UNIVERSITÉ
BISHOP'S
UNIVERSITY

Forecasting Zero-Day vulnerabilities using Machine Learning

Research project

Fall 2022

Jon Pilarte

1 Introduction

A zero-day is a computer-software vulnerability previously unknown to those who should be interested in its mitigation, like the vendor of the target software. Until the vulnerability is mitigated, hackers can exploit it to adversely affect programs, data, additional computers or a network.

The more recently that the vendor has become aware of the vulnerability, the more likely it is that no fix or mitigation has been developed. Once a fix is developed, the chance of the exploit succeeding decreases as more users apply the fix over time. For zero-day exploits, unless the vulnerability is inadvertently fixed, such as by an unrelated update that happens to fix the vulnerability, the probability that a user has applied a vendor-supplied patch that fixes the problem is zero, so the exploit would remain available.

In this project we will be trying to predict if a we are facing a Zero-Day vulnerability or not by looking at some of the code features (such as the size of the code, the characteristics...). Based on this information we will develop a Supervised ML Binary Classifier.

1.1 CVE and NVD

The CVE List was launched by MITRE as a community effort in 1999, and the U.S. National Vulnerability Database (NVD) was launched by the National Institute of Standards and Technology (NIST) in 2005.

- CVE - A list of records—each containing an identification number, a description, and at least one public reference—for publicly known cybersecurity vulnerabilities. CVE Records are used in numerous cybersecurity products and services from around the world, including NVD.
- NVD - A vulnerability database built upon and fully synchronized with the CVE List so that any updates to CVE appear immediately in NVD.
- Relationship – The CVE List feeds NVD, which then builds upon the information included in CVE Records to provide enhanced information for each record such as fix information, severity scores, and impact ratings. As part of its enhanced information, NVD also provides advanced searching features such as by OS; by vendor name, product name, and/or version number; and by vulnerability type, severity, related exploit range, and impact. While separate, both CVE and NVD are sponsored by the U.S. Department of Homeland Security (DHS) Cybersecurity and Infrastructure Security Agency (CISA), and both are available to the public and free to use.

2 Dataset and features

For this project we are going to be using a Zero-Day vulnerability database that can be found in Kaggle. The first thing we should look into when starting a new Machine Learning project is the data dimensionality. As we can see once importing the data with Pandas library, the data has 216352 records $\times$ 58 features. From the computational point of view training models will be very costly, as we have to think about the cost as the number of records times the number of features. In order to evaluate the most adequate way of reducing the data, we must first see how the output class is distributed.

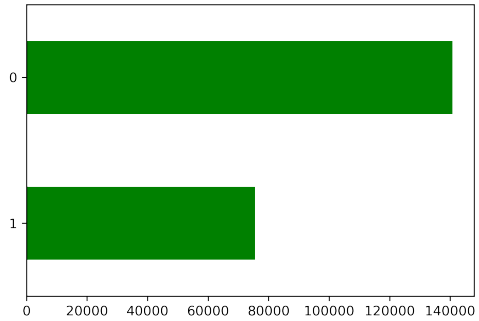


Figure 1: Class imbalance
Class 0 non-vulnerabilities
Class 1 vulnerabilities

As we can see using this plot, there are 2 output classes (0 for non-vulnerabilities and 1 for vulnerabilities). As we could expect, we have many fewer records for the vulnerabilities class. This can create a big problem when trying to create a Binary Classifier, as the results will be biased towards the most frequent class. There are different ways of facing this problem, we could try to optimize the model looking for the highest "f1-score" instead of "accuracy" as the mathematical formula for this score punishes the low results in any of both classes.

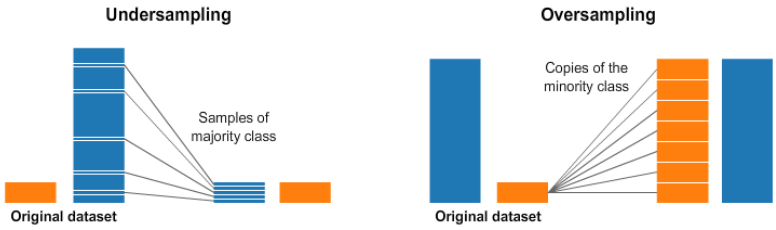
$$F1\ Score = 2 \times \frac{recall \times precision}{recall + precision}$$

Figure 2: F1-Score formula

Even if we use this metric for optimizing the results we are going to be suffering of very biased results, therefore we will need to balance the dataset as we are goona see in the next section.

