ABSTRACT

Cyberattacks are one of the most pressing issues of our time. The impact of cyberthreats can damage various sectors such as business, health care, and governments, so one of the best solutions to deal with these cyberattacks and reduce cybersecurity threats is using Machine Learning. In this project, we will be creating an in-depth study model to detect SQL Injection Attacks. The advantage of combining Machine learning with cybersecurity in our system is to detect and prevent short-term attacks without human interaction, so our system can reduce and prevent sql Injection attacks. Many webpages accept the sensitive information (e.g. username, passwords, bank details, etc.) from the users and store this information in the database that also resides over the internet. Despite the fact that this online database has much importance for remotely accessing the information by various business purposes but attackers can gain unrestricted access to these online databases or bypass authentication procedures with the help of SQL Injection Attack. This attack results in great damage and variation to database and has been ranked as the topmost security risk by OWASP TOP 10. Considering the trouble of distinguishing unknown attacks by the current principle coordinating technique, a strategy for SQL injection detection dependent on Machine Learning is proposed. Our motive is to detect this attack by splitting the queries into their corresponding tokens with the help of tokenization and then applying our algorithms over the tokenized dataset. We used four Ensemble Machine Learning algorithms: Random Forest,Gradient Boosting Machine (GBM), Adaptive Boosting (AdaBoost), Extended Gradient Boosting Machine (XGBM), and Light Gradient Boosting Machine (LGBM).

1. INTRODUCTION

Web applications are software that can be run on web browsers and can be used for a variety of purposes, such as social networking, commercial transactions, and academic research. Almost all these kind of web apps are connected to data using RDBMS databases in backend. These databases use SQL to store data such as usernames, passwords, and other sensitive information also. This sensitive data linked to online web apps are vulnerable to attacks, giving attackers to hack these databases to access that data illegally. To maintain the security of your web applications, developers need to be vigilant about protecting data. These databases are more vulnerable to the SQLi attack, which is one of the most riskier security attacks.. Since the last fifteen years, OWASP has considered this vulnerability to be one of the most serious threats. Today, sophisticated software and tools are used to carry out machine-controlled injection attacks. SQLi is a vulnerability that can be exploited to access sensitive information in a web application's database. The most common reason this vulnerability is caused is because insufficient input validation is done, and the vulnerability is explicitly incorporated into a SQL query. A hacker can use these vulnerabilities and can submit malicious SQL queries to the database. Detecting and preventing these SQLi attacks is very important to protect the sensitive information of people. As these loss can cause many problems to people and also to organizations. So leveraging the AI and machine learning can help us to detect and also to prevent these kinds of cyber attacks.

1.1 NEED AND OBJECTIVE

Database-driven websites are vulnerable to SQLi attacks. As this susceptibility is easily spotted and can be used by attackers, almost all the websites that have a user base are targeted by hackers. To perform SQLi attacks, attackers generally adds a character which is not a part of the actual SQL query. SQL injection attacks can have a wide range of impacts, from exposing sensitive information to crashing databases. There are lot of bad consequences of SQLi attacks. The most important one is loss of personal and sensitive information which can be used for malicious activities. Also with these SQLi attacks, authorization information can also be changed. Along with these, the data can be modified and useful information can be lost by SQLi attacks. So, there is a huge need to detect these attacks and also to prevent them, to get rid of all these consequences. This project's objective is to develop ensemble machine learning models. Both injected and standard sql queries, as well as plain text, are used to create a dataset, since we want our model to distinguish between query and plain text, and required queries and queries-based features are extracted from them.

2. LITERATURE SURVEY

2.1 PROBLEM STATEMENT

SQL injection occurred in electronic records in a database, and it continues to persist even after two decades.SQLi attacks are a major threat to web applications. Despite the fact that technology has advanced significantly in recent years, hackers can still exploit security flaws to do SQL injections. Hackers can use a variety of methods to accomplish SQL injections, and there are numerous ways to prevent SQL injections from occurring. The objective of our project is to leverage the ensembling ML techniques to detect various SQLi attacks.

2.2 PROPOSED SYSTEM

Gradient boosting techniques such as XGBoost and LightGBM Boost have made significant progress in recent years. So, in order to address our problem of determining whether or not an employee will be kept, we will use three gradient boosting algorithms: XGBoost, LightGBM Boost, and LightGBM Boost. We will employ the conventional 70 percent train, 30 percent test data splits to accurately train the model, and then we will train the model with several ensemble machine learning method

3. SYSTEM ANALYSIS

"System analysis" and "requirements analysis" are two terms that are interchangeable. Additionally, it can be applied to assist the decision maker in coming up with a better plan of action and decision than he or she would have otherwise. This process includes brainstorming and disassembling the system into its component parts in order to analyze the scenario, evaluate the project objectives, and disassemble what must be built and used to engage people so that clear requirements can be given.

3.1 SOFTWARE REQUIREMENTS:

➢ **Operating System:** Microsoft Windows / Linux / MacOS

➢ **Technology:** Machine Learning

➢ **Tools:** Jupyter Notebook or Spyder

➢ **Platform:** Anaconda Distribution or Google Colab

3.2 HARDWARE REQUIREMENTS:

Processor:

- Any Intel or AMD x86 - 64bit processor is required as a minimum.
- Any Intel or AMD CPU with at least 4 logical cores and hyperthreading support is recommended. Intel i5 8th generation equivalent or higher is preferred.

RAM:

- **Minimum:** 8 GB DDR4 RAM
- **Recommended:** 8 - 16 GB DDR4 RAM or 8 GB DDR5 RAM

4. SYSTEM DESIGN

System design is basically the process of establishing the interfaces, architecture, components, modules along with data for a system to satisfy it's requirements. In a nutshell, it's the application of theory of systems to product creation. The use of object-oriented design and analysis procedures is quickly becoming one of the most common methods for developing computer systems.

