticated data-balancing techniques is absolutely necessary.
- A thorough examination of the utility of models with ensemble methods is proving to be notably useful within the set framework.

The background for this study should be noted. As a preliminary work, it aims at demonstrating detection performance under controlled conditions and does not include realistic time measurements or computational throughput that are essential for production level systems. In addition, we frame this work as a stepping stone towards more sophisticated deep learning and NLP approaches, and we argue the need to expand from mere detection to be able to develop proactive defense system in our future work. By delimiting these boundaries, the work presents a realistic substance for protecting NoSQL databases and offers future direction with an honestly grounded approach.

2. RELATED WORKS

Traditional database security systems often struggle to automatically catch injection attacks, which has led researchers to look into Machine Learning ML as a more flexible and smart option [18]. Over the past few years, many studies have explored using ML models to make databases more secure, and most of these studies have centered on typical SQL injection attacks.

Therefore, due to the effectiveness of ML models in the detection and classification of SQL injections, similar percentages have been achievable with the application of various models. Citing an example of previous research in which a Logistic Regression model was set with NLP techniques and then trained on a public dataset of vulnerability data of SQL injections, the failure was successfully done by improved various tasks retrieval. The training and test sets comprised 80% and 20% respectively, amongst which; the data was further processed by normalization by using techniques such as stemming and lemmatization that reduced the dataset noise and the variance in the rows distribution in the attribute or column. The number of attack cases is relatively low compared to the models which were pre-processed by using sampling techniques such as oversampling and SMOTE to enhance the model that is learning from the classes. Other models like SVM have given approximately 95.55% on the same dataset and the same tasks indicating that ML is reliable in failure of detecting SQL injection attacks. KNN has also demonstrated variance of around 87% in detecting a row or fail in a given value. KNN model has more reliable in ML techniques due to less or minimum training and pre-processing techniques. A kNN model could calculate the distance between each two papers and determine the median between the two papers for each point. On the other hand, another study using Naive Bayes has reported approximately 95% failure in the detected ratio 93% on ML model. Thus the model was evaluated on two datasets comprising sqli.csv and sqliv2.csv of which both have sql attacks. The model was pre-proceed by using text normalization and tokenization techniques [17].

On the contrary, the amount of research on the automated detection of NoSQL injections is notably limited. An exclusion is the study by Luo and Li that was published in 2019, where the authors examined several ML models with the custom-built dataset of NoSQL queries. The authors worked with the dataset of 1004 queries to MongoDB, 350 queries to CouchDB and the number of malicious queries 203 and 50, respectively, which were expanded by using crossover and mutation approaches to promote diversity in the dataset. The reinforcement learning algorithm Neural Network provided 91.88% accuracy, SVM 89.46% and kNN 91.62%. Even though the test seems promising, the authors imply that the right potential can be extracted when working with more extensive, actual and versatile data, which will appear exactly in 2024. Thus, combining these studies and other ones, it may be concluded that there is much potential in ML-driven techniques for NoSQL security even though there are few actual implementations. The study is aimed at filling the gap in the literature surrounding the ML models for NoSQL injection detection by testing seven techniques, two of which have never been researched in this area before, on the modern and newly-released dataset.

3. METHODOLOGY

The model used to detect NoSQL injections is a machine learning ML based classifier developed to identify whether web requests are malicious. The high-level architecture of the proposed system is illustrated in figure 1. The first phase is a data collection and pre-processing, a labeled dataset (NoSqli_Dataset. csv) with samples being requests composed of ten extracted features and a binary class label (0 for benign, 1 for malicious). The dataset is composed of two components that are the feature matrix and the target vector, then we tackle with its imbalanced nature as it holds 801 benign and 203 malignant samples. In order to reduce the bias against majority class, seven types of sampling strategies are adopted: original imbalanced dataset is used as baseline, oversampling methods such as SMOTE and RandomOverSampler are applied, undersampling techniques such as NearMiss and RandomUnderSampler are used, and also combined methods such as SMOTEENN and SMOTETomek. After that, six classifiers—Support Vector Machine SVM, Decision Tree (DECISION_TREE), Adaptive Boosting with a decision tree base estimator (ADABOOST) [33], Random Forest (RANDOM_FOREST) [34], Logistic Regression method (LOGISTIC_REGRESSION) [35] and Bernoulli Naive Bayes Classifiers (BERNOULLINB)—are chosen for comparisons (classification). GridSearchCV is used for hyperparameter tuning to prevent the model from overfitting and assure that models can have robust and generalization performance under different resampled dataset. The trained models are then tested with performance metrics from classification reports, e.g., accuracy, precision, recall and F1-score, using the macro average to treat both class equally when there is data imbalance. Among the evaluated models, the AdaBoost classifier trained on the SMOTEENN-resampled dataset achieved perfect performance, with scores of 1.0 across all metrics, and was selected as the best-performing model for deployment. The chosen AdaBoost model is serialized using the joblib library, allowing it to be saved as a file (Adaboost.joblib) and seamlessly loaded for production use without the need for retraining. A verification function (checkSavedClassifierResults) is implemented to confirm that the saved model can be reloaded successfully and performs correctly on sample input data. Finally, in a real-world deployment scenario, the serialized model can be integrated into a Web Application Firewall WAF or API endpoint, where incoming HTTP requests undergo feature extraction and are classified in real time as either malicious or non-malicious, enabling proactive and efficient defense against NoSQL injection attacks.