3 Undersampling and Oversampling

When facing imbalanced datasets we can opt out for 2 different solutions, undersampling or oversampling. These both methods can be applied with the Imbalanced-Learn Python library. Undersampling consists of reducing the number of records of the most frequent class until both classes have the same number of records (75503). There are different methods to choose which records we will be eliminating from the most popular class, but for this exercise we will take the RandomUnderSampler() which randomly selects a number of records from the 0-class until both are balanced. On the other hand, we can take SMOTE Oversampler which uses the k-nearest neighbour algorithm to create synthetic data out ouf the Vulnerabilities class until balancing both out with 140849 records.



(a) Undersampling vs. Oversampling

(b) Imbalanced-Learn

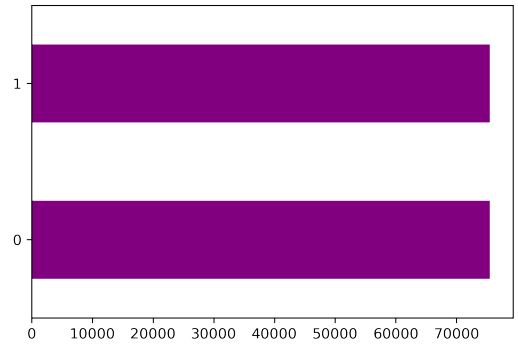


Figure 3: After Undersampling

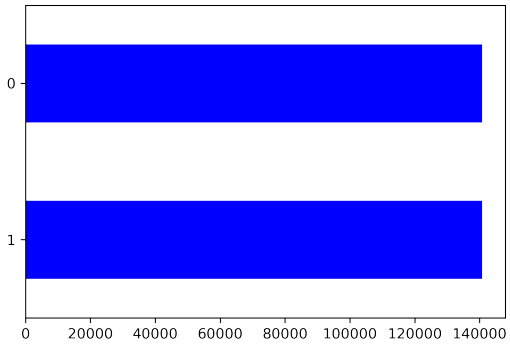


Figure 4: After Oversampling

4 Principal Component Analysis for reducing the number of features

In this section, we will be analysing the different features and how these are distributed. For this purpose, we will need to plot all the existing features in a 57x57 matrix (we take out the "legitimate" output column). Thanks to this graph, we can see that there are highly correlated columns, which will be the ones taking out first when using PCA dimensionaly reductor. Positive correlation means that these are directly correlated, negative correlation means inverse correlation and near 0 results means no correlation between these 2 variables.