4.1 ARCHITECTURAL DESIGN

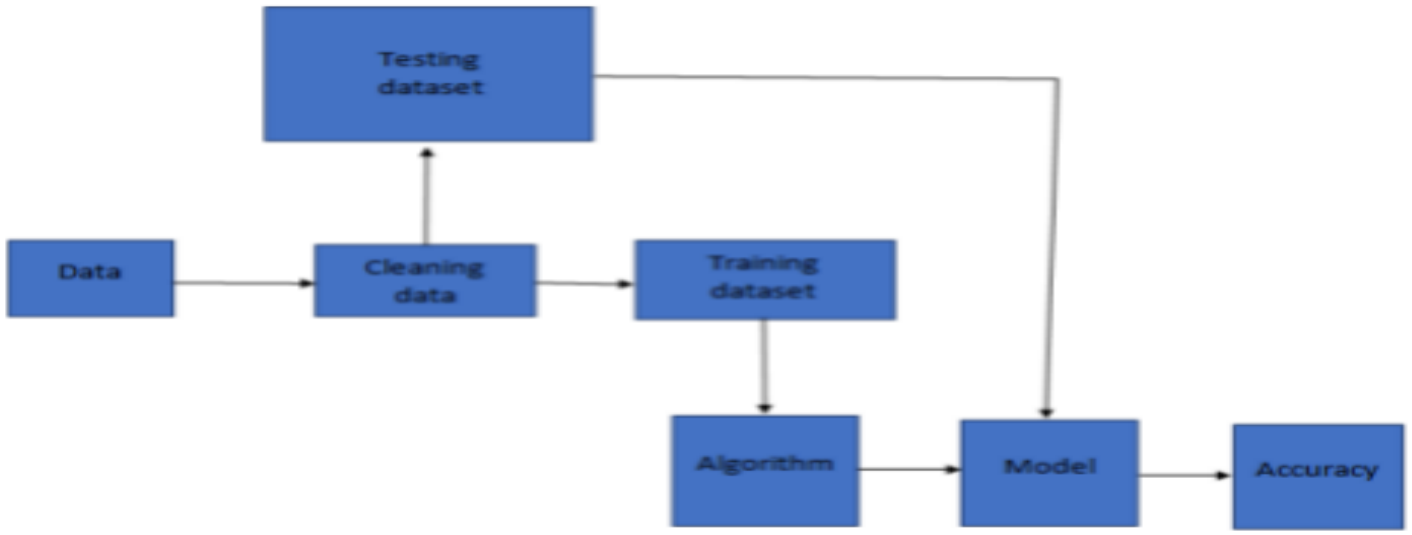


Fig 4.1.1 Architectural flow

Fig 4.1.2 Data flow of mode

4.2 UML DIAGRAMS

CLASS DIAGRAM

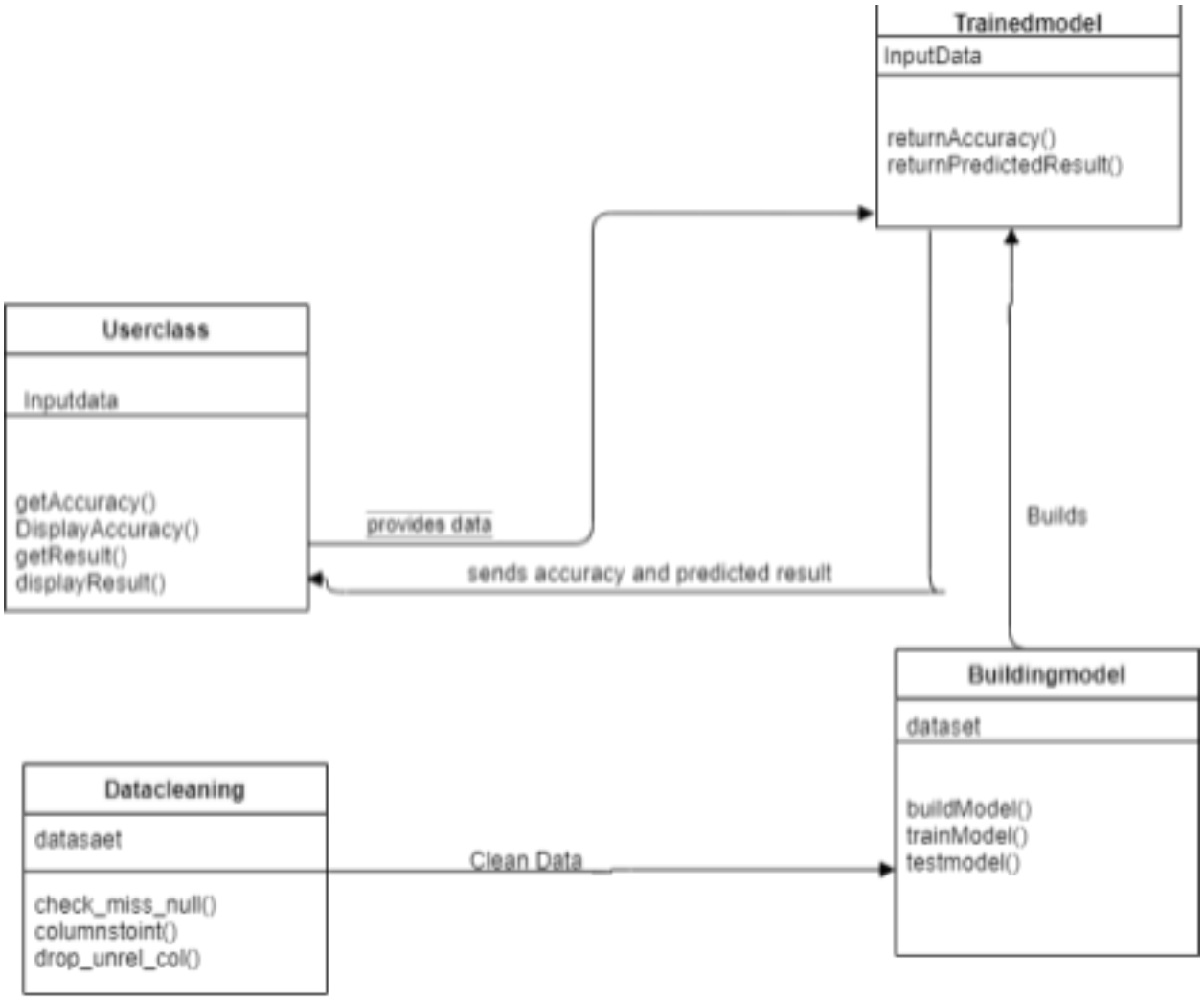


Fig 4.2.1 Class diagram

Class is basically a collection of objects with similar properties, methods, connections, and definitions. Classes that implement one or more interface classes are the most important building blocks of any object-oriented system.

USE CASE DIAGRAM :

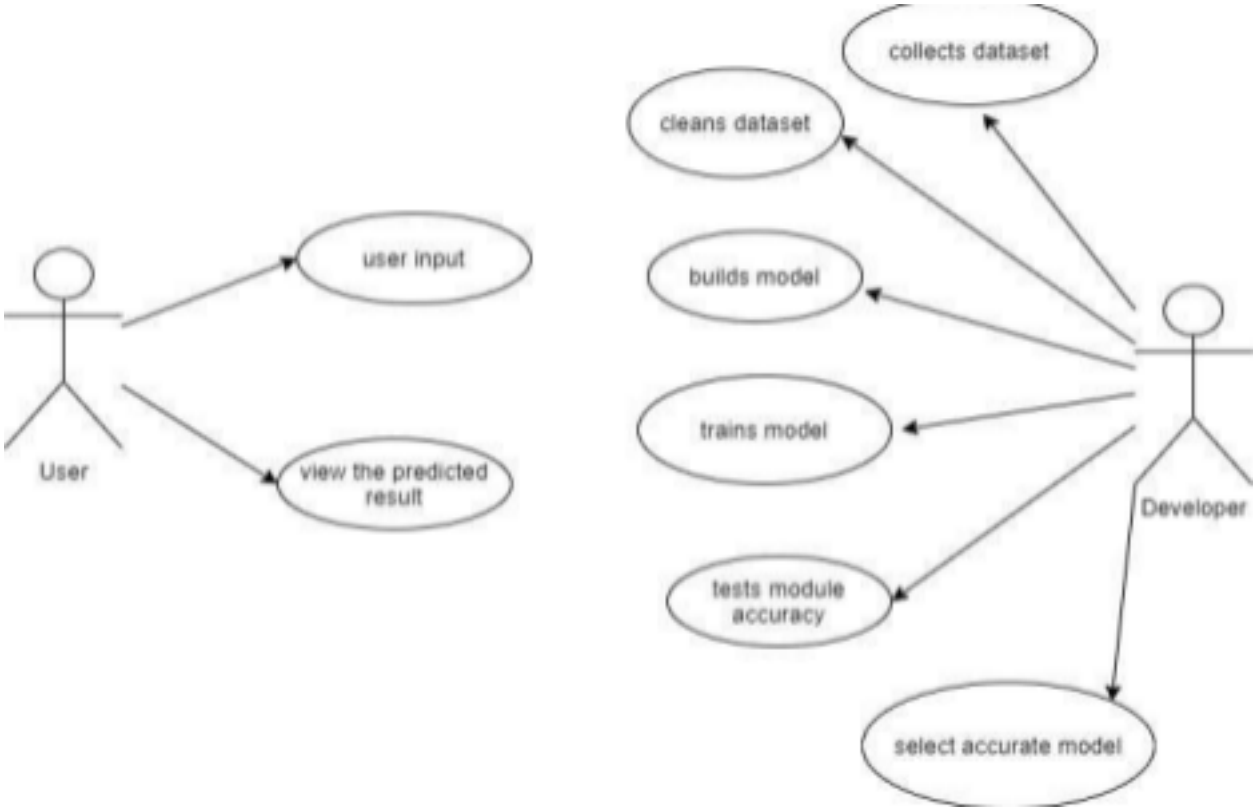


Fig 4.2.2 Use case diagram

Use case diagrams are a particular kind of diagram that are employed to gather data regarding a system's requirements, taking into account both internal and external aspects. The majority of these requirements concern design. Studying a system's capabilities leads to the development of use cases and the identification of actors.

The use case diagram can be used for a variety of purposes, including:

1. Gathering information about a system's requirements.
2. To get a bird's eye view of a system.
3. Determine the system's external and internal impacts.
4. Demonstrate the link between the actors and the criteria.

