API Security Analysis

Team Member: Ke Xu, Ruiyang Liu, Tianze Ran, Yuxuan Zhao

Note: This file contains general writeup about our project. For detail implementation and explanation, please refer to the 2 Jupyter notebook. The first one contains data preparation, and first 3 models: Bayesian Ridge Regression, SVM & GMM. The second one contains the Autoencoder (we separate the notebook for consideration of dependency).

Overview

1.Motivation

Nowadays businesses use APIs to transfer information between each other. Unprotected API will lead to data breach for business in the process of transmitting data.

In Cyber Security, API security is important to protect APIs from attacks. The detection of API activities is thus worth researching. If we can detect the unusual activity precisely, we can make preventive actions to avoid huge loss.

2.Data Source & Description

We'll be using a dataset from the Kaggle. Go to https://www.kaggle.com/datasets/tangodelta/api-access-behaviour-anomaly-dataset

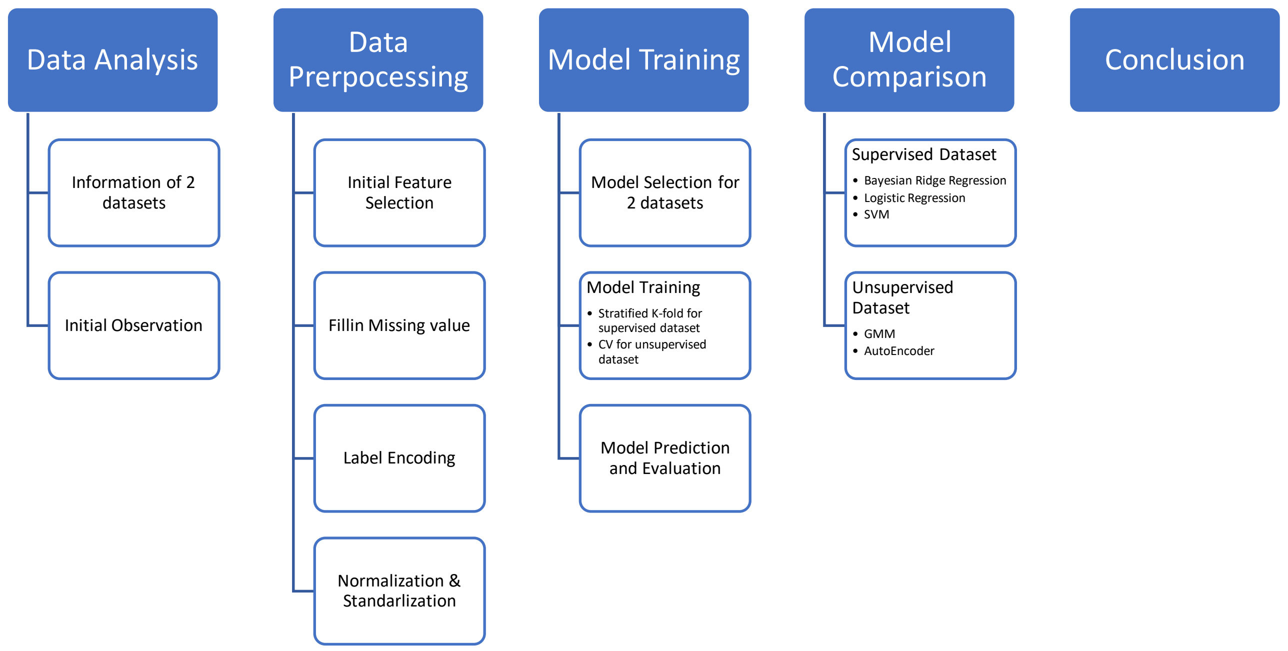
This API security dataset captures API access patterns in terms of behavior metrics.

There are 2 datasets available. The one called supervised_dataset.csv has behaviors labeled as normal or outlier by human. The second file called remaining_behavior_ext.csv has a larger number of samples that are not labeled but has additional columns. It also has a classification created by another algorithm called ALGO-X. Each row is one instance of an observed behavior that has been manually classified as normal or outlier.

3.Problem Definition

We want to find which APIs are outliers(malicious) based on the 2 datasets.

System Approach

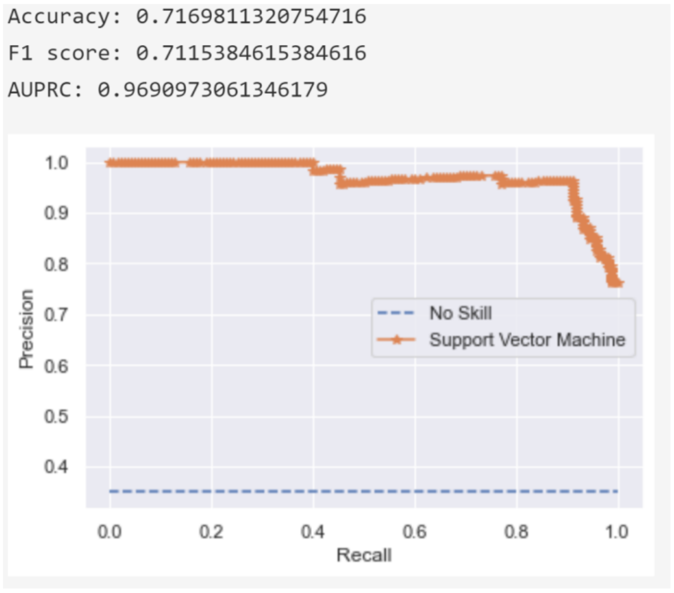

Data Analysis
Information of 2 datasets
Initial Observation
Data Perpocessing
Initial Feature Selection
Fillin Missing value
Label Encoding
Normalization & Standarlization
Model Training
Model Selection for 2 datasets
Model Training
• Stratified K-fold for supervised dataset
• CV for unsupervised dataset
Model Prediction and Evaluation
Model Comparison
Supervised Dataset
• Bayesian Ridge Regression
• Logistic Regression
• SVM
Unsupervised Dataset
• GMM
• AutoEncoder
Conclusion

Step by step illustration

Method for each step
Please refer to the two Jupyter Notebooks for step-by-step illustration.