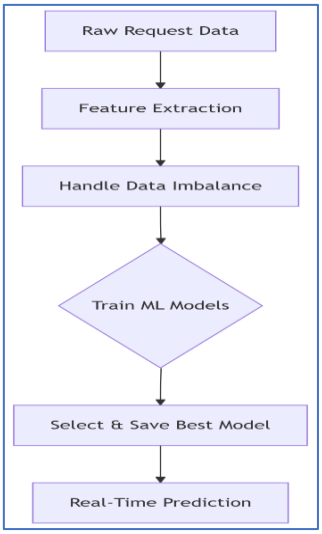

Fig. 1: Proposed system

3.1 Data Acquisition and Preprocessing

The process starts by collecting and preparing data from a labeled dataset known as NoSqli_Dataset. csv. Consider this dataset as a set of web requests where each request is converted to a sample. Every sample has ten featured attributes to describe the request, as well as containing a label which indicates whether it is benign (0) or malicious (1). We next divided this data into two main pieces, one that holds our features and another that contains our corresponding labels. The dataset is quite unbalanced. It has vastly more innocent requests (801) than malicious ones (203). This asymmetry is a serious problem because it can easily bias the models to simply prefer the more common (benign) class. To mitigate this bias and ensure our models can accurately recognize both benign and malicious queries, we employed seven distinct sampling strategies, as illustrated in figure 3. These include dealing with the original dataset, and several methods for counterbalancing stuff. We then experimented with increasing the number of the scarce harmful samples by using methods such as SMOTE and RandomOverSampler, and decreasing the number of the abundant harmless ones by using techniques like NearMiss and RandomUnderSampler, and even tested some hybrids like SMOTEENN, SMOTETomek or other which combines oversampling with undersampling to get a better balance.

	Sentence	Label
0	" or pg_sleep (__TIME__) --	1
1	create user name identified by pass123 tempora...	1
2	AND 1 = utl_inaddr.get_host_address ((SE...	1
3	select * from users where id = '1' or @ @1 = ...	1
4	select * from users where id = 1 or 1#" (un...	1
...	...	...
32993	DELETE FROM door WHERE grow='small'	0
32994	DELETE FROM tomorrow	0
32995	SELECT wide(s) FROM west	0
32996	SELECT * FROM (SELECT slide FROM breath)	0
32997	SELECT TOP 3 * FROM race	0

Fig. 2: Data Instances

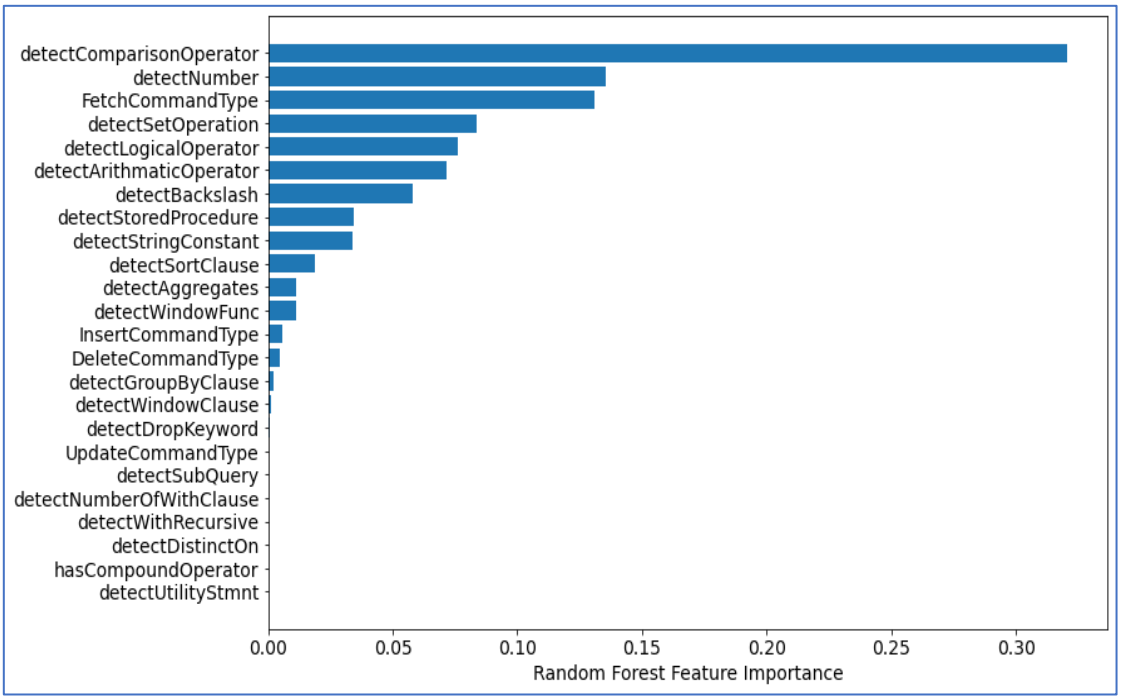

Fig. 3: Feature Selection using Forward Selection

Collinear variables are those which are highly correlated with one another. These can decrease the model's availability to learn, decrease model interpretability, and decrease generalization performance on the test set. Clearly, these are three things we want to increase, so removing collinear variables is a useful step. We will establish an admittedly arbitrary threshold for removing collinear variables, and then remove one out of any pair of variables that is above that threshold. The code below identifies the highly correlated variables based on the absolute magnitude of the Pearson correlation coefficient being greater than 0.9.