ACTIVITY DIAGRAM

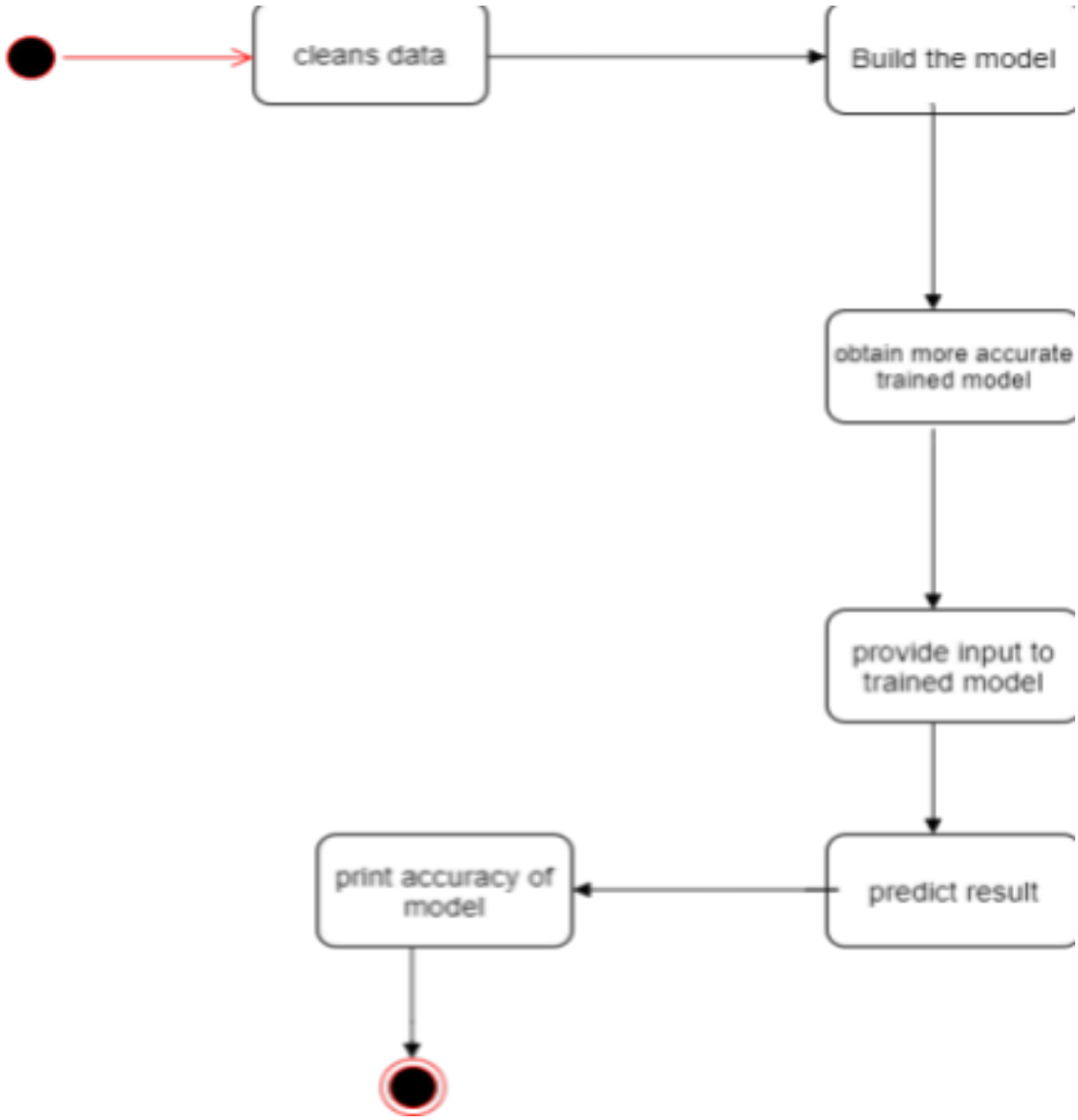


Fig 4.2.3 Activity diagram

It's a state chart diagram that displays the flow of information within a system from one action to the next. It is concerned with the dynamic perspective of a system. Put a lot of attention on the control flow between elements.It illustrates a one-of-a-kind state diagram in which the most of the states are action states and the almost all of the transitions are generated by actions in source states being completed. This figure depicts internal processing-driven fluxes.

SEQUENCE DIAGRAM

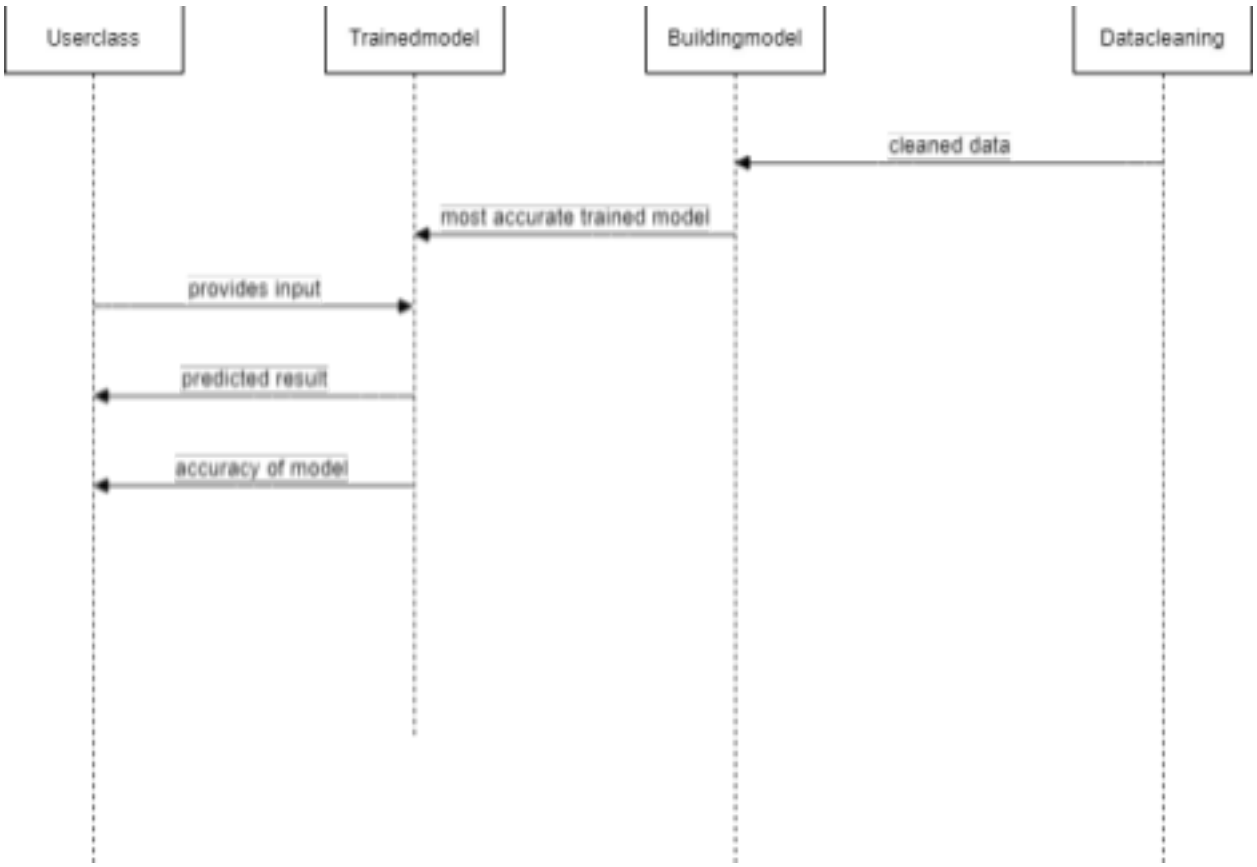


Fig 4.2.4 Sequence diagram

In the sequence diagram, the time ordering of messages is emphasized. It basically shows a collection of objects and the messages they have sent/received, with a concentration on message time ordering. It depicts the sequence in which the various elements of the interaction occur. There are two dimensions to this: vertical (time) and horizontal (space) (different objects).

A identified element in a sequence diagram that represents a specific participant is called a

lifeline. Each incident is symbolized by a lifeline in a sequence diagram. A sequence diagram's lifeline components are at the very top.

Messages are used to show how items communicate with one another. The messages on the lifeline are displayed in chronological sequence. Messages are represented by arrows. The heart of a sequence diagram is made up of lifelines and messages.

4.3 MODULES

Data Gathering module :

The set of normal and injected inquiries was gathered using Kaggle, an open-source service. This service provides a csv file containing a set of sql injection queries (which includes normal queries, injected queries, and plain text). To get the data, go to pn185090/sql_injection_ds (github.com) However, because the dataset only contained two labels (1 for injected questions, 0 for regular queries), we added a third label (2 for plain text) to the dataset that contains plain text queries. 30000 random queries are taken from this dataset to train the ML models.

To train the ML models, 10000 random normal(legitimate) queries, around 5000 injected queries, and 800 plain queries were obtained from this dataset.

Feature Extraction:

In this project, we have extracted features by using tokenization. We have splitted the SQL queries into tokens and used those tokens to prepare data for training and testing of the machine learning models. Here we have extracted 20 features from the tokens. These all features are basically count based ones from the tokens. Few examples of features are: no.of single quotes, no.of double quotes, no.of one line comments, no.of multiple line comments, number of spaces, normal keywords, malicious keywords, NA values, percentage symbols, alphabets, digits.

These kind of features are extracted from each query and 20 such features are generated from the queries.

