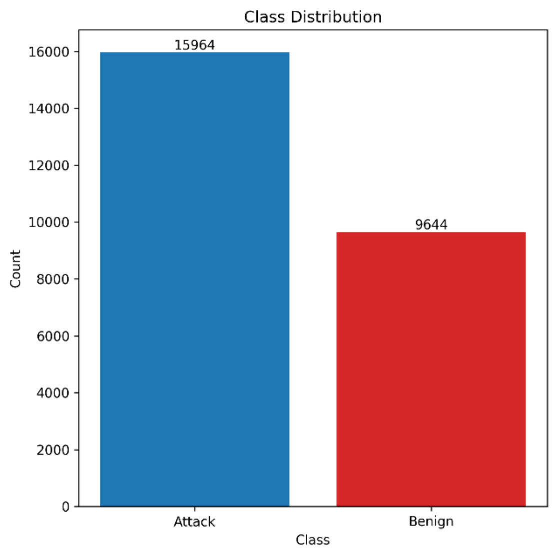
Final Report - SQL Injection Detection using Machine Learning

1. Data and Feature Description

Dataset Overview

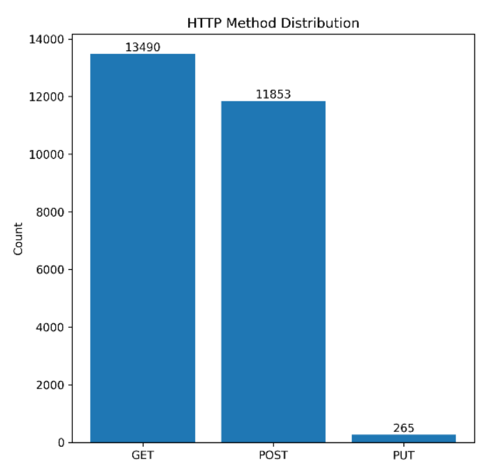
The analysis utilized the CSIC 2010 HTTP Dataset, containing HTTP traffic data for detecting SQL injection attacks. The dataset consists of:

- **Total samples**: 25,608 HTTP requests
- **Attack samples**: 15,964 (62.3% of total)
- **Benign samples**: 9,644 (37.7% of total)
- **Class distribution**: Moderately imbalanced favoring attack samples



Data Characteristics

- **Average request length**: 196.3 characters
- **Request length range**: 31 - 895 characters
- **HTTP methods distribution**:
  - GET: 52.7%
  - POST: 46.3%
  - PUT: 1.0%



Feature Engineering

The model employs a multi-stage feature extraction pipeline:

Text Features (TF-IDF Vectorization)

- **Input**: Combined URL and request body (`url + " " + body`)
- **Vectorizer**: TfidfVectorizer with character-level n-grams
- **Parameters**:
  - `ngram_range`: (3, 5) - 3 to 5 character sequences
  - `analyzer`: "char_wb" - character within word boundaries
  - `min_df`: 10 - minimum document frequency
  - `max_features`: 300 - moderate feature space
- **Purpose**: Capture character-level patterns indicative of SQL injection payloads

Categorical Features

- **HTTP Method**: One-hot encoded (GET, POST, PUT)
- **Purpose**: Account for method-specific attack patterns

Numerical Features

- Request length, URL length, body length
- Special character counts and ratios
- Suspicious keyword counts
- Entropy

2. Selected Algorithm and Parameters

Algorithm: Logistic Regression

Rationale: Logistic regression was selected for its:

- Interpretability (feature importance analysis)
- Computational efficiency
- Probabilistic outputs for threshold tuning
- Established performance in text classification tasks

Model Configuration

- **Regularization**: L2 (Ridge)
- **Solver**: liblinear (suitable for small datasets)
- **Class weighting**: "balanced" (addresses class imbalance)
- **Maximum iterations**: 200

Hyperparameter Optimization

Grid search was performed over regularization strength (C parameter):

- **C values tested**: [0.1, 1.0, 10.0]