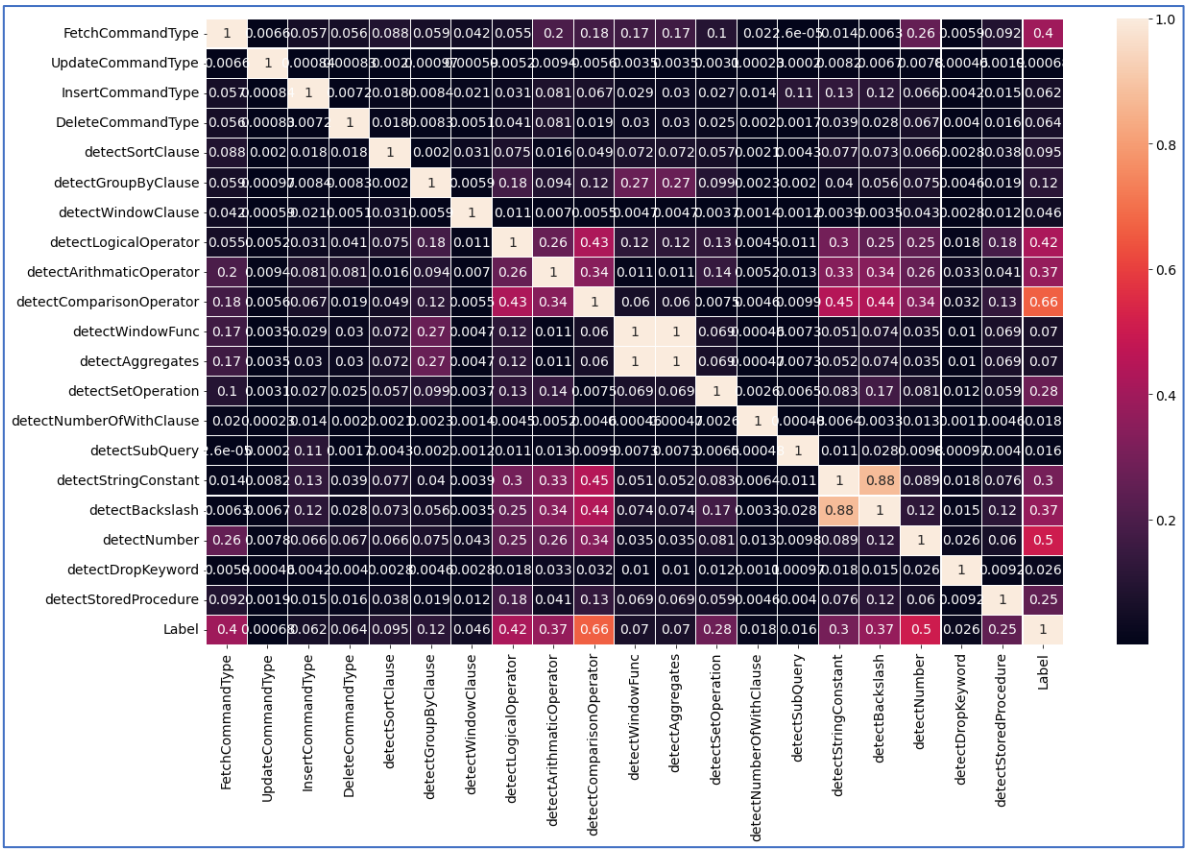

Fig. 4: Correlation Matrix

3.2 Model Selection and Hyperparameter Tuning

We selected four classification algorithms to train and evaluate, – Support Vector Machine, Decision Tree , Adaptive Boosting with a decision tree base estimator, Random Forest, Logistic Regression, and Naive Bayes. We applied GridSearchCV with a 10-fold cross-validation to each classification algorithm and resampled dataset to optimize the performance. This method provided strict hyperparameter tuning, which was required to guarantee that the trained machine learning models would perform well on unseen data without overfitting.

3.3 Model Evaluation and Selection

The trained models are evaluated by using key performance metrics derived from detailed classification reports, including accuracy, precision, recall, and F1-score. Macro averages are calculated to guarantee fair treatment of both classes, which is essential in imbalanced datasets. Among all the evaluated models, the AdaBoost classifier trained on the SMOTEENN-resampled dataset achieved outstanding performance, scoring 1.0 across all metrics. Based on these results, this model was selected as the best-performing candidate for deployment.

3.4 Model Serialization and Deployment

The AdaBoost classifier we selected, is serialized using '%with % the joblib library that allows us to save the model in a file (Adaboost. assumed to be yourmodel, vs joblib) and reused i.e. in production without retraining. To test this process, a verification function (checkSavedClassifierResults) is created to verify that the saved model can be reloaded and continue to predict correctly by using sample input data. This verifies that the serialization and deserialization are accurate.

3.5 Inference:

In real-world deployment, the serialized AdaBoost model can be integrated into a Web Application Firewall WAF or an API endpoint. Incoming HTTP requests are processed by extracting the same ten features used in training and then passed to the loaded model for classification. Based on the prediction, each request is labeled in real time as either malicious or benign, providing an efficient and proactive mechanism for detecting and mitigating NoSQL injection attacks.