SQLi keywords are generally leveraged to conduct actions on database tables. These keywords are critical in launching a SQL injection attack since they do unexpected actions. As a consequence, while utilizing these phrases, it's critical to know the difference between legal and fraudulent requests. The tokenization technique is used to extract tokens from real requests in order to carry out such a procedure. The method of tokenizing a query is to break it down into tokens (keywords). From these generated tokens from the queries, model will extract the attributes . Each question is portrayed as a numerical sequence, with every number corresponding to one of the Tab attributes. In order to detect SQLi attacks, it is necessary to accurately identify these features. The basis for choosing these characteristics is the recognition of most SQIA kinds, including those that are handled similarly to SQL queries, such as stored procedures, redundancies/tautologies, unions, piggybacked, unlawful, and logically incorrect.

Training the Model

After dividing the dataset into train and test data using the two methods described in above module now we will construct a model and train the model using the train dataset. Here in our project we will use some different types of Ensembling machine learning algorithms implementations they are namely , LightGBM, XGBoost ,AdaBoost ,GBM, Random Forest to train our model.

Testing the Model

After the model has been trained, it will be put to the test using test data to determine the model's correctness. We may select the most accurate model out of all models tested based on accuracy so that we can more correctly forecast attacks because we have constructed models utilizing various techniques and strategies for dividing test and train data.

5. IMPLEMENTATION

5.1 LANGUAGE OR TECHNOLOGY USED

The code is written in the Python programming language. Python has a large number of libraries that can be used for scientific and computational purposes. Scikit-learn, pandas, numpy, matplotlib, seaborn, lightgbm, xgboost, and adaboost are examples of libraries. The project's major functionalities are as follows:

1. Data collection and Feature Extraction :

In this section we have gathered the dataset from the internet and cleaning the dataset involves removing the outliers, removing irrelevant columns etc.

2. Model training :

After dividing the cleaned dataset into train and test splits we will use AdaBoost, LightGBM, XGBoost to train the model Testing Model : In this we will test the trained model and find the accuracy so that we can pick best accurate model.

5.2 ALGORITHMS IMPLEMENTED

ENSEMBLE LEARNING :

To tackle a specific computational intelligence problem, ensemble learning is the process of building and combining a large number of models, such as classifiers or experts. Most frequently, ensemble learning is used to assist a group of people perform better (classification, prediction, function approximation, etc.)

LIGHTGBM :

The distributed gradient boosting platform for machine learning known as LightGBM (Light Gradient Boosting Machine) by Microsoft is free and open source. It is used for classification, ranking, and other machine learning applications and is based on decision tree approaches. Performance and scalability are prioritized in the development strategy.

Algorithms including GBT, GBDT, GBRT, GBM, MART, and RF are supported by the LightGBM framework. Sparse optimization, parallel training, numerous loss functions, regularization, bagging, and early pausing are just a few of the advantages LightGBM possesses over XGBoost.

The structure of trees is a distinguishing factor between the two. Unlike the majority of previous implementations, LightGBM does not grow a tree row by row. Instead, it develops trees leaf by leaf, starting from the ground up. It selects the leaf that it thinks will minimize loss the greatest. Additionally, LightGBM does not employ the popular sorted-based decision tree learning algorithm, which looks for the best split point on sorted feature values, unlike XGBoost and other implementations. Instead, LightGBM uses a highly optimized decision tree learning technique based on histograms, which enhances performance and conserves memory. The LightGBM algorithm uses the innovative techniques GOSS (Gradient-Based One-Side Sampling) and EFB (Exclusive Feature Bundling) to run more quickly while retaining exceptional accuracy.On a cluster of 16 PCs, the distributed version of LightGBM can train a CTR predictor on the Criteo dataset, which has 1.7 billion records and 67 characteristics, in just one or two hours. With support for C++, Python, R, and C#, LightGBM runs on Linux, Windows.

Differences from other tree-based algorithms

Other algorithms develop trees horizontally, whereas Light GBM grows them vertically. This difference suggests that Light GBM grows trees leaf by leaf rather than level by level. The diagrams below show LightGBM and various boosting methods.

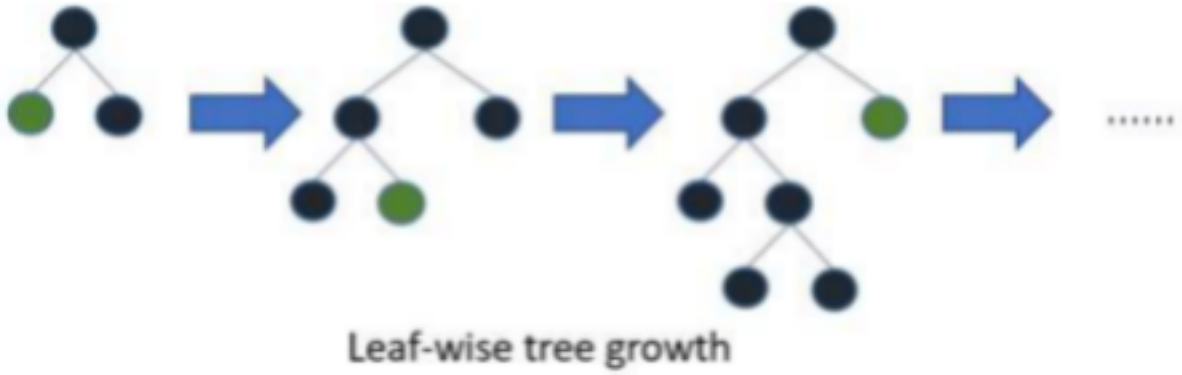


Fig 5.2.1 Tree-growth of LightGBM

Fig 5.2.2 Describes how boosting algorithm works other than LightGBM.

Light GBM getting so much traction :

The exponential growth of data makes it challenging for conventional data science approaches to deliver quicker answers. Due to its quick speed, Light GBM is prefixed by the word "Light". Large amounts of data can be handled by Light GBM while using less memory. The popularity

of Light GBM is also attributed to its emphasis on accuracy. Data scientists are increasingly embracing LGBM to create data science applications since it supports GPU learning.

The LightGBM algorithm
Input:
Training data: D = {(χ1, y1), (χ2, y2), ..., (χN, yN)}, χi ∈χ, χ ⊆ R, yi∈{−1,+1}; loss function: L(y, θ (χ));
Iterations:
M; Big gradient data sampling ratio: a; slight gradient data sampling ratio: b;
1: Combine features that are mutually exclusive (i.e., features never simultaneously accept nonzero values) of χi, i = {1, ...,N} by the exclusive feature bundling (EFB) technique;
2: Set $\theta_0(\chi) = arg\ min_c \sum_i^N L(y_i, c)$;
3: For m = 1 to M do
4: Calculate gradient absolute values:

$$r_i = \left| \frac{\partial L(y_i, \theta(x_i))}{\partial \theta(x_i)} \right|_{\theta(x)=\theta_{m-1(x)}}, i = \{1, \dots, N\}$$

5: Resample data set using gradient-based one-side sampling (GOSS) process:
$topN = a \times len(D)$; $randN = b \times len(D)$;
$sorted = GetSortedIndices(abs(r))$;
$A = sorted[\ 1 : topN]$; $B = RandomPick(sorted[\ topN : len(D)]\ , randN)$;
$D = A + B$;
6: Calculate information gains:

$$V_j(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_l} r_i + \frac{1-a}{b}\sum_{x_i \in B_l} r_i\right)^2}{n_l^j(d)} + \frac{\left(\sum_{x_i \in A_r} r_i + \frac{1-a}{b}\sum_{x_i \in B_r} r_i\right)^2}{n_r^j(d)} \right)$$

7: Develop a new decision tree $\theta_m(x)'$ on set D'
8: Update $\theta_m(\chi) = \theta_{m-1}(\chi) + \theta_m(\chi)$
9: End for
10: Return $\tilde{\theta}(x) = \theta_M(x)$

Fig 5.2.3 LightGBM algorithm

Benefits of Light GBM

1. Speedier training: LGBM uses a bar graph approach to categorize and arrange continuous features into discrete bins, which completes training process fast.

2. Memory consumption is reduced: Continuous values are replaced with discrete bins, resulting in a reduction in memory usage.

3. It produces noticeably more detailed trees utilizing a leaf-wise split strategy as opposed to a level-wise split method, which is the key to getting greater accuracy. It exceeds every other boosting technique in terms of accuracy. Nevertheless, it could cause overfitting, eventually which can be prevented by boosting the maximum depth choice.

4. Large-Scale Dataset Compatibility: When compared to XGBOOST, it can handle large datasets just as well while taking substantially less time to train.