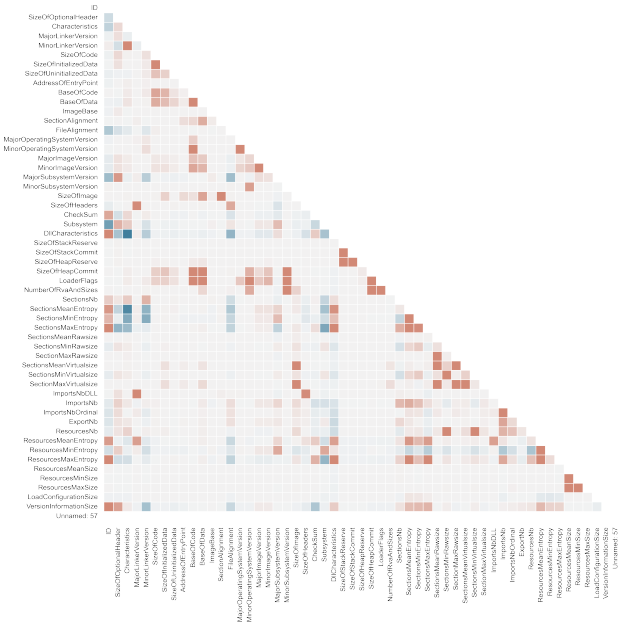


Figure 5: Correlation matrix

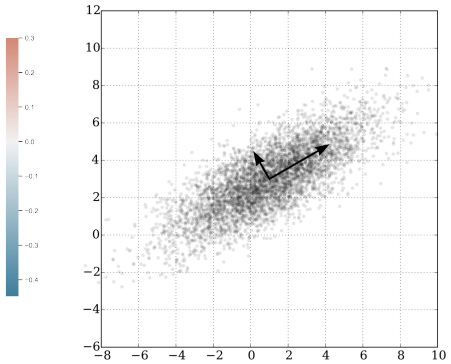


Figure 6: Principal Component Analysis

As we have been seen before in the data dimensionality we have over 50 features which is a big disadvantage at the time of training models as with these number of colums the process is gonna be very resource-consuming. For that reason, we will be using the Principal Component Analysis dimensionality reductor for taking out the less representing columns (the features that aport the least information). This is accomplished by linearly transforming the data into a new coordinate system where (most of) the variation in the data can be described with fewer dimensions than the initial data.

5 First classification models

If we look at the overall view of the Scikit-Learn Machine Learning module we can find out that as we are working on a Binary Classifier with over 100k samples some of the Classification models won't be working properly.

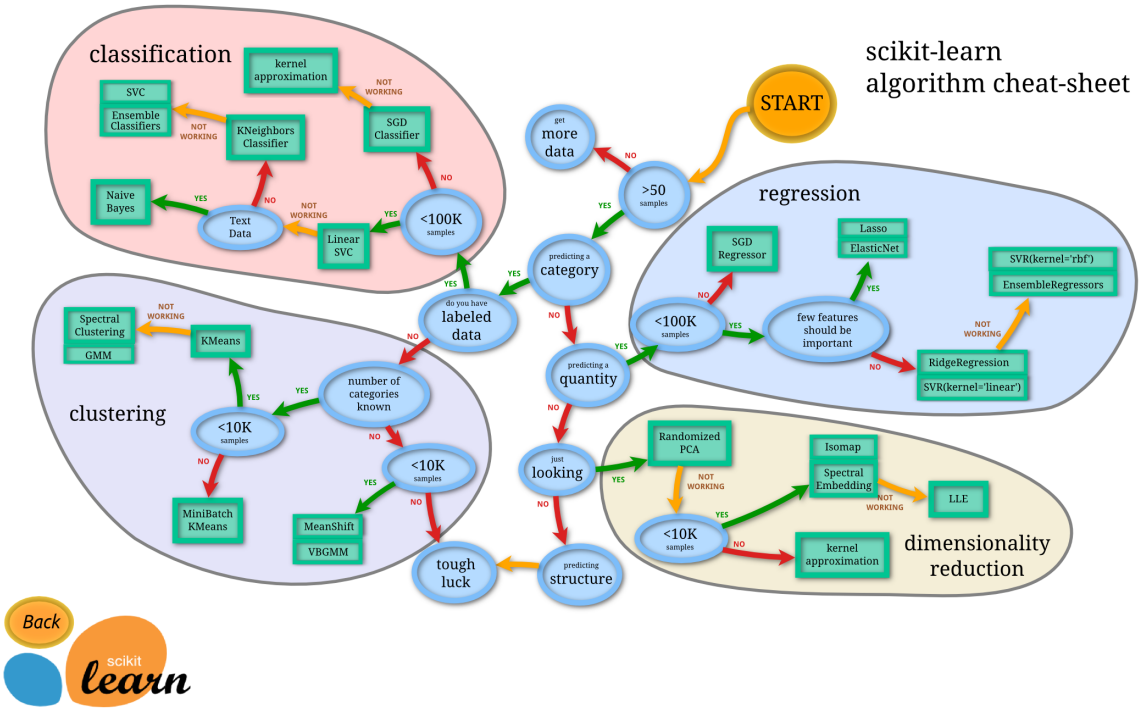


Figure 7: Scikit-Learn map for Machine Learning

To create our first approach we are gonna use a Stratified Cross Validation of 5 folds. We are choosing only 5 fold Cross Validation as the number of records is very high and being a Stratified Cross Validation the number of records in each class will respect the ratio distribution (this won't be really needed as the class imbalance is already solved with oversampling or undersampling). We will also be choosing a 60% learning and 40% testing split, given the high number of records. The first model we are gonna develop is gonna be a default settings Random Forest Classifier (trees=100, criterion=gini, max_depth=None...)

If we check the confusion matrix of this Random Forest we will see that we are getting over 99% accuracy and the errors are not biased. For this example we were using SMOTE Oversampler (so we had almost 140k records in each class) and we didn't use PCA Dimensionality Reductor (we used 57 features to train our model).

