Security aspects of machine learning
Assignment 2:
Malware Detection and Hardware Trojan Detection

Daniil Terekhin

June 6, 2023

1 Malware Detection

1.1 Understanding the Data

The dataset consists of feature vectors of 215 attributes extracted from 15,036 applications. In the dataset there several kinds of features: Manifest Permission, API call signature, Intent and Commands signature. From these kinds of features i can gain insights into the behavior, functionality, and potential malicious activities of the programs in the dataset. Identifying patterns, anomalies, or correlations in these features can help in detecting hardware trojans or understanding the characteristics of the programs associated with them. I think that such kind of features as Manifest Permission would be important to detect malware, especially SEND_SMS, READ_PHONE_STATE, READ_EXTERNAL_STORAGE and ACCESS_FINE_LOCATION.

I think that additional feature engineering is necessary, because there could be interesting insights in data and correlations of features. For creating new features it is nessasary to be aware of each of these attributes as deep as possible.

1.2 Evaluating Models

Considering the potential consequences of misclassification, in most cases, it is more impactful to minimize false-negatives (malware applications classified as benign). The priority is to detect and prevent malware from being labeled as benign to ensure the security and privacy of users.

For this classification I think the F1 score is the best way to evaluate models. It provides a balanced measure that considers both false positives and false negatives. It is useful when there is an uneven class distribution (9476 benign and 5560 malicious programs).

1.3 Training Your Models

For classification I chose these three models: Decision tree, Gaussian Naive Bayes and Gradient boosting.

Decision tree is suitable for this task because can capture non-linear relationships and interactions between features, allowing them to learn complex decision boundaries.

Gaussian Naive Bayes is a suitable classifier for this task because it assumes that the features follow a Gaussian distribution. In the context of application classification, certain features such as API call signatures or permissions may exhibit Gaussian-like distributions. Naive Bayes classifiers are computationally efficient, especially with high-dimensional data.

Gradient boosting is a powerful ensemble method that combines multiple weak learners (decision trees in this case) to create a strong classifier. Gradient boosting iteratively fits the model to the residuals of the previous iteration, allowing it to focus on the samples that are more difficult to classify correctly. This makes it well-suited for complex classification tasks like distinguishing between benign and malware applications.

1.3.1 Decision tree (DecisionTreeClassifier)

I consider different values of max depth parameter and made training curve, Figure 1. According to these curves I chose 11 as max depth. I got 0.957 F-score for test dataset, Figure 2. This model works good for recognizing malware applications. Such big F-score means that model detect true positive and true negative, so model does not biased.

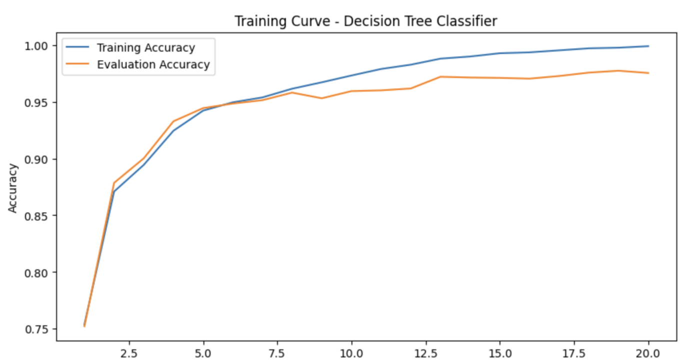


Figure 1: Accuracy score for DecisionTreeClassifier.

```
Accuracy: 0.9574326571333555
Precision: 0.946894689468947
Recall: 0.9384478144513827
F1 score: 0.9573935764157876
```

Figure 2: Result on test dataset

1.3.2 Gaussian Naive Bayes (GaussianNB)

This model does not have any parameters. That is why i used K-fold and cross validation for training model. Final score was 0.71 for F-score, Figure 3. For Recall i had 0.97, but for Precision only 0.56. It means that model good at predicting positive observations over all the actual positives (less number of false negatives), but does not good at predicting positive observations over the total predicted positives (high number of false positives).