Applications for LightGBM

The following major difficulties are best solved using LightGBM:

1. To categorize binary data, the log loss objective function is utilized.

2. Regression model based on the L2 loss

3. There are a lot of different ways to categorize things.

4. Cross-entropy is calculated using the log loss objective function.

5. LambdaRank employing lambdarank and NDCG as the goal function.

LightGBM is recognized as a very speedy algorithm and the most commonly utilized method in

machine learning when it comes to obtaining swift and high-accuracy outcomes. There are almost 100 settings to choose from in the LightGBM handbook.

XGBOOST :

XGBoost is a popular ensembling machine learning algorithm. It is a tree based ensembling algorithm which works on the basis of boosting technique. This algorithm is actually developed as a research project. This algorithm is used in many industry applications and also gained lot of popularity as a key algorithm for winning in the data science competetions.

It has lately received a lot of coverage and attention as the preferred algorithm for many successful machine learning teams.

Some of the ways in which the algorithm distinguishes itself are as follows:

1. Multiple applications: This tool can be used to tackle regression, classification, ranking, and user-defined prediction problems.

2. Adaptability: It works well on Windows, Linux, and Mac OS X.

3. Languages: C++, Python, R, Java, Scala, and Julia are among the programming languages supported.

4. Integration with the Cloud: Supports clusters on AWS, Azure, and Yarn, as well as Flink, Spark, and other ecosystems.

XGBoost effectiveness

Ensembling tree based algorithms like LGBM, GBM and XGBoost use gradient descent algorithm to strengthen the underperforming trees. In another way, by optimizing the system and enhancing the algorithms, XGBoost enhances the underlying GBM framework.

Optimization of System:

1. Parallelization: XGBoost makes use of a parallelized approach to manage the sequential tree construction. This is possible because the loops used to construct base learners can be utilized interchangeably; the outermost loop counts the tree leaf nodes, while the innermost loop computes the features. Because the inner one—the more computationally intensive of the two—must be finished before the outer loop can be started, parallelization is constrained. So, to

decrease runtime, the order of the loops is changed utilizing parallel thread sorting and initialization via a global scan of all instances. By accounting for compute parallelization overheads, this option increases algorithmic speed.

2. Optimization of Hardware: To maximize the use of the hardware resources at hand, this approach was devised. This is accomplished by cache awareness, which mandates that each thread build internal buffers to store gradient statistics. For instance, out-of-core computing makes the most of storage capacity while processing big data sets that don't fit in memory.

Algorithm 1: XGboost algorithm

Data: Dataset and hyperparameters
Initialize $f_0(x)$;
for $k = 1, 2, \ldots, M$ **do**
  Calculate $g_k = \frac{\partial L(y,f)}{\partial f}$;
  Calculate $h_k = \frac{\partial^2 L(y,f)}{\partial f^2}$;
  Determine the structure by choosing splits with maximized gain
  $\mathbf{A} = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right]$;
  Determine the leaf weights $w^* = -\frac{G}{H}$;
  Determine the base learner $\hat{b}(x) = \sum_{j=1}^{T} wI$;
  Add trees $f_k(x) = f_{k-1}(x) + \hat{b}(x)$;
end
Result: $f(x) = \sum_{k=0}^{M} f_k(x)$

Fig 5.2.5 XGBoost algorithm

5.3 CODE :

In this project, we have implemented three types of ensemble machine techniques like XGBoost, AdaBoost, Random Forest, LGBM.

Initially we are importing all the necessary packages and then loading the data from the dataset.

```
#importing basic packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Loading the data
url='https://raw.githubusercontent.com/pn185090/sql_injection_ds/main/SqlQueriesData.csv'
data0 = pd.read_csv(url,encoding= 'unicode_escape')

# Sepratating & assigning features and target columns to X & y
y = data['Label']
X = data.drop('Label',axis=1)
X.shape, y.shape

# Splitting the dataset into train and test sets: 70-30 split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 12)
X_train.shape, X_test.shape
#importing packages
from sklearn.metrics import accuracy_score
from yellowbrick.classifier import classification_report
from yellowbrick.classifier.rocauc import roc_auc
# Creating holders to store the model performance results
ML_Model = []
acc_train = []
acc_test = []

#function to call for storing the results
def storeResults(model, a,b):
  ML_Model.append(model)
  acc_train.append(round(a, 3))
  acc_test.append(round(b, 3))
```

```
# Decision Tree model
from sklearn.tree import DecisionTreeClassifier
# instantiate the model
tree = DecisionTreeClassifier(max_depth = 5)
# fit the model
tree.fit(X_train, y_train)

#predicting the target value from the model for the samples
y_test_tree = tree.predict(X_test)
y_train_tree = tree.predict(X_train)

"""**Performance Evaluation:**"""

#computing the accuracy of the model performance
acc_train_tree = accuracy_score(y_train,y_train_tree)
acc_test_tree = accuracy_score(y_test,y_test_tree)

print("Decision Tree: Accuracy on training Data: {:.7f}".format(acc_train_tree))
print("Decision Tree: Accuracy on test Data: {:.7f}".format(acc_test_tree))

"""**Storing the results:**"""

visualizer = classification_report(
    tree, X_train, y_train, X_test, y_test, classes=[0,1], support=True
)
roc_auc(tree, X_train, y_train, X_test=X_test, y_test=y_test, classes=[0,1])

visualizer = classification_report(
    tree, X_train, y_train, X_test, y_test, classes=[0,1], support=True
)

roc_auc(tree, X_train, y_train, X_test=X_test, y_test=y_test, classes=[0,1])
```

```
# Random Forest model
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest = RandomForestClassifier(max_depth=5)

# fit the model
forest.fit(X_train, y_train)

#predicting the target value from the model for the samples
y_test_forest = forest.predict(X_test)
y_train_forest = forest.predict(X_train)

"""**Performance Evaluation:**"""

#computing the accuracy of the model performance
acc_train_forest = accuracy_score(y_train,y_train_forest)
acc_test_forest = accuracy_score(y_test,y_test_forest)

print("Random forest: Accuracy on training Data: {:.7f}".format(acc_train_forest))
print("Random forest: Accuracy on test Data: {:.7f}".format(acc_test_forest))

visualizer = classification_report(
    forest, X_train, y_train, X_test, y_test, classes=[0,1], support=True
)
roc_auc(forest, X_train, y_train, X_test=X_test, y_test=y_test, classes=[0,1]

"""**Storing the results:**"""

#storing the results. The below mentioned order of parameter passing is important.
#Caution: Execute only once to avoid duplications.
storeResults('Random Forest', acc_train_forest, acc_test_forest)
```

```
from xgboost import XGBClassifier

# instantiate the model
xgb = XGBClassifier(learning_rate=0.4,max_depth=7)
#fit the model
xgb.fit(X_train, y_train)

#predicting the target value from the model for the samples
y_test_xgb = xgb.predict(X_test)
y_train_xgb = xgb.predict(X_train)

"""**Performance Evaluation:**"""

#computing the accuracy of the model performance
acc_train_xgb = accuracy_score(y_train,y_train_xgb)
acc_test_xgb = accuracy_score(y_test,y_test_xgb)

print("XGBoost: Accuracy on training Data: {:.7f}".format(acc_train_xgb))
print("XGBoost : Accuracy on test Data: {:.7f}".format(acc_test_xgb))

visualizer = classification_report(
    xgb, X_train, y_train, X_test, y_test, classes=[0,1], support=True
)
roc_auc(xgb, X_train, y_train, X_test=X_test, y_test=y_test, classes=[0,1])

"""**Storing the results:**"""
storeResults('XGBoost', acc_train_xgb, acc_test_xgb)

#Support vector machine model
from sklearn.svm import SVC

# instantiate the model
svm = SVC(kernel='linear', C=1.0, random_state=12)
#fit the model
svm.fit(X_train, y_train)
```

```
from sklearn.ensemble import AdaBoostClassifier
# Create adaboost classifer object
Adaboost = AdaBoostClassifier(n_estimators=100,learning_rate=0.9)
# Train Adaboost Classifer
Adaboost= Adaboost.fit(X_train, y_train)

#Predict the response for test dataset
y_test_Adaboost = Adaboost.predict(X_test)
y_train_Adaboost = Adaboost.predict(X_train)

"""**Performance Evaluation:**"""

#computing the accuracy of the model performance
acc_train_Adaboost = accuracy_score(y_train,y_train_Adaboost)
acc_test_Adaboost = accuracy_score(y_test,y_test_Adaboost)

print("AdaBoost: Accuracy on training Data: {:.7f}".format(acc_train_Adaboost))
print("AdaBoost : Accuracy on test Data: {:.7f}".format(acc_test_Adaboost))

"""**Storing the results:**"""

visualizer = classification_report(
    Adaboost, X_train, y_train, X_test, y_test, classes=[0,1], support=True
)
roc_auc(Adaboost, X_train, y_train, X_test=X_test, y_test=y_test, classes=[0,1])

#storing the results. The below mentioned order of parameter passing is important.
#Caution: Execute only once to avoid duplications.
storeResults('Adaboost', acc_train_Adaboost, acc_test_Adaboost)

from lightgbm import LGBMClassifier
lgbm = LGBMClassifier()
lgbm.fit(X_train, y_train)

#Predict the response for test dataset
y_test_lgbm = lgbm.predict(X_test)
```

