Cyber Security Attack Type Detection

Pythonlab Machine learning project
Group 10

Hanna Abi Akl (Supervisor), Akash Sriramadasu (DE), Minh Quand Le (DA), Xuejing Li (DA), Thuy Trang Nguyen (DA), Vidwan Reddy Nimma (DE), Perrin Letembet-Luquet (DE), Jordan Porcu (DS)

Github

project github link

1. Introduction

1.1 Context and goal

The goal of this project is to build a machine learning pipeline that can predict the **Attack Type** from a dataset with cybersecurity attacks inputs. The project will consist of ordered steps such as Exploratory Data Analysis, feature engineering, training and evaluation of models. Then, we will create a web application to allow live testing of the project.

1.2 Description of dataset

The given dataset contains **40,000 inputs** with **25 features**. The data is raw (not cleaned, not pre-processsed). We have features such as : date, ip addresses, geolocation, post attack metrics etc. More details in the next section.

2. Methodology

2.1 Exploratory Data Analysis

2.1.1 Structure & types

The dataset contains features such as :

- **Temporal** : Timestamp
- **Numerical** : Packet Length, Anomaly Scores
- **Categorical** : Source and Destination IP Addresses, Source and Destination Port, Protocol, Packet Type, Traffic Type, Action Taken, Attack Signature, Severity Level, Network Segment, Geo-location Data, Payload Data, User Information, Device

Information, Malware Indicators, Alerts/Warnings, Firewall Logs, IDS/IPS Alerts, Log Source

2.1.2 Missing values and cardinality

The EDA shows that :

- Several columns have around 50% missing values :
  Malware Indicators (50%), Alerts/Warnings (50.17%), Proxy Information (49.63%), Firewall Logs (49.90%), IDS/IPS Alerts (50.12%).
- Some of these columns have only 1 unique value (ex : Malware Indicators, Alerts/Warnings, Firewall Logs, IDS/IPS Alerts). Which shows (and get confirmed after more investigation) that they are “hidden” boolean.
- High cardinality on some features :
  Source and destination IP addresses (40,000 uniques), Payload Data (40,000 uniques), User Information (32,389 uniques), Device Information (32,104 uniques).

For the main notebook, we created a python function called big_info() that is a more detailed version of the .info() function of Pandas. We will use it through the EDA and the data engineering part to make sure we are on the right path concerning these parts. The results are given in annexes.

2.1.3 Target and classes balance

Attack Type, the target, has 3 distinct classes (DDoS, Intrusion, Malware). The main observation of the EDA is that the target is perfectly balanced. It’s the same for all the features. For example, all the features with 3 classes are 33/33/33% balanced. For the numerical values, no outliers are found.

2.1.4 Hypothesis concerning the quality of the data

In the EDA and the main notebook, we noticed some observations :

- Payload Data looks like Lorem ipsum kind of text, it seems randomly generated
- Geo-location Data seems to consist of Indian cities and states. We also notice that it has nothing to do with IP addresses location, and the cities and states are non coherent (the states associated with the cities are not reflecting reality, which suggest a random generation for it)
- **EDA conclusion :** the main hypothesis is that the dataset is randomly generated. Now the goal is to determine if there is a dependency between the features and the target, which would explain the “hidden” rule of random generation. If not, which would imply that the rule to attribute the Attack Type to any log is “pick 1 of 3 random type”, then any model trained could only reach the accuracy of a randomizer (= 33.33% accuracy)

2.2 Feature Engineering

2.2.1 Strategy

Due to the high cardinality of some categorical features, we choose to extract features from these columns. Here is the strategy we adopted :

1. **Extract the features :** depends of each features, any extraction will be detailed below
2. **Transform :** turning the hidden boolean into real boolean features
3. **Delete :** for the high cardinality features that would explode the dataset on the encoding part

2.2.2 Features created

A) Timestamp

- year, month, day, hour, minute, second
- ts_dayofweek
- ts_is_weekend (boolean)

B) Source and Destination IP

- src_country (via GeoLite2 base)
- dst_country
- same_country (boolean)

We could have also used the city and the ASN (and others) but we did and we had a high cardinality result again. So we decided to only use the countries

C) Source et Destination Ports

- src_port_class and dst_port_class with 2 classes : registered ($\leq$ 49151) or dynamic ($>$ 49151)

D) Payload Data

This column is a bit special since it's latin random generated words, but since the goal is to find a logical rule that led to the random generation of Attack Type, we can't afford to ignore a potential pattern in the generated payload data. That's why we extract :

- payload_char_count
- payload_word_count
- payload_punct_count
- payload_punct_ratio

E) User Information

- user_first_name, user_last_name
- user_char_count

F) Device Information

- device_os
- device_type
- device_browser

G) Geo-location Data

- geo_city
- geo_state

It's important to note 2 things : 1. The geolocation has nothing to with the source or destination IP address geolocation and 2. The city and state are completely random since most of the observations of cities don't belong to the associated state. We conclude that the column was generated by a rule like : Geo-location Data = rand(city) + " , " + rand(state)