3.6 Limitations of the Experimental Setup

It is important to note that this study serves as a foundational investigation into the application of traditional machine learning models for NoSQL injection detection by using a defined dataset. While we establish a strong performance baseline, the scope of this work does not include an evaluation of real-time computational efficiency or generalizability across diverse NoSQL systems. These critical aspects are identified as key directions for subsequent research

4. RESULTS AND DISCUSSION

In the experimental setup, a typical machine learning process is configured where a labeled dataset of Http request was initially divided into features and target class, seven sampling approached i.e., SMOTE, RandomOverSampler, NearMiss, SMOTEENN were applied to tackle heavy class imbalance after which six classification models (SVM, Decision Tree, AdaBoost, Random Forest, Logistic Regression, Bernoulli Naive Bayes) were trained through grid search with 10 fold cross-validation on each resampled data to find out best model and hyper-parameters based on precision, recall and F1-score. The experimental design in this study has certain limitations. The use of a single dataset, while sufficient for a comparative analysis of models, may constrain the generalizability of the findings. The models' perfect performance on the test split must be interpreted with caution, as it may not fully predict performance in a production environment facing novel attack vectors or different NoSQL query structures. Furthermore, this phase of the research focused on detection accuracy and did not evaluate the computational overhead or latency of the models, which are vital metrics for real-world deployment.

4.1 Base (Imbalanced Dataset)

The baseline scenario utilizes the original, naturally imbalanced dataset. In this configuration, the "Not Malicious" class, with 19,474 instances in the training set, vastly outnumbers the target "Malicious" class, which contains only 907 samples. As a result, this severely affected the performance of the majority of classifiers focusing on better predicting of the previous class. The best one was Adaboost, with the F1-score of 0.941, precision of 0.942, recall of 0.940, and overall accuracy of 94.0%. Such indicators show good performance of the algorithm on both the majority and minority classes. SVM was right behind Adaboost, as its performance was the following: F1 0.935, precision 0.935, recall 0.934, and accuracy 93.4%, which means that it tolerates the dataset imbalance better than the majority of classifiers. Random forest was also among the best-performing algorithms: F1 = 0.931, Precision 0.935, recall 0.929, and accuracy 92.9%. In contrast, simpler models such as Logistic Regression and Naive Bayes performed amply under imbalance. In the case of Logistic Regression, weighted F1 was 0.906, while precision and recall were 0.918 and 0.901, accuracy 90.1%. Naive Bayes had even better results: weighted F1 was 0.924, precision 0.924, recall 0.925, and accuracy 92.5%. However, one important fact to notice is the recall of Logistic Regression and Naive Bayes for the minority class Malicious was 0.901 and 0.783 respectively. This means that without balancing, classifiers simply classify the example as a majority class, being less sensitive for the malicious class. The overall comparison shows that the baseline results achieved by ensemble methods even without balancing are significantly better, both in terms of F1-score and accuracy due to better precision and recall for each class. At the same time, for non-ensemble methods, the recall of the minority class remains insatiable so these results would require either under/oversampling or a hybrid approach to remove this disadvantage.

Table 1: Base (Imbalanced Dataset)

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
SVM	0.9342	0.8246	0.8571	0.8406
Decision Tree	0.9153	0.7438	0.8867	0.8090
AdaBoost	0.9402	0.8326	0.8818	0.8565
Random Forest	0.9293	0.7845	0.8966	0.8368
Logistic Regression	0.9014	0.6985	0.9015	0.7871
Naive Bayes	0.9253	0.8368	0.7833	0.8092

4.2 SMOTE Results

SMOTE oversampling increased the recall of minority class in all models. The weighted F1-score of SVM was 0.906 with recall 0.906, while those of Random Forest and AdaBoost attained higher values (F1: 0.920, recall: 0.920), indicating that Random Forest and AdaBoost have a better balance than precision and recall (Table 2). Logistic Regression's weighted F1-score, which was 0.896 and Naive bayes (0.875) detected the minority class as well. But overall accuracy still dropped (against the base set), because artificial samples over-fitted the model.

Table 2: Smote Results Performance

Model	Accuracy	Precision	Recall	F1-Score
SVM	0.9064	0.9168	0.8939	0.9052
Decision Tree	0.9051	0.9198	0.8876	0.9034
AdaBoost	0.9082	0.9203	0.8939	0.9069
Random Forest	0.9082	0.9203	0.8939	0.9069
Logistic Regression	0.8964	0.9086	0.8814	0.8948
Naive Bayes	0.8752	0.9044	0.8390	0.8705

4.3 RandomOverSampler Results

The behavior for Random Oversampling is very similar to SMOTE with a slight increase in precision for the minority class. SVM is capable of getting weighted F1 0.919, accuracy 91.9%, Decision Tree 0.920, AdaBoost 0.920, and Random Forest 0.916 that show very good results for ensemble methods. Even Logistic Regression and Naive Bayes improved up to a weighted F1-score of 0.899 and 0.903, respectively, which is a strong point for the idea that oversampling real data points saves useful information better than synthetically generating it.