- **Cross-validation**: 5-fold stratified
- **Optimization metric**: ROC AUC

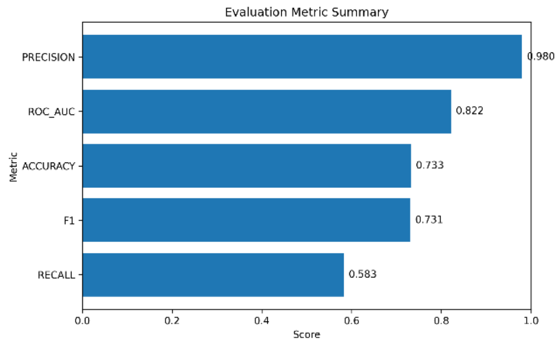
Best hyperparameters: C = 10.0 (highest regularization strength tested)

3. Results with Graphs

Performance Metrics

The final model achieved the following performance on the held-out test set:

Metric	Value
Accuracy	73.30%
Precision	98.01%
Recall	58.32%
F1-Score	73.19%
ROC AUC	82.20%

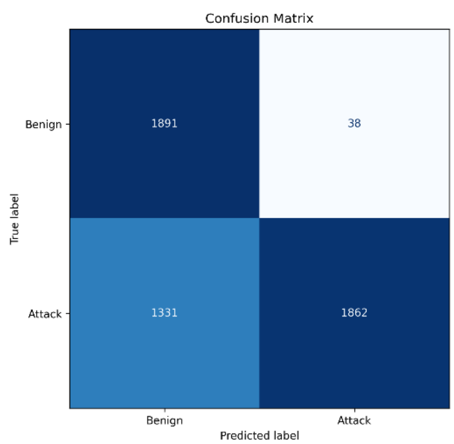


Cross-Validation Results

Grid search cross-validation performance across different regularization strengths:

C Parameter	CV ROC AUC	CV Accuracy
0.1	0.736	0.589
1.0	0.809	0.674
10.0	**0.820**	**0.730**

Confusion Matrix



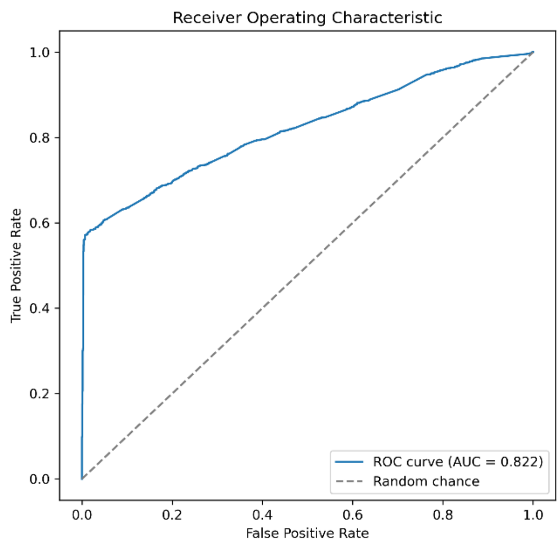
Key Visualizations

Class Distribution

The dataset shows a moderate class imbalance with approximately 62% attack samples and 38% benign samples.

ROC Curve

The Receiver Operating Characteristic curve demonstrates strong discriminative ability with an AUC of 0.822, indicating the model has good ability to distinguish between benign and malicious requests.

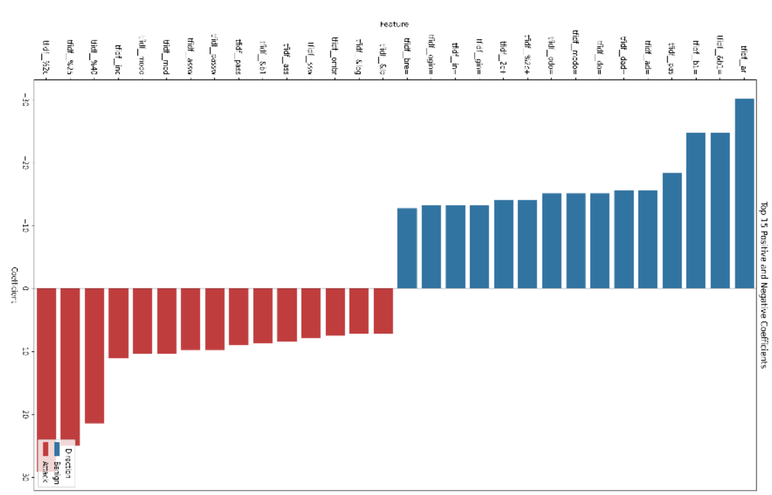


Feature Importance Analysis

Top predictive features ranked by absolute coefficient magnitude:

Attack-Indicative Features:

1. `tfidf__%2c` (coefficient: 19.861) - URL-encoded comma (common in SQL injection)
2. `tfidf__%25` (coefficient: 19.027) - URL-encoded percent sign
3. `tfidf__%40` (coefficient: 15.155) - URL-encoded at sign (@)
4. `tfidf__ass` (coefficient: 7.653) - Fragment of "password" or "assignment"
5. `tfidf__inc` (coefficient: 7.180) - Fragment of "include" or injection patterns
6. `tfidf__pass` (coefficient: 6.088) - Password-related strings
7. `tfidf__login` (coefficient: 3.355) - Login authentication patterns
8. `categorical__method_PUT` (coefficient: 4.374) - PUT method usage



Benign-Indicative Features:

- Standard HTTP constructs with negative coefficients
- Common web application patterns

4. Interpretation

Feature Importance Insights

The model's predictions are primarily driven by character-level n-gram patterns that capture SQL injection signatures:

1. **URL Encoding Patterns**: The strongest predictors are URL-encoded characters (`%2c`, `%25`, `%40`) which are commonly used in SQL injection attacks to bypass input validation and encoding filters.

2. **Authentication Keywords**: Features like "pass", "login", and "ass" fragments indicate the model has learned patterns associated with credential-based attacks and authentication bypass attempts.

3. **Injection Payload Fragments**: Character sequences like "inc" (possibly "include" or injection patterns) suggest the model captures common SQL injection payload structures.

4. **HTTP Method Anomalies**: PUT method usage shows strong association with attacks, likely reflecting unusual HTTP method exploitation.

Model Behavior Analysis

- **Strengths**: The model demonstrates good discriminative ability with 82.20% ROC AUC, successfully identifying URL encoding patterns and authentication-related attack vectors.
- **Precision vs Recall Trade-off**: High precision (98.01%) indicates low false positive rate, but moderate recall (58.32%) suggests some attacks are missed.
- **Feature Learning**: The 300 TF-IDF features provide sufficient representational capacity to capture meaningful attack patterns.

Probability Distribution

The model shows clear separation between classes in predicted probabilities, with attack samples generally receiving higher probability scores, indicating confident classification for detected threats.

5. Limitations and Recommendations

Major Limitations

1. **Precision-Recall Trade-off**: While precision is excellent (98.01%), recall is moderate (58.32%), meaning the model misses approximately 45% of actual

attacks. This conservative approach prioritizes false positive avoidance over attack detection.

2. **Feature Space Constraints**: With only 300 TF-IDF features, the model may miss more subtle or complex attack patterns that require larger feature representations.

3. **Domain Specificity**: The character-level n-gram approach, while effective, may not capture higher-level semantic patterns in SQL injection attacks.

4. **Class Imbalance Impact**: Despite balanced class weighting, the moderate class imbalance (62% attacks) may still influence model behavior.

Recommendations for Improvement

Immediate Improvements

1. **Optimize Classification Threshold**: Adjust the decision threshold to improve recall while maintaining acceptable precision level.
2. **Increase Feature Space**: Expand TF-IDF `max_features` to 1000+ to capture more nuanced attack patterns.

Feature Engineering Enhancements

1. **Domain-Specific Features**: Add features specifically designed for SQL injection detection (e.g., SQL keyword proximity, quote balance, escape sequence patterns).
2. **Temporal Features**: Include request timing patterns if available.
3. **Behavioral Features**: Incorporate session-level patterns and request sequences.

Conclusion

The implemented SQL injection detection system demonstrates promising performance with an 82.20% ROC AUC and strong precision (98.01%), successfully identifying key attack patterns through character-level n-gram analysis. The model effectively captures URL encoding patterns and authentication-related attack vectors, making it suitable for deployment in security-focused applications where false positives must be minimized. Future improvements focusing on enhanced feature representations and threshold optimization could further improve recall while maintaining the current high precision levels.