This model does not have training curve.

```
Fold 1 - Accuracy: 0.6915
Fold 1 - Precision: 0.5459
Fold 1 - Recall: 0.9838
Fold 1 - F1-Score: 0.7022
------------------------------
Fold 2 - Accuracy: 0.7039
Fold 2 - Precision: 0.5568
Fold 2 - Recall: 0.9784
Fold 2 - F1-Score: 0.7097
------------------------------
Fold 3 - Accuracy: 0.7119
Fold 3 - Precision: 0.5643
Fold 3 - Recall: 0.9712
Fold 3 - F1-Score: 0.7138
------------------------------
Fold 4 - Accuracy: 0.7219
Fold 4 - Precision: 0.5731
Fold 4 - Recall: 0.9730
Fold 4 - F1-Score: 0.7213
------------------------------
Fold 5 - Accuracy: 0.7199
Fold 5 - Precision: 0.5702
Fold 5 - Recall: 0.9856
Fold 5 - F1-Score: 0.7225
------------------------------
...
Accuracy: 0.7140
Precision: 0.5658
Recall: 0.9762
F1-Score: 0.7163
```

Figure 3: Result on test dataset

1.3.3 Gradient boosting (GradientBoostingClassifier)

Gradient Boosting model has good results with 0.94 F-score, Figure 4. I made training curve according to changing number of estimators, Figure 5. Accuracy reached 0.95 at 200 number of estimators, so I took this value for tuning my model. This model works good for predicting malware applications.

```
Scores for Text data
Accuracy: 0.9394745593614898
Precision: 0.921832884097035
Recall: 0.9152542372881356
F1 score: 0.9394302360222226
```

Figure 4: Result on test dataset

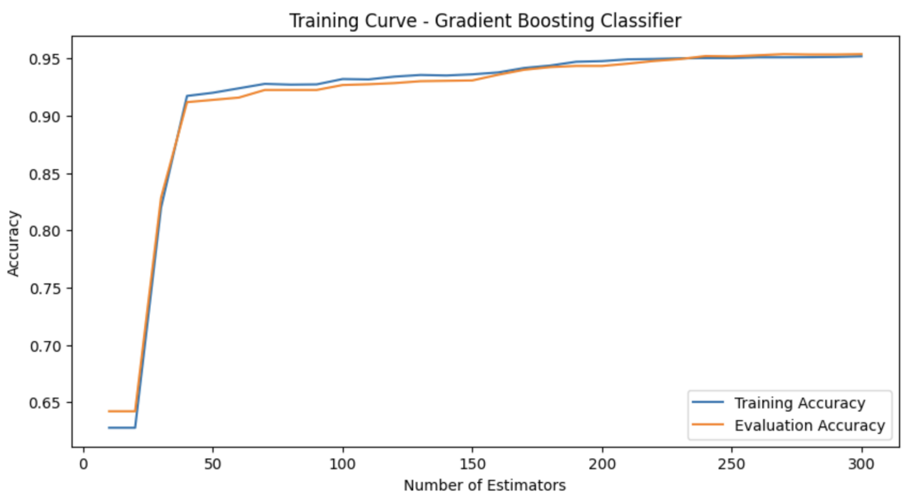


Figure 5: Training curve for number of estimators

1.4 Quality of the Dataset

The provided dataset is valuable for real-world malware detection due to its balance between benign and malicious applications and the variety of binary feature values like API calls, manifest permissions, and intent signatures. However, enhancing it with time-based data, application metadata, behavioral characteristics, and regular updates could make it more effective. Additional contextual data about the running environment could also be beneficial.

1.5 Bypassing your Model

For bypassing my model I need to change functionality of an application.
I took index of the application 2021, which class is malware, Figure 5. I tried several approaches for bypassing the model: 1) change random features which have the most importance for the model, 2) change certain features which were

detected by intersection of the most frequently used features for malicios programs and least used features for benign programs, 3) change certain most important features.

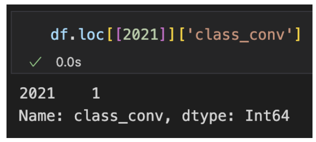


Figure 6: Index of an application

1.5.1 Approach 1

My first idea is to analyze top 4 most important features from 'feature_importances_' of our Gradient Boosting classifier and change values in these columns to opposite (from 0 to 1 and from 1 to 0). These features are:
1) READ_PHONE_STATE,
2) SEND_SMS,
3) transact,
4) onServiceConnected

This approach did not work, model still predicted this application as malicious.