H) Proxy Information

- is_proxy (boolean : is there a proxy or not)

2.3 Data Cleaning

2.3.1 Deleting the high cardinality features

As said above, we delete the high cardinality features, because we now have the extracted features from these columns :

- Timestamp
- Source/Destination IPs
- Source/Destination ports
- Payload Data
- User Information
- Device Information
- Geo-location Data
- Proxy Information

2.3.2 Transformation of boolean features

Hidden boolean features transformed into real boolean :

- Malware Indicators
- Alerts/Warnings
- Firewall Logs
- IDS/IPS Alerts

2.3.3 Missing values

The last data cleaning part is to remove the lines with missing values, since it only concerns few lines (less than 1% of the dataset)

2.4 Training and evaluation

2.4.1 Split Train Test

- Split in **80/20%** (classic in machine learning)
- Isolation of target
- random_state = 1337 (for reproducibility)

2.4.2 Preprocessing pipeline

Here again, as a basic machine learning pipeline :

- **StandardScaler** on numerical features
- **OneHotEncoder** on categorical features

2.4.3 Selected models

We chose to evaluate 3 models :

- **a logistic regression model** : for basic multi-class classification problem
- **a random forest model** : for more complex multi-class classification problem
- **a decision tree model** : for non-linear multi-class classification problem

2.4.4 Metrics

We chose metrics such as :

- **Accuracy** (main one)
- **F1 macro** (just to add more details in observation)
- **Train score vs Test score** to detect over/underfitting

---

3. Results

3.1 Compared performances

As a final results we have :

Model	Accuracy	F1-Macro	Train score	Test score
LogReg	0.3425	0.3424	0.4391	0.3425
RandomForest	0.3327	0.3324	1.0000	0.3327
DecisionTree	0.3260	0.3260	1.0000	0.3260

3.2 Interpretation

1. **Performance really close to randomness**
   This result is really coherent with our hypothesis : the dataset being randomly generated, there is no obvious dependency between the features and the target. It looks more like the Attack Type being randomly given to any log, not depending on any feature.
2. **Overfitting on complex model**
   The face that the complex models reach 100% accuracy on the train set but 33% on the test set shows that we are facing overfitting. It means that the model can perfectly learn from the training set, but cannot reproduce any pattern learnt on new data => it acts like a random generator (which explains the 33%)

3.3 Max_depth fine-tuning

Because the particularity of DecisionTree is its non-linearity, one of our final tests to confirm our hypothesis is to showcase the overfitting on different max_depth (fine-tuning). Here is the result :

max_depth	Train score	Test score
1	0.3361	0.3356
2	0.3375	0.3284
4	0.3396	0.3279
6	0.3414	0.3324
8	0.3444	0.3275
10	0.3478	0.3275
20	0.3831	0.3288
50	0.6676	0.3324
100	1.0000	0.3260

So what we can see here is that for a low depth, the model cannot really find anything better than a randomizer, but as we increase the depth, it slowly turns into an overfitting case.

5. Conclusion

5.1 Synthesis

- A complete machine learning pipeline has been created : EDA, feature engineering, data cleaning, model selection, evaluating

- The results of the multi class problem are close to randomness, despite :
    - feature extraction on several categorical features (we ended up with 40 features)
    - use of non-linear complex model (Decision Tree)

5.2 Main interpretation

From our results, we conclude that either :

- Attack Type is independent of other features (randomly generated)
- the generation rule of Attack Type is too complicated to capture with the given features
- we need a really more complex model such as neural networks

5.3 Enhancement insights

1. **Leak of data and post attack features**
    - Some columns are post attack features. It could be worth it to try an only post attack or only pre attack features experiment.
2. **Cross-validation**
    - Instead of one split, use cross validation to stabilize the metrics.
3. **Use a neural network model**
    - Because of the context of the project, we only used “basic” machine learning models. It could be interesting to see if a really complex neural network model can capture the complexity of the supposed target dependency to features.

4. Deployment

As asked in the project, we created a web app (Streamlit) to test the model live.

It can be found in this link

The web app takes 2 input modes :

- Upload CSV
- Random row(s)