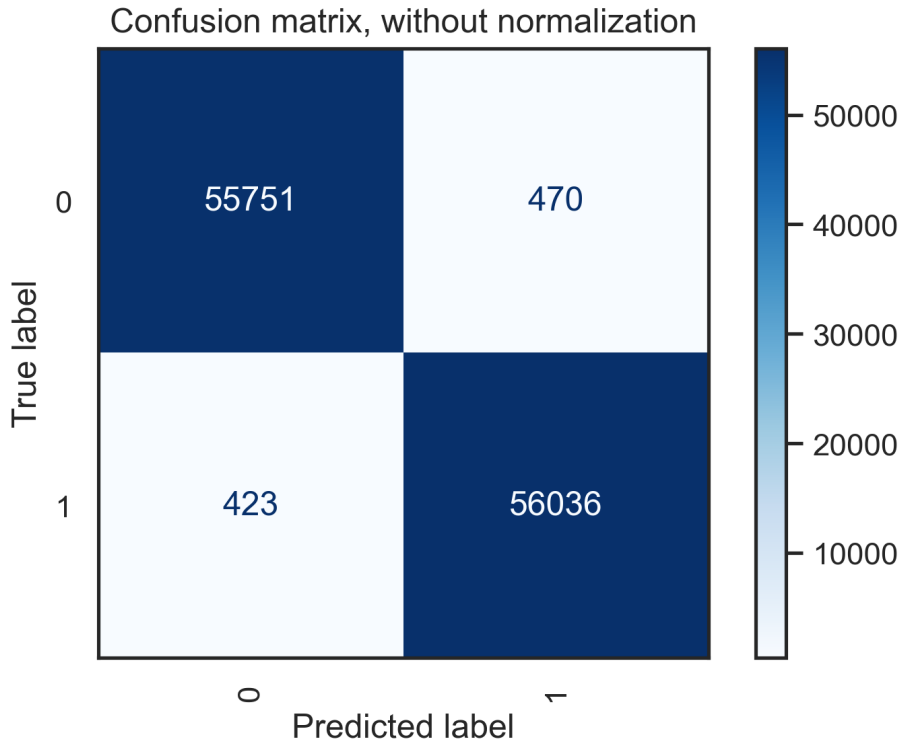


Figure 8: Confusion matrix, SMOTE Oversampling and No PCA

6 Hyperparameter tunning

In order to speed up the development process, we are gonna test different models and its respectives configurations with the Random Undersampler and PCA=15 features. We will find out the next results while training these models with the default configurations:

model_name	Mean Accuracy	Standard deviation
DecisionTreeClassifier	87.15%	2.42%
RandomForestClassifier	87.15%	2.42%
SVC	50.01%	0.01%

Figure 9: Accuracy for Decision Tree, Random Forest and SVC

In this table we will be reflecting the Mean Accuracy Score between all the 5 folds and the Standard Deviation between these. As we can see we get the same scores for the Decision Tree and Random Forest, but for the SVC with default settings we will only get around 50% which means our model is underfitted.

Our next step is gonna be testing 40 different SGDClassifier configurations by alternating different "alpha", "loss", "max_iter" and "penalty" hyperparameters. After calculating the accuracy metric of all configurations we will get that the best one is 50.1001%, which means the model is still underfitted. Given this we can conclude that this model isn't suitable for this case and we can try next with Random Forest.

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
Score CV Test: 0.5010099011879062
 Score CV Training : 0.5001931458832012
 {'penalty': 'l2', 'max_iter': 5000, 'loss': 'log_loss', 'alpha': 0.0001}
```

Figure 10: Best SGDClassifier configuration based on accuracy score

As we can see here we are trying out 5 different Random Forest configurations by tunning the "n_estimators", "max_depth", "min_weight_fraction_leaf", "criterion", "max_features" and "bootstrap" hyperparameters. The best model we will get has an accuracy of 87.18% with the hyperparameters shown in the image below.

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Score CV Test: 0.8718254198262283
 Score CV Training : 0.8718254352177469
 {'n_estimators': 100, 'min_weight_fraction_leaf': 0.03, 'max_features': 'log2', 'max_depth': 5,
'criterion': 'entropy', 'bootstrap': True}
```

Figure 11: Best RandomForestClassifier configuration based on accuracy score

7 Advanced AutoML usecase

For our last and most advanced approach we are going to use the AutoML TPOT tool which automates a great part of a Machine Learning project workflow including: Feature Preprocessing, Model Selection and Parameter Optimization.