Comparison between different models and Conclusion

Labeled Dataset (Supervised Learning)
The 2 models (Bayesian Ridge Regression+ SVM, Logistic Regression+SVM) all use 9 features, Stratified K-fold=4.
Bayesian Ridge Regression + SVM: Regression Model performs bad on supervised_dataset with SVM.
Final output for the first combination:
```
Accuracy: 0.717
F1 score: 0.712
AUPRC: 0.969
```
SVM Precision-Recall curve:

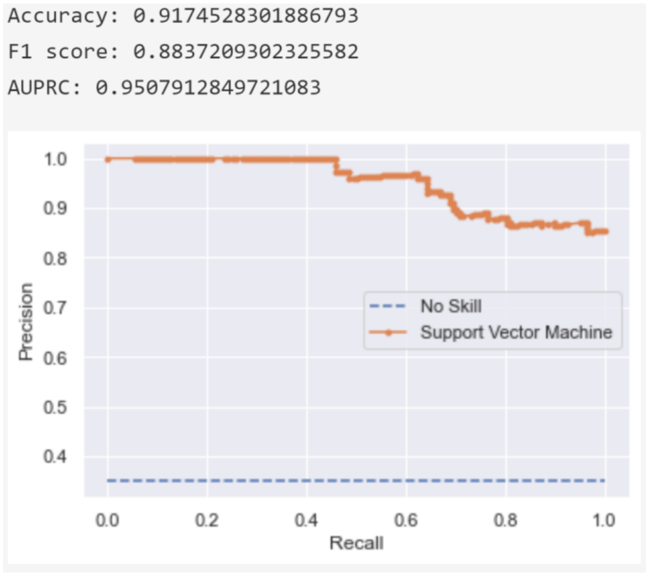


```
Accuracy: 0.7169811320754716
F1 score: 0.7115384615384616
AUPRC: 0.9690973061346179
```

Logistic Regression + SVM: Accuracy & F1 score are around 0.71, which performs better than the first combination.
Final output for the second combination:

```
Accuracy: 0.917
F1 score: 0.884
AUPRC: 0.951
```

SVM Precision-Recall curve:



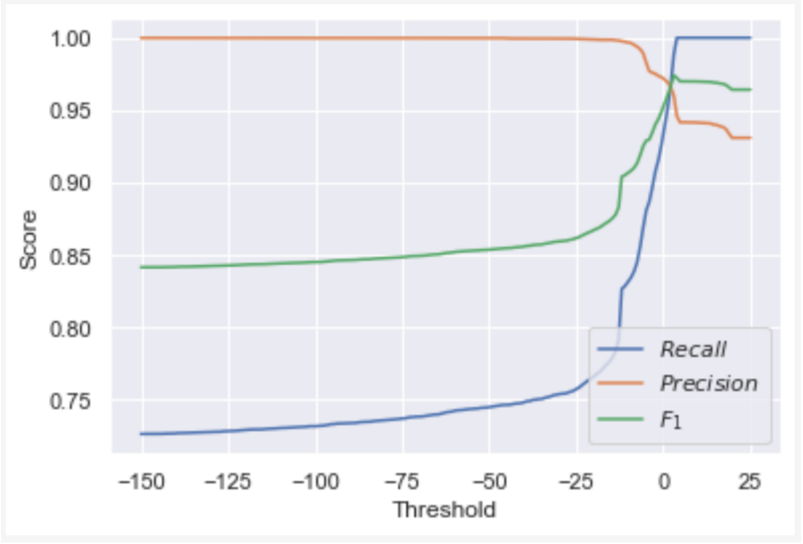
```
Accuracy: 0.9174528301886793
F1 score: 0.8837209302325582
AUPRC: 0.9507912849721083
```

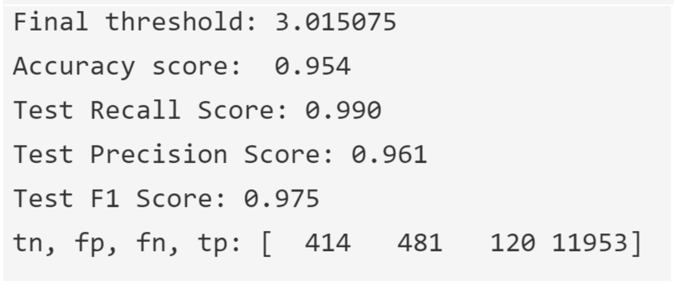
Unlabeled Dataset (Unsupervised Learning)
The 2 models (GMM, AutoEncoder) all use 9 features, 3-way cross validation with data split=[Training=80% normal, Validation=10% normal+50% abnormal, Test=10% normal + 50% abnormal]

GMM: Performs well on the unlabeled dataset. Best suits the API dataset.
Final Evaluation Metrics:

```
Accuracy score: 0.954
Recall = 0.990
Precision = 0.961
F1 Score = 0.975
```

Precision-recall and F1 curve among different thresholds and final test results table for GMM:





```
Final threshold: 3.015075
Accuracy score:  0.954
Test Recall Score: 0.990
Test Precision Score: 0.961
Test F1 Score: 0.975
tn, fp, fn, tp: [  414   481   120 11953]
```

AutoEncoder: Worse and slower than the GMM model.
Final result:

```
Accuracy = 0.815
Precision = 0.922
Recall = 0.875
F1 = 0.898
```