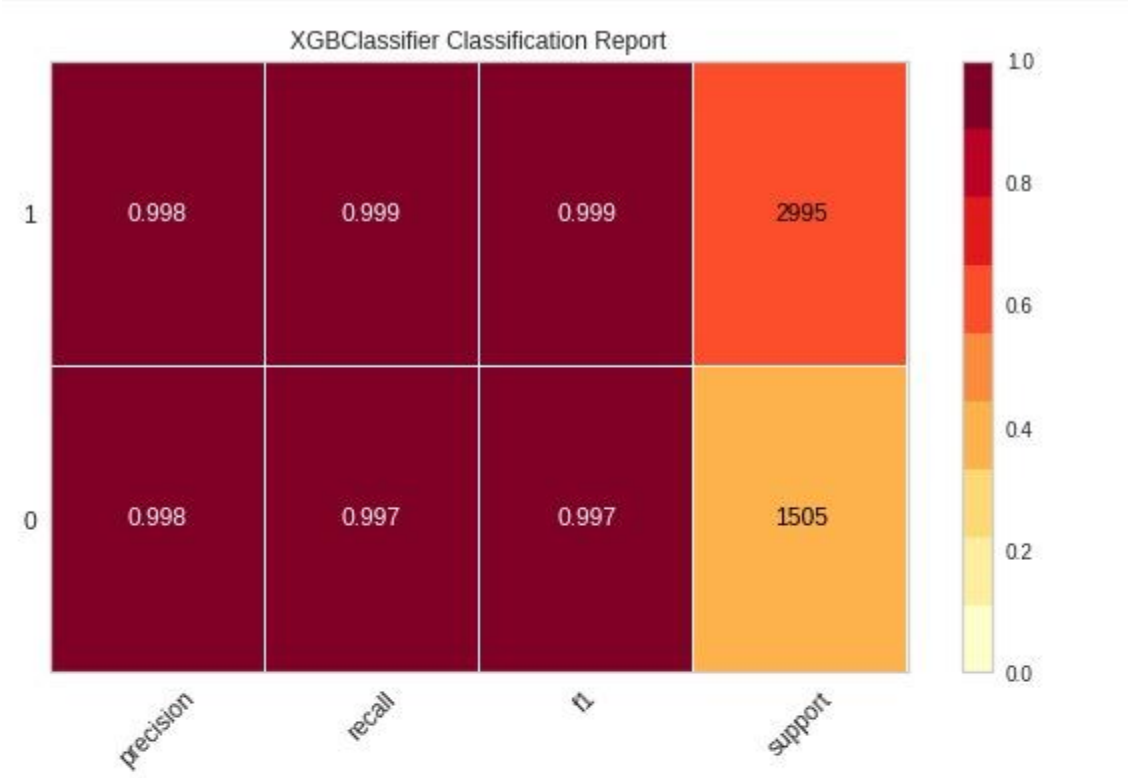
```
from sklearn.ensemble import GradientBoostingClassifier
# fit the model on the whole dataset
Gbm = GradientBoostingClassifier()
Gbm.fit(X_train, y_train)

#Predict the response for test dataset
y_test_Gbm = Gbm.predict(X_test)
y_train_Gbm = Gbm.predict(X_train)

"""**Performance Evaluation:**"""

#computing the accuracy of the model performance
acc_train_Gbm = accuracy_score(y_train,y_train_Gbm)
acc_test_Gbm = accuracy_score(y_test,y_test_Gbm)

print("Gbm: Accuracy on training Data: {:.7f}".format(acc_train_Gbm))
print("Gbm : Accuracy on test Data: {:.7f}".format(acc_test_Gbm))

"""**Storing the results:**"""

visualizer = classification_report(
    Gbm, X_train, y_train, X_test, y_test, classes=[0,1], support=True
)
roc_auc(Gbm, X_train, y_train, X_test=X_test, y_test=y_test, classes=[0,1])

#storing the results. The below mentioned order of parameter passing is important.
#Caution: Execute only once to avoid duplications.
storeResults('Gradient Boosting', acc_train_Gbm, acc_test_Gbm)

#creating dataframe
results = pd.DataFrame({ 'ML Model': ML_Model,
    'Train Accuracy': acc_train,
    'Test Accuracy': acc_test})
results
```

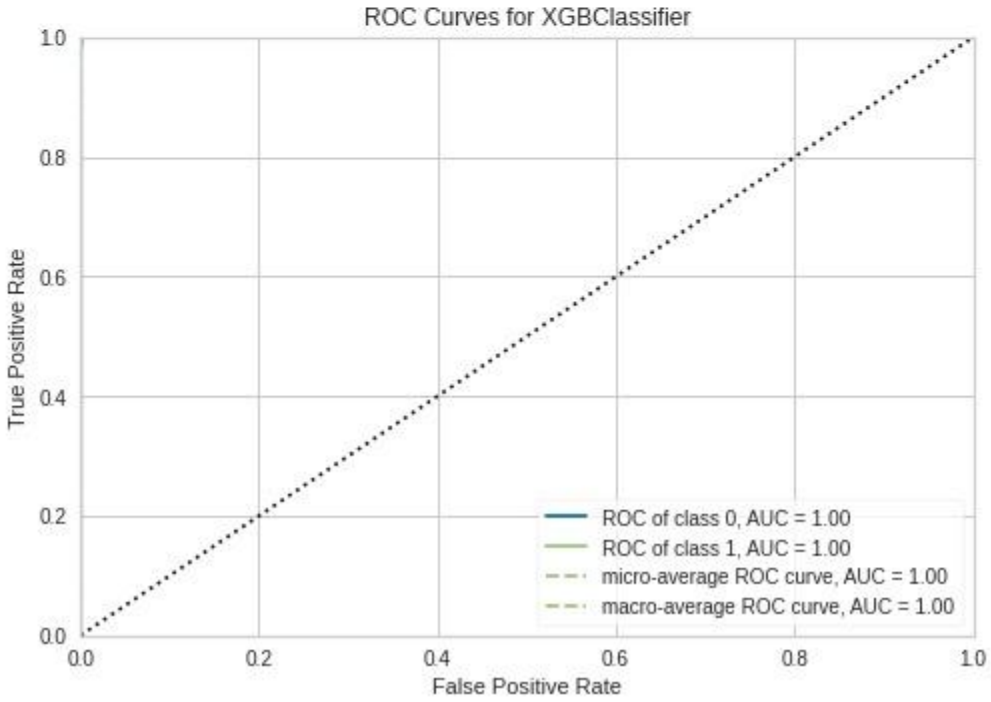
6. RESULTS

To compare the performances of models, we have used classification report and ROC curves. As this is a classification problem, just comparing the models' performance with accuracy is not sufficient.

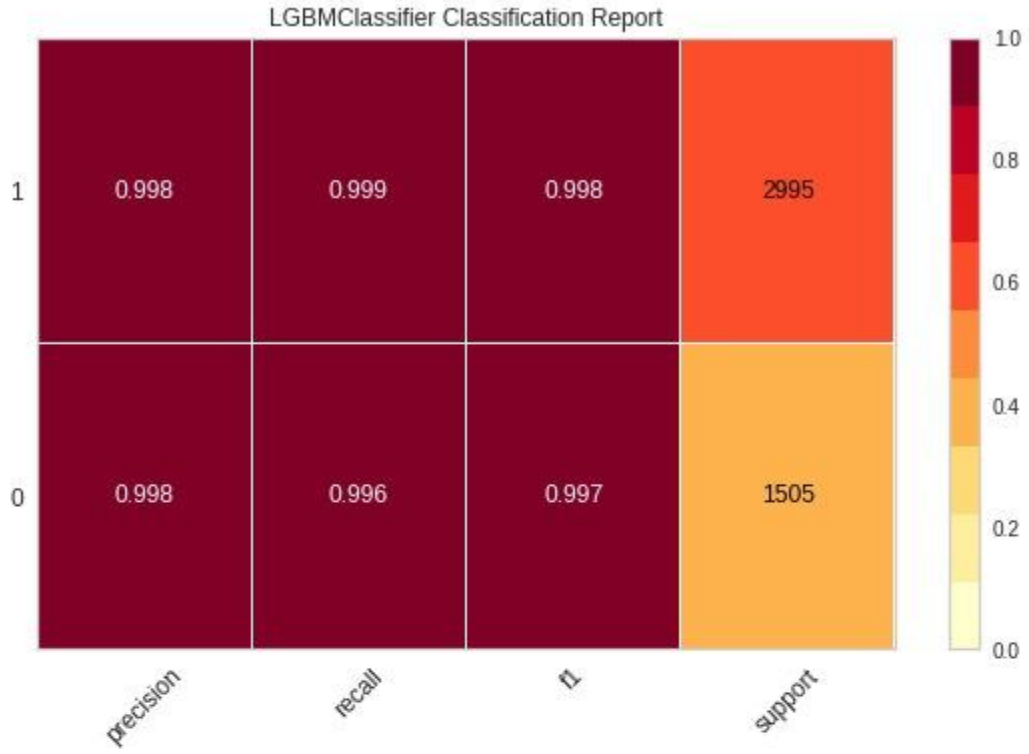
6.1 XGBOOST MODEL:

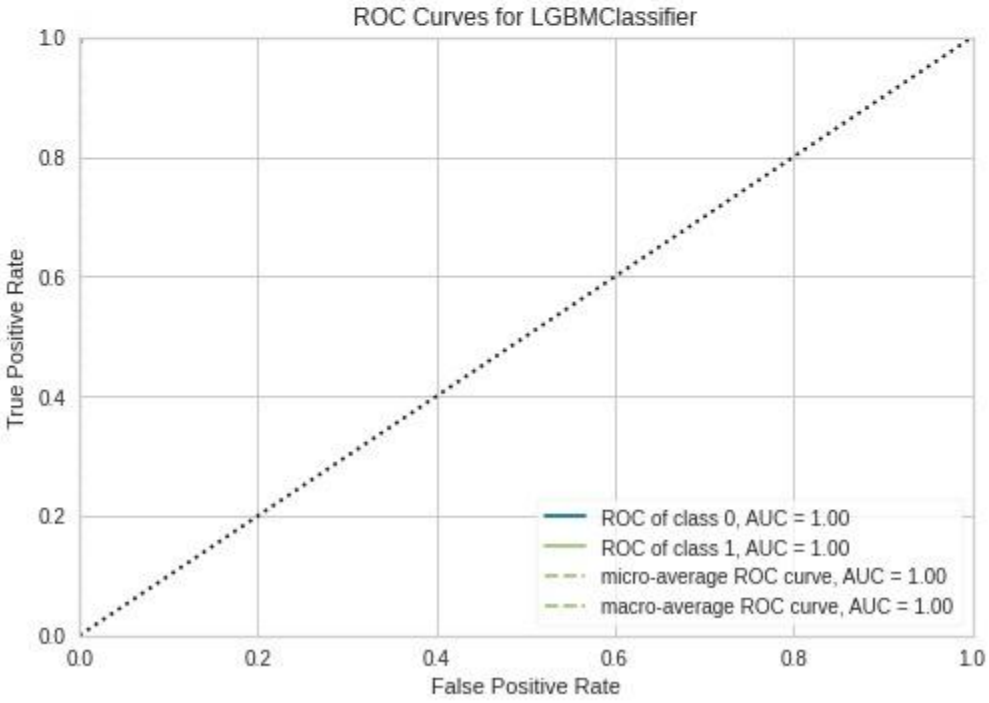
For XGBoost model, we have achieved an f1 score of 0.99 for both the classes.





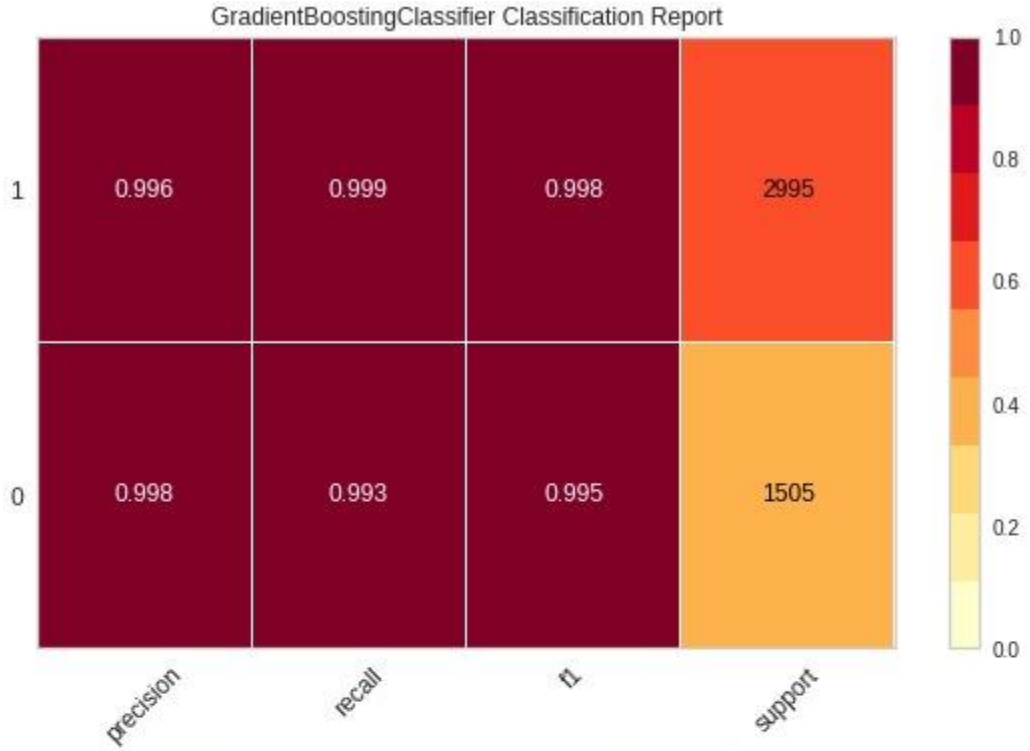
6.2 LGBM:



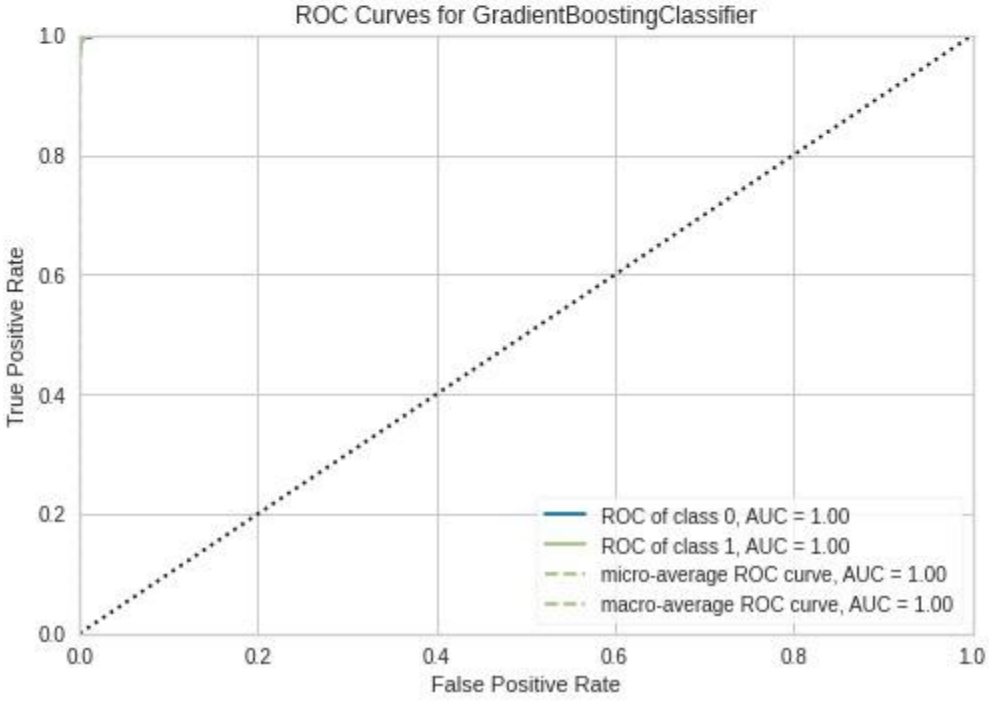



```
ROCAUC(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f96f059c850>,
       classes=[0, 1], estimator=LGBMClassifier())
```