1.5.2 Approach 2

In this approach I tried to find most commomly used features for malicios programs and least used features for benign programs and took intersection of them. After this i checked these features to having 1 and took columns with only 1s. This information could give me insights which parameters I need to change for bypass the model.

So i found these features:
1) READ_EXTERNAL_STORAGE,
2) chmod,
3) BROADCAST_STICKY

This approached worked for me. I could bypass the model, it predicted this application and benign.

1.5.3 Approach 3

In this approach I analyzed features and chose only which i was going to change from 0 to 1. I chose:
1) transact,
2) Ljava.lang.Class.getCanonicalName

This approach also worked for me. Model predicted this application as benign.

1.6 Improving your Model

I used Approach number 3 fro starting point. For improving my GradientBoostingClassifier model I can use different parameters such as max_depth and max_features, which could limit feature weights and make the mode more protected.

I used GridSearchCV and found the most suitable parameters for the model: learning_rate': 0.1, 'max_depth': 7, 'max_features': 'sqrt', 'n_estimators': 200

It helped me to make the model more protected. It could still predict this application as malicious, so my changes do not have any impact. Overall the perfomance also increased, i had 0.99 F-score.

2 Hardware Trojan Detection

2.1 Dataset

I took dataset from Kaggle platform (https://www.kaggle.com/datasets/subhajournal/trojan-detection). The data can be used to detect the Trojan Horse Traffic. The data contains the records of the traffics like Trojan Horse and Benign so the detection of Trojan and Benign can be done using Binary Classification. The data is originated from CIC website.

Trojan Data with 90683 Observations
Benign Data with 86799 Observations

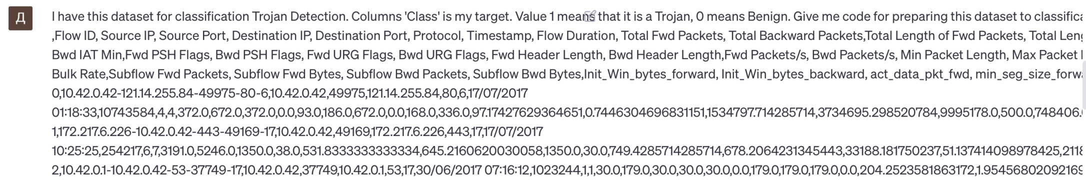
There were 177482 rows and 85 columns, such as Source IP, Source Port, Protocol, Flow Duration, Total Length of Fwd Packets and others.

2.2 Chat GPT

For this task I asked chat GPT to write me code for predicting Trojan and Benign. I always asked chat to improve results, add metrics, use more advanced preprocessing techniques, add use ensemble methods and cross validation, Fugires 7 - 11.

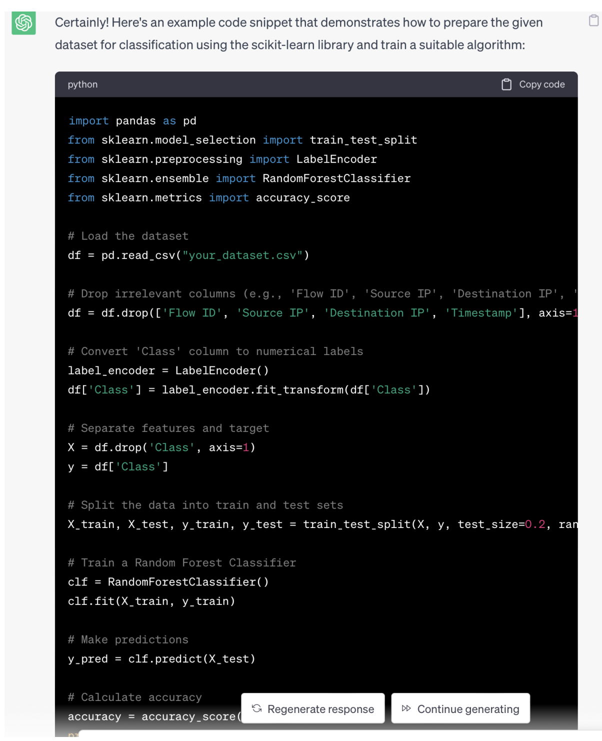
2.3 Models and results

For this task I (chat GPT) used ensemble of RandomForestClassifier, GradientBoostingClassifier and XGBClassifier. In Figure 12 we can see result: 0.73 F-score.