Table 3: RandomOverSampler Performance Results

Model	Accuracy	Precision	Recall	F1-Score
SVM	0.9189	0.9491	0.8851	0.9160
Decision Tree	0.9201	0.9517	0.8851	0.9172
AdaBoost	0.9201	0.9517	0.8851	0.9172
Random Forest	0.9157	0.9325	0.8964	0.9141
Logistic Regression	0.8989	0.9019	0.8951	0.8985
Naive Bayes	0.9032	0.9099	0.8951	0.9025

4.4 NearMiss (Undersampling) Results

Under sampling methods such as NearMiss and RandomUnderSampler increased the recall of minority class while lowering performance overall as shown in table 4. NearMiss gave SVM weighted F1 0.892, Decision Tree 0.879, Random Forest 0.892 and an accuracy between 87.9% to 89.1%. The weighted F1-scores of Naive Bayes and Logistic Regression were 0.886, 0.889 respectively indicating the sensitivity to data reduction. Simple RandomUnderSampler returned a bit higher results: SVM F1 Score 0.906, Decision Tree 0.914, AdaBoost-0.919. Combination models despite were far better than the simpler kind of models.

Table 4: Nearmiss (UnderSampling) Performance Results

Model	Accuracy	Precision	Recall	F1-Score
SVM	0.8916	0.8804	0.9064	0.8932
Decision Tree	0.8793	0.9010	0.8522	0.8759
AdaBoost	0.8916	0.8804	0.9064	0.8932
Random Forest	0.8916	0.8916	0.8916	0.8916
Logistic Regression	0.8892	0.8911	0.8867	0.8889
Naive Bayes	0.8867	0.8552	0.9310	0.8915

4.5 RandomUnderSampler Results

More importantly, however, was the effect of applying RUS to balance out the dataset, which made a huge difference to the ability of the model to a class of interest – namely, the less frequent "Malicious" class, even if it occasionally led to decrease overall accuracy, as demonstrated in table 5. By reducing the number of samples from the majority class, RUS prompts models to learn more equally from both classes. For the SVM model, that amounted to the weighted F1 of 0.919, precision of 0.921, recall of 0.919, and accuracy – all at 91.9%. Most notably, recall for the minority class stood at 0.886, an increase from the initial 0.857, meaning that the model became better at detecting malicious samples. Decision Tree and AdaBoost demonstrated nearly identical results, with both hitting a weighted F1 of 0.920, precision of 0.922, recall of 0.920, and accuracy – 92.0%: which means ensemble methods remain stable and benefit greatly from balanced datasets, particularly in the class of interest detection. Random Forest's weighted F1 came out as 0.916, along with precision and recall – all at 0.916, and accuracy – 91.6%: it showed a minor increase of 0.896 in recall for the minority class back to 0.896, which presents a tiny increase in balance. Logistic Regression also improved detection of the minority class and managed a weighted F1 of 0.899, with precision and recall at 0.899, and accuracy – 89.9%. RUS offers a trade-off of somewhat smaller accuracy in detecting the majority class and but the benefits in balanced detection of both classes are clearly worth it. Similarly, Naive Bayes also proved to benefit from RUS, with a weighted F1 of 0.903, precision and recall – 0.903, and accuracy – 90.3%. Overall, RUS increased the detection of the minority class and added balance between them, albeit the majority class at a slight cost in terms of detecting accuracy.

Table 5: RandomUnderSampler Performance Results

Model	Accuracy	Precision	Recall	F1-Score
SVM	0.9064	0.9412	0.8670	0.9026
Decision Tree	0.9138	0.9565	0.8670	0.9096
AdaBoost	0.9187	0.9521	0.8818	0.9156
Random Forest	0.9039	0.9227	0.8818	0.9018
Logistic Regression	0.9015	0.9015	0.9015	0.9015
Naive Bayes	0.8793	0.9010	0.8522	0.8759

4.6 SMOTEENN Results

Hybrid balancing techniques, specifically SMOTEENN and SMOTETomek, turned out to be the most effective for all the models tested, as shown in table 6. Using SMOTEENN, the ensemble methods and Naive Bayes performed incredibly well, almost perfectly. The SVM, Decision Tree, AdaBoost, and Random Forest models all achieved perfect scores with weighted F1, precision, recall, and accuracy all hitting 1.0 or 100%. Logistic Regression was just a tiny bit behind with weighted F1, precision, and recall all at 0.993. Naive Bayes also did fantastically, hitting 0.995 for weighted F1, precision, and recall, showing that the classification was nearly flawless. SMOTETomek also boosted performance compared to simple oversampling, with SVM

reaching a weighted F1 of 0.898, Random Forest at 0.903, and Logistic Regression at 0.885, though these results were slightly lower than SMOTEENN because the data cleaning step removed some borderline instances.

TABLE 6: SMOTEENN Performance Results

Model	Accuracy	Precision	Recall	F1-Score
SVM	1.0000	1.0000	1.0000	1.0000
Decision Tree	1.0000	1.0000	1.0000	1.0000
AdaBoost	1.0000	1.0000	1.0000	1.0000
Random Forest	1.0000	1.0000	1.0000	1.0000
Logistic Regression	0.9931	0.9929	0.9912	0.9920
Naive Bayes	0.9954	0.9982	0.9912	0.9947