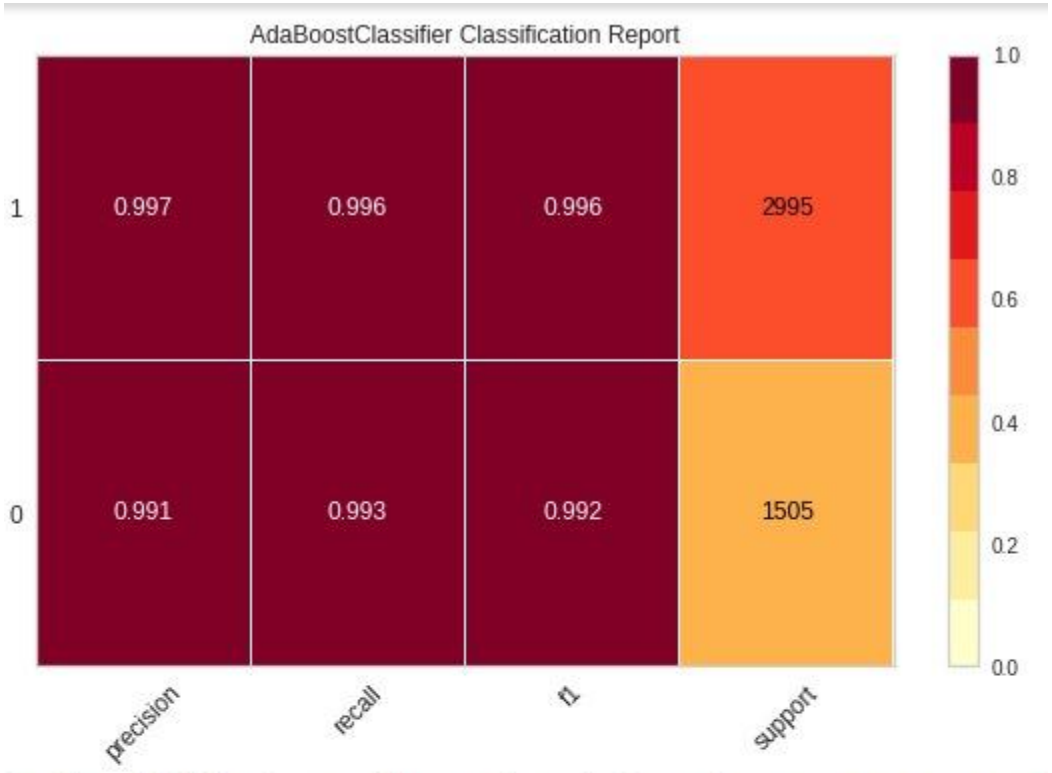
6.3 GBM:



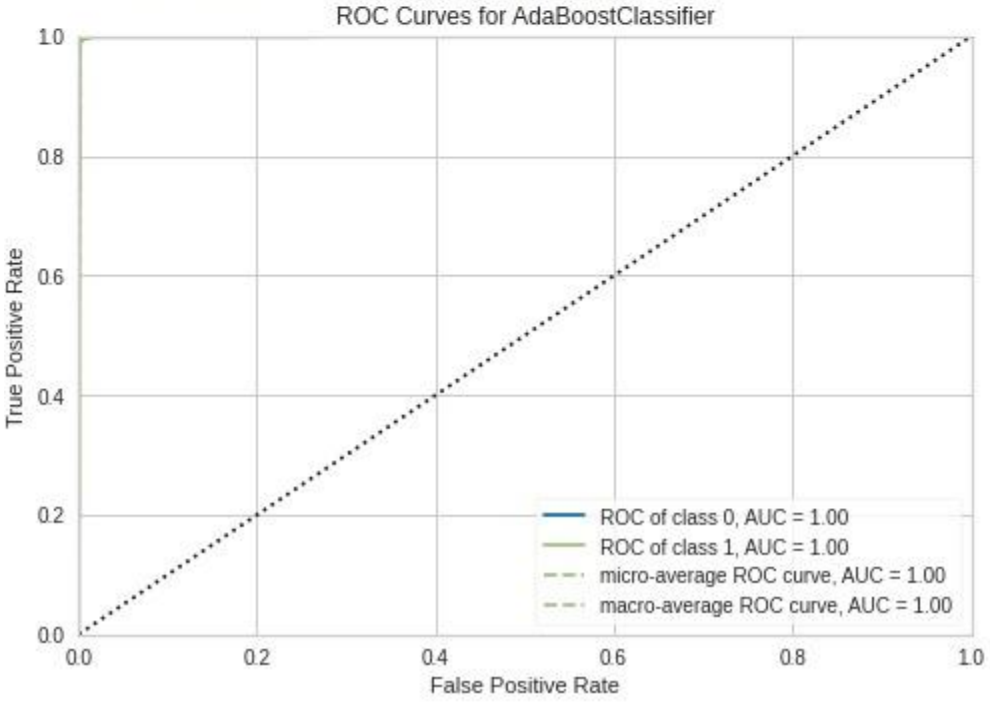

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X
  "X does not have valid feature names, but"
```



6.4 AdaBoost




```
'usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarni
  "X does not have valid feature names, but"
```



Comparison of Model Accuracy Summary:

	ML Model	Train Accuracy	Test Accuracy
2	XGBoost	1.000	0.998
6	LightGBM	1.000	0.998
7	Gradient Boosting	0.997	0.997
5	Adaboost	0.996	0.995
1	Random Forest	0.984	0.986
4	SVM	0.978	0.982
0	Decision Tree	0.977	0.977
3	AutoEncoder	0.796	0.802

7. CONCLUSION AND FUTURE SCOPE

CONCLUSION :

We obtained superior accuracy and F1 scores using the ensembling machine learning algorithms, namely XGBoost, Light GBM, and AdaBoost, with a 70 percent train and 30 percent test dataset split, as compared to other machine learning methods.

FUTURE SCOPE :

We'll be measuring several web-based application codes in the public domain as part of a future study in order to obtain high accuracy in SQL injection detection methods. Many web-based dangers will be detected using SQL integration with nikto HTTP scanner, HTTP scanning proxies, and metasploit. In this case, CNN is being utilized for future studies and to construct a detection model; however, we can improve the model by using various algorithms and finding the optimal model. We'll employ the static analysis method here, and we'll add run-time analysis as a future project. We have simply established a detection system for SQL Injection, but we can create a prevention mechanism in the future.

BIBLIOGRAPHY

REFERENCES :

[1] Sonali Mishra , “SQL Injection Detection using Machine Learning ”, from https: //scholarworks.sjsu.edu/cgi/ viewcontent.cgi?arti c le=1727context=etdprojects, on23May2019 pp.10 - 29.

[2] Bojken Shehu and Aleksander Xhuvani ,”A LiteratureReviewandComparativeAnalyseson SQL Injection: Vulnerabilities, Attacks and their Prevention and Detection Techniques”from https :// pdfs.semanticscholar.org,Vol.11,Issue4,No 1,July 2014 pp 20 - 34.

[3] SuhaimiIbrahim,”SQLInjectionDetectionand Prevention Techniques” from https ://pdfs.semanticscholar.org/ Volume 3, Number 7, August 2011 , pp 85 - 89.

[4] G.Wassermann,Z.Su,“Ananalysisframeworkfor security in web applications,” In: Proceedings of the FSE Workshop on Specificationand Verification of Component-Based Systems, fromhttps://link.springer.com/chapter/1 0.1007/978 −0 −387 − 44599−15 SAVCBS, pp. 70–78, 2004.

[5] Mei Junjin, “An Approach for SQL Injection Vulnerability Detection,” Proceedings. of the 6th Int. Conf. on InformationTechnology: NewGenerations,LasVegas,Nevada,pp.14-19, Apr. 2009.

[6] V.Haldar, D.Chandra, and M.Franz, "Dynamic Taint Propagation for Java," Proc. 2 1s t Annual Computer Security Applications Conference, Dec 2005.

[7] S.W.Boyd and AD.Keromytis, "SQLrand: Preventing SQL Injection Attacks," Proc. the 2nd Applied Cryptography and Network Security (ACNS) Conference, pp. 292-302, Jun 2004.

[8] G.T.Buehrer, RW.Weide, and P.AG.Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks," International Workshop on Software Engineering and Middleware (SEM), 2005.

[9] Evans Dogbe, Richard Millham, Prenitha Singh “A Combined Approach to Prevent SQL Injection Attacks,” Science and Information Conference 2013 October 7-9, 2013, London, UK.

[10] Ryohei Komiya, Incheon Paik, Masayuki Hisada, “Classification of Malicious Web Code by Machine Learning,” Awareness Science and Technology (iCAST), 2011 3rd International Conference on , vol., no., pp.406,411, 27-30 Sept. 2011.

[11] Z.Su and G.Wassermann "The Essence of Command Injection Attacks in Web Applications," The 33rd Annual Symposium on Principles of Programming Language (POPL 2006), Jan 2006. [8] The Open Web Application Security Project (OW ASP). The Ten Most Critical Web Application Security Risks 20 10. https://www.owasp.org/ index.php/Top_10_2013.