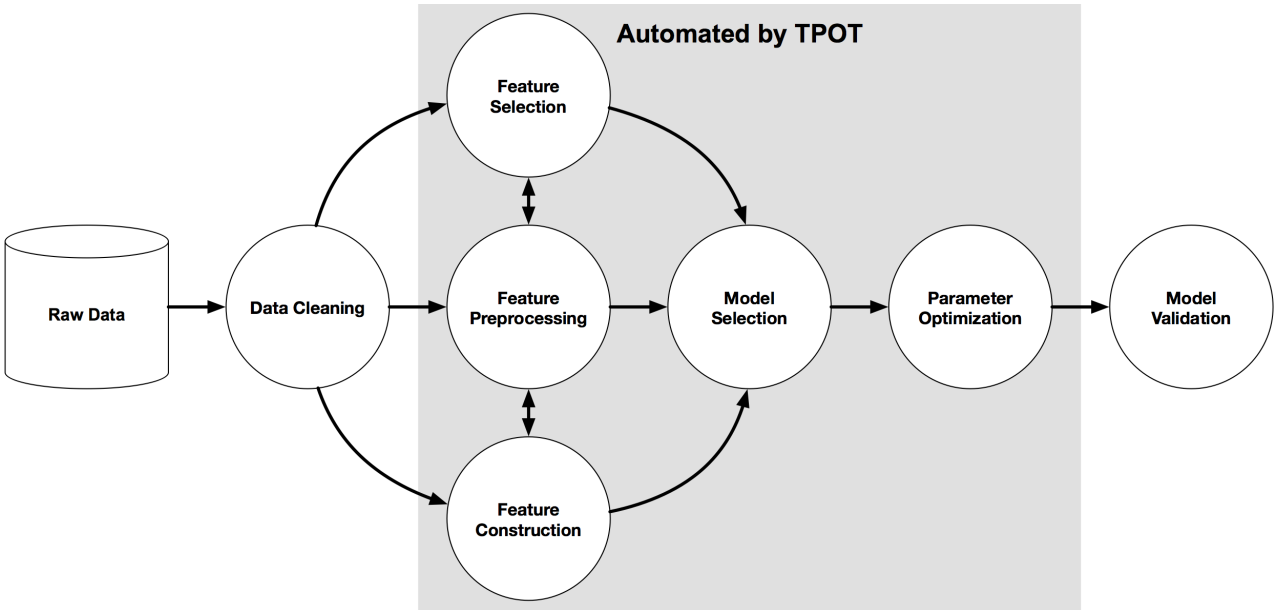


Figure 12: TPOT AutoML tool workflow

For this approach we will be using two different configurations, the one with more data is gonna run for less generations to speed up the computing process. As the table down below shows the Configuration 1 is using SMOTE Oversampler while the Configuration 2 uses the Random Undersampler. This is the reason why the first one will be running for 5 generations instead of 15, as it will take longer to compute for each generation.

After all the computations we will find out that the Configuration 1 gets an accuracy score of 98.99% with no Preprocessing functions and the Extra Trees Classifier. On the other side, Configuration 2 gets and accuracy of 90.15% with PCA Preprocessing and KNeighbors Classifier. In conclusion, we can get very high end results using this dataset but the main drawback for our analysis will be the computations power / time.

	Configuration 1	Configuration 2
Generations	5	15
Sampling	SMOTE Oversampling	Random Undersampling
Preprocessing	None	PCA
Model	ExtraTrees	KNeighbors
Accuracy	0.9899	0.9015

8 Bibliography

Viet Hung Nguyen and Fabio Massacci, "A Systematically Empirical Evaluation of Vulnerability Discovery Models: a Study on Browsers' Vulnerabilities", 2013

Faranak Abri, "Can Machine/Deep Learning Classifiers Detect Zero-Day Malware with High Accuracy?", 2019

Mingyue Yang, "Using Machine Learning to Detect Software Vulnerabilities", 2020

Mohammad Shamsul Hoque, "An Improved Vulnerability Exploitation Prediction Model with Novel Cost Function and Custom Trained Word Vector Embedding", 2020

Kaggle Database: https://www.kaggle.com/competitions/malware-detection/data?select=Kaggle-data.csv