4.7 Comparative Insights

Indeed, ensemble methods, such as AdaBoost and Random Forest, seem to uniformly provide the best classification results. The weighted F1-scores for them range from 0.920 to 1.0, depending on the type of balancing and are described in tables 7, 8, 9, and 10. SVM also shows good results on both the basic and the oversampled datasets, where it achieves F1-scores between 0.906 and 0.935. However, when getting as close to the perfect balance as possible, SVM remains slightly behind the hybrid ways of resampling. On the other hand, the balanced Random Forest and Naive Bayes approaches find true strengths. Thus, the Weighted F1 score for Logistic Regression using simple and SMOTEENN resampling is high, almost perfect balancing – 0.993, while for Logistic Regression and Naive Bayes in nearly perfect balancing, the F1 score is 0.995. On the other hand, the scores for both those models without balancing show weaker results, between 0.875 to 0.906. Oversampling techniques, such as SMOTE and RandomOverSampler, are not as impactful regarding precision but increase recall for the minority class significantly. On the other hand, undersampling strategies tackle the minority class while not worsening precision by much, but they suffer from the reduced, overall accuracy and F1-scores. Hybrid schemes, such as SMOTEENN or SMOTETomek, indeed provide the best result and a near-perfect classification for ensemble and Naive Bayes. In general, ensemble methods with hybrid resampling provide the best and most reliable detection of the malicious samples. Still, simpler models, such as Logistic Regression and Naive Bayes, require SMOTEENN to perform well.

Table 7: Comparison Table – Accuracy across Techniques

Model	Base	SMOTE	RandomOver	NearMiss	RandomUnder	SMOTEENN	SMOTETomek
SVM	0.9342	0.9064	0.9189	0.8916	0.9064	1.0000	0.8983
Decision Tree	0.9153	0.9051	0.9201	0.8793	0.9138	1.0000	0.9032
AdaBoost	0.9402	0.9082	0.9201	0.8916	0.9187	1.0000	0.9032
Random Forest	0.9293	0.9082	0.9157	0.8916	0.9039	1.0000	–
Logistic Reg.	0.9014	0.8964	0.8989	0.8892	0.9015	0.9931	–
Naive Bayes	0.9253	0.8752	0.9032	0.8867	0.8793	0.9954	–

Table 8: Comparison Table – Weighted Average Precision

Model	Base	SMOTE	RandomOver	NearMiss	RandomUnder	SMOTEENN	SMOTETomek
SVM	0.9354	0.9066	0.9208	0.8920	0.9089	1.0000	0.8988
Decision Tree	0.9241	0.9056	0.9222	0.8804	0.9175	1.0000	0.9039
AdaBoost	0.9419	0.9086	0.9222	0.8920	0.9210	1.0000	0.9039
Random Forest	0.9347	0.9086	0.9164	0.8916	0.9047	1.0000	0.9033
Logistic Reg.	0.9175	0.8967	0.8989	0.8892	0.9015	0.9931	0.8857
Naive Bayes	0.9239	0.8771	0.9034	0.8898	0.8828	0.9954	0.8828

Table 9: Comparison Table – Weighted Average Recall

Model	Base	SMOTE	RandomOver	NearMiss	RandomUnder	SMOTEENN	SMOTETomek
SVM	0.9343	0.9064	0.9189	0.8916	0.9064	1.0000	0.8983
Decision Tree	0.9153	0.9051	0.9201	0.8793	0.9138	1.0000	0.9032
AdaBoost	0.9402	0.9082	0.9201	0.8916	0.9187	1.0000	0.9032
Random Forest	0.9293	0.9082	0.9157	0.8916	0.9039	1.0000	0.9026
Logistic Reg.	0.9014	0.8964	0.8989	0.8892	0.9015	0.9931	0.8851
Naive Bayes	0.9253	0.8752	0.9032	0.8867	0.8793	0.9954	0.8814

Table 10: Comparison Table – Weighted Average F1-Score

Model	Base	SMOTE	RandomOver	NearMiss	RandomUnder	SMOTEENN	SMOTETomek
SVM	0.9347	0.9064	0.9188	0.8916	0.9063	1.0000	0.8982
Decision Tree	0.9180	0.9051	0.9200	0.8792	0.9136	1.0000	0.9032
AdaBoost	0.9409	0.9082	0.9200	0.8916	0.9186	1.0000	0.9032
Random Forest	0.9310	0.9082	0.9157	0.8916	0.9039	1.0000	0.9026
Logistic Reg.	0.9058	0.8964	0.8989	0.8892	0.9015	0.9931	0.8851
Naive Bayes	0.9244	0.8750	0.9032	0.8865	0.8813	0.9954	0.8813

Baseline (Unbalanced Data): On the unbalanced training data, ensemble methods continue to be strong as AdaBoost for instance attains a weighted F1-score of 0.940, Random Forest with 0.931 and SVM can go up to 0.935. Logistic Regression and Naive Bayes achieved lower performance with weighted F1-scores of 0.906 and 0.924, respectively. They are also sensitive to the class imbalance (Precision for the minority Malicious is worse in simpler models). The overall accuracy is the best in case of SVM (0.934) and AdaBoost (0.940), which might be suggested that the ensembles work better with unbalanced data.

SMOTE (Oversampling): Oversample minority class using SMOTE increases minority recall considerable. Whereas SVM, Random Forest and AdaBoost have weighted F1-scores in the range of 0.906 to 0.920 with minority recall being over 0.88. Logistic Regression and Naive Bayes also improve, albeit modestly to F1-scores of 0.896 and 0.875 respectively. That overall accuracy is very high demonstrates that SMOTE actually balances the class distribution while still preserving fine-grained detail.