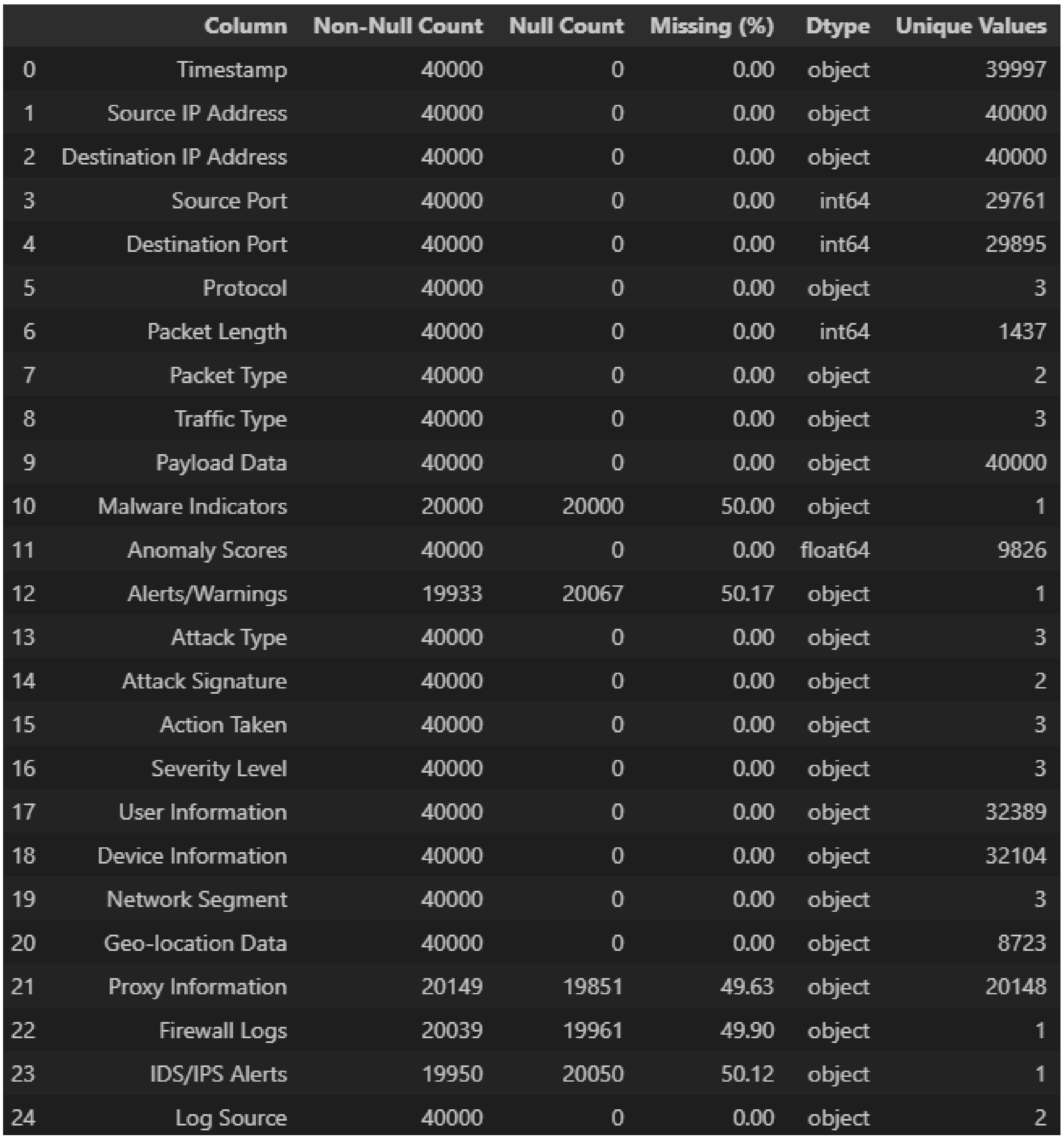
In **upload CSV**, the user can upload its own data. We provided in the github 4 ready-to-use .csv files (in the csv folder) :

- random_5.csv : 5 randomly generated (from every possible data on each features) lines
- random_10.csv : 10 randomly generated
- sample_5.csv : a .sample(5) of the original dataset
- sample_10.csv : a .sample(10) of the original dataset

In **random row(s)**, the user can select X number of rows, and will generate X lines of data to predict.

In both modes, the user can then now click on “predict” to see each model prediction on the selected data. A randomizer (not a model) has been added on both mode to compare

- Annex A : big_info()



	Column	Non-Null Count	Null Count	Missing (%)	Dtype	Unique Values
0	Timestamp	40000	0	0.00	object	39997
1	Source IP Address	40000	0	0.00	object	40000
2	Destination IP Address	40000	0	0.00	object	40000
3	Source Port	40000	0	0.00	int64	29761
4	Destination Port	40000	0	0.00	int64	29895
5	Protocol	40000	0	0.00	object	3
6	Packet Length	40000	0	0.00	int64	1437
7	Packet Type	40000	0	0.00	object	2
8	Traffic Type	40000	0	0.00	object	3
9	Payload Data	40000	0	0.00	object	40000
10	Malware Indicators	20000	20000	50.00	object	1
11	Anomaly Scores	40000	0	0.00	float64	9826
12	Alerts/Warnings	19933	20067	50.17	object	1
13	Attack Type	40000	0	0.00	object	3
14	Attack Signature	40000	0	0.00	object	2
15	Action Taken	40000	0	0.00	object	3
16	Severity Level	40000	0	0.00	object	3
17	User Information	40000	0	0.00	object	32389
18	Device Information	40000	0	0.00	object	32104
19	Network Segment	40000	0	0.00	object	3
20	Geo-location Data	40000	0	0.00	object	8723
21	Proxy Information	20149	19851	49.63	object	20148
22	Firewall Logs	20039	19961	49.90	object	1
23	IDS/IPS Alerts	19950	20050	50.12	object	1
24	Log Source	40000	0	0.00	object	2