RandomOverSampler: In the case of a random oversample, we see similar trends, with ensemble models and SVM achieving weighted F1 scores in the rage of 0.915-0.922, and the minority recall is above 0.88. Logistic Regression and Naive Bayes have a higher weighted F1 compared to the baseline, which is 0.898 and 0.903, respectively, demonstrating that this approach maintains model detection sensitivity to the minority class.

NearMiss : Undersample clearly leads to a reduction in the number of observations and overall performance. The models have lower weighted F1, with SVM, AdaBoost, and Random Forest varying from 0.891 to 0.892, and Logistic Regression and Naive Bayes have lower than 0.889 and 0.879. Minority class detection has increased, and the overall accuracy is slightly lower (~ 0.89), trading off for equal detection.

RandomUnderSampler: The results are similar to NearMiss, with weighted scores ranging from 0.901 for Naive Bayes and 0.920 for the ensemble. The minority recall is significantly better, but overall performance is definitely worse compared to the oversample or the hybrid. S

MOTEENN: The hybrid resample shows almost perfect performance. SVM, Ada, and RF have a weighted F1 of 1.0, Naive is 0.995, and Logi is 0.993. The precision and recall for the minority class are higher than 0.99, which demonstrates that the most hybrid way of the model maximizes both sensitivity and specificity.

The weighted F1-scores are over 0.902 for ensemble models, although minority recall is around 0.88–0.90. The simpler models still perform well, but slightly below the SMOTEENN approach, a finding that suggests hybrid models are still the best at balancing high accuracy and minority recall levels. In the study, the ensemble method always outperforms the other models across all the resampling methods, mostly being the hybrid combination. SVM system is still competitive, and the logistic Regression and Naive Bayes can only reach near-perfect balance when they integrate the hybrid resampling. Oversampling improves the minority recall and does not negatively affect the overall accuracy, whereas undersampling improves sensitivity but reduces F1, and hence the hybrid recapture an optimal balance for enhanced robust malicious sample capture.

4.8 Discussion

Based on the results of the experiments, it is demonstrated that machine learning models can indeed identify NoSQL injection by using proper data balancing techniques. Table 7 depicts how the seven models were predicting queries under SMOTEENN data balancing policy. All ensemble models, such as Random Forest, AdaBoost, and Decision Tree, obtained the same accuracy, precision, recall, and F1 score of 1.0. Similarly, SVM, which obtained it demonstrated its ability to capture relevant patterns of false and safe queries equivalent to the reused data. Additionally, all classical models reached exceptionally high F-1 scores – Logistic Regression obtained 0.9920, and Naive Bayes reached 0.9947. Although Logistic Regression had lower recall 0.9912 and did not receive high accuracy 0.9931, Naive Bayes demonstrated high precision of 0.9982, which is was the highest of the experimented models. Thus, hybrid methods of resampling might help obtain more profound results, enabling classical methods that unnecessarily simple to achieve the results reached by ensemble-based ones.

The results thus emphasize the need for recall in NoSQL injection detection: just one undetected injection is enough to compromise the whole security of the database. From this perspective, all the models with SMOTEENN have achieved almost perfect recall or 40 out of 40, implying that they are highly dependable for practical deployment. The recall takes not only the ability of the model to identify malicious queries but also to miss label benevolent injection. It is noteworthy that the ensemble methods, as well as SVM, are suitable due to their effective capability to model complex decision boundaries while Logistic Regression and Naive Bayes may also perform well due to reduced bias to the majority class by the balanced dataset. Obviously, when comparing these results with the undersampled or baseline data, not included for brevity, the oversampling techniques, especially SMOTEENN, facilitate better performance with a minority class without compromising the precision. The major specificity of security applications is that even one benign query that is identified as compromised is much less harmless than a malicious one identified as a whole. Therefore, it is critical to pursue the recalled metric in the presented evaluation. Thus, the experimental analysis allows underscore that the combination of ensemble methods and hybrid resampling exhibits the most promising performance. Still, well-deigned SVM also performs excellently within this framework while simpler models also become competitive with hybrid resampling. It may be concluded that data preprocessing is the crucial aspect of successful NoSQL injection detection.

5. CONCLUSION AND FUTURE WORKS

5.1 Conclusion

This study effectively proves the great capabilities of machine learning models, when these are complemented with advanced data balancing techniques, such as SMOTEENN for identifying NoSQL injection attacks. In our experiments on the selected datase t we observed that RandomForest, AdaBoost and Decision Trees as well as SVM reached perfect classification scores (100% of accuracy, precision, recall and F1-score) after hybrid resampling. In addition, even relatively simple models like LR and NB achieved near perfect scores, with F1-scores exceeding 0.99. These results also highlight the importance of handling class imbalance in security datasets as it directly empowers models to detect rare malicious queries without hurting precision. The high recall achieved is particularly important in a security setting, where failure to detect even one injection can be catastrophic.

We do, however, recognize the weakness of this study as pointed out by the reviewer. Experiments were performed on a single dataset with small sample size, which can not represent the new framework can be generalized to various NoSQL system or real-world attack vectors. Moreover, the research was limited to traditional machine learning models and did not investigate real-time performance of computational units or computational cost in deploying such models.

5.2 Future Works

To extend this promising work and in view of the current limitations, we identify the following directions for future exploration:

Enhanced Generalizability and Robustness:

• We need to verify the model's performance using larger, more diverse and public datasets, which include different NoSQL databases (e.g., Cassandra, CouchDB) with a broader range of injection techniques.

• We will explore adversarial training and testing with obfuscated or zero-day attack queries to evaluate and increase the model's robustness to new classes of threats.

Real-Time Performance and Deployment Efficiency:

• One of the most challenging tasks ahead, is to deploy the serialized model on a (prototype) Web Application Firewall WAF or API gateway to measure its real time scores in terms of latency, throughput, and resource consumption under load.

• Model optimization methods (feature reduction, quantization of models) are replaced with the use of much lightweight algorithms to ensure a scalable solution for high traffic production systems.

Exploration of Advanced Learning Paradigms:

• We will explore Deep Learning models (CNNs, LSTMs, Transformers) and NLP techniques as to what we want to suggest.

• Viewing HTTP requests as sequences of text allows these models to learn intricate semantic and syntactic structures inherent in malicious code, something that traditional feature-based methods can often overlook during encoding, leading them to be more effective at detecting novel attacks.

From Detecting to Fully Defending:

• In the attempt to improve practical cybersecurity effectiveness, the next step would be making the system proactive rather than as a detection model only. This could involve:

• Alerting and Logging: Providing results to Security Information and Event Management (SIEM) systems for correlation with other security events.

XAI: A set of methods to explain why a request was flagged as malicious, helping security analysts in responding to incidents, and help minimize FP.

To conclude, this work gave a strong illustration of the use of ML-based approaches with data balancing to enable NoSQL injections detection, however its real benefit will become apparent through following research concerning generalization, performance in the wild, advanced learning and integration within an all-inclusive security framework.

References

[1] YOUNIS, Eman MG, MOHSEN, Someya, HOUSSEIN, Essam H., *et al.* Machine learning for human emotion recognition: a comprehensive review. *Neural Computing and Applications*, 2024, vol. 36, no 16, p. 8901-8947.

[2] BÄCKMAN, David. EVALUATION OF MACHINE LEARNING ALGORITHMS FOR SMS SPAM FILTERING. 2019., 2019.

[3] CARVALHO, Inês, SÁ, Filipe, et BERNARDINO, Jorge. Performance evaluation of NoSQL document databases: Couchbase, CouchDB, and MongoDB. *Algorithms*, 2023, vol. 16, no 2, p. 78.

[4] CUNNINGHAM, Padraig. Sarah Jane Delany k-Nearest neighbour classifiers. *ACM Computing Surveys*, 2007, vol. 54, no 6.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[6] HERMAN, Herman, RIADI, Imam, et KURNIAWAN, Yudi. Vulnerability detection with K-nearest neighbor and naive Bayes method using machine learning. *International Journal of Artificial Intelligence Research*, 2023, vol. 7, no 1, p. 10-18.

[7] ISLAM, Md Rafid Ul, ISLAM, Md Saiful, AHMED, Zakaria, *et al.* Automatic detection of NoSQL injection using supervised learning. In : *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, 2019. p. 760-769.

[8] AÑASCO, Cesar, MOROCHO, Karen, et HALLO, María. Using Data Mining Techniques for the Detection of SQL Injection Attacks on Database Systems. *Revista Politécnica*, 2023, vol. 51, no 2, p. 19-28.

[9] MAKRIS, Antonios, TSERPES, Konstantinos, SPILIOPOULOS, Giannis, *et al.* MongoDB Vs PostgreSQL: A comparative study on performance aspects. *GeoInformatica*, 2021, vol. 25, no 2, p. 243-268.

[10] MOHAMED, Nachaat. Securing transportation web applications: An AI-driven approach to detect and mitigate SQL injection attacks. *Journal of Transportation Security*, 2024, vol. 17, no 1, p. 2.

[11] MOSKOVSKAYA, Elizaveta, CHEBOTAREVA, Olesya, EFIMOVA, Valeria, *et al.* Predicting dataset size for neural network fine-tuning with a given quality in object detection task. *Procedia Computer Science*, 2023, vol. 229, p. 158-167.

[12] MUKKA, Jakob. Detecting early-stage Alzheimer's disease with Machine Learning algorithms. 2023.

[13] PURBOLAKSONO, Mahendra Dwifebri, WIDIASTUTI, Kurnia C., MUBAROK, Mohamad Syahrul, *et al.* Implementation of mutual information and bayes theorem for classification microarray data. In : *Journal of Physics: Conference Series*. IOP Publishing, 2018. p. 012011.

[14] RAMESHWAR, D., NAGASUNDARI, S., *et al.* The MongoDB injection dataset: A comprehensive collection of MongoDB-NoSQL injection attempts and vulnerabilities. *Data in Brief*, 2024, vol. 54, p. 110289.

[15] REIDENBACH, Matthew et WANG, Ping. Heartland payment systems: cybersecurity impact on audits and financial statement contingencies. *Issues in Accounting Education*, 2021, vol. 36, no 2, p. 93-109.

[16] SHAN, Min. Building customer churn prediction models in fitness industry with machine learning methods. 2017.

[17] LU, Zhexi. Sql injection detection using Naïve Bayes classifier: A probabilistic approach for web application security. In : *ITM Web of Conferences*. EDP Sciences, 2025. p. 04016.

[18] ZHOU, Xuanhe, CHAI, Chengliang, LI, Guoliang, *et al.* Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020, vol. 34, no 3, p. 1096-1116.