Д

I have this dataset for classification Trojan Detection. Columns 'Class' is my target. Value 1 means that it is a Trojan, 0 means Benign. Give me code for preparing this dataset to classifica
,Flow ID, Source IP, Source Port, Destination IP, Destination Port, Protocol, Timestamp, Flow Duration, Total Fwd Packets, Total Backward Packets,Total Length of Fwd Packets, Total Len
Bwd IAT Min,Fwd PSH Flags, Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, Fwd Header Length, Bwd Header Length,Fwd Packets/s, Bwd Packets/s, Min Packet Length, Max Packet l
Bulk Rate,Subflow Fwd Packets, Subflow Fwd Bytes, Subflow Bwd Packets, Subflow Bwd Bytes,Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forw
0,10.42.0.42-121.14.255.84-49975-80-6,10.42.0.42,49975,121.14.255.84,80,6,17/07/2017
01:18:33,10743584,4,4,372.0,672.0,372.0,0.0,93.0,186.0,672.0,0.0,168.0,336.0,97.17427629364651,0.7446304696831151,1534797.714285714,3734695.298520784,9995178.0,500.0,748406.
1,172.217.6.226-10.42.0.42-443-49169-17,10.42.0.42,49169,172.217.6.226,443,17,17/07/2017
10:25:25,254217,6,7,3191.0,5246.0,1350.0,38.0,531.8333333333334,645.2160620030058,1350.0,30.0,749.4285714285714,678.2064231345443,33188.181750237,51.137414098978425,211
2,10.42.0.1-10.42.0.42-53-37749-17,10.42.0.42,37749,10.42.0.1,53,17,30/06/2017 07:16:12,1023244,1,1,30.0,179.0,30.0,30.0,30.0,0.0,179.0,179.0,179.0,0.0,204.2523581863172,1.954568020921

Figure 7: My first request with samples of the dataset

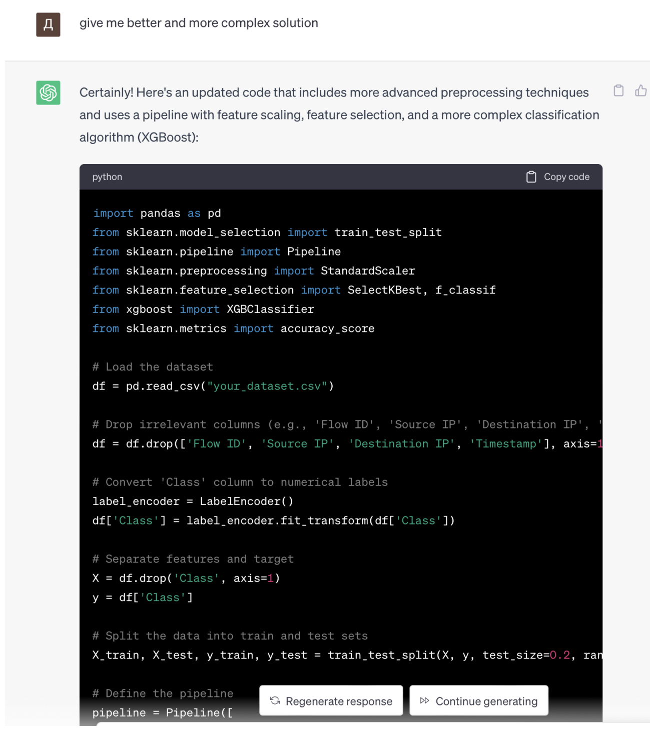

Certainly! Here's an example code snippet that demonstrates how to prepare the given dataset for classification using the scikit-learn library and train a suitable algorithm:

python Copy code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
df = pd.read_csv("your_dataset.csv")

# Drop irrelevant columns (e.g., 'Flow ID', 'Source IP', 'Destination IP', '
df = df.drop(['Flow ID', 'Source IP', 'Destination IP', 'Timestamp'], axis=1

# Convert 'Class' column to numerical labels
label_encoder = LabelEncoder()
df['Class'] = label_encoder.fit_transform(df['Class'])

# Separate features and target
X = df.drop('Class', axis=1)
y = df['Class']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Train a Random Forest Classifier
clf = RandomForestClassifier()
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(
```

Regenerate response Continue generating

Figure 8: Chat GPT Answer



give me better and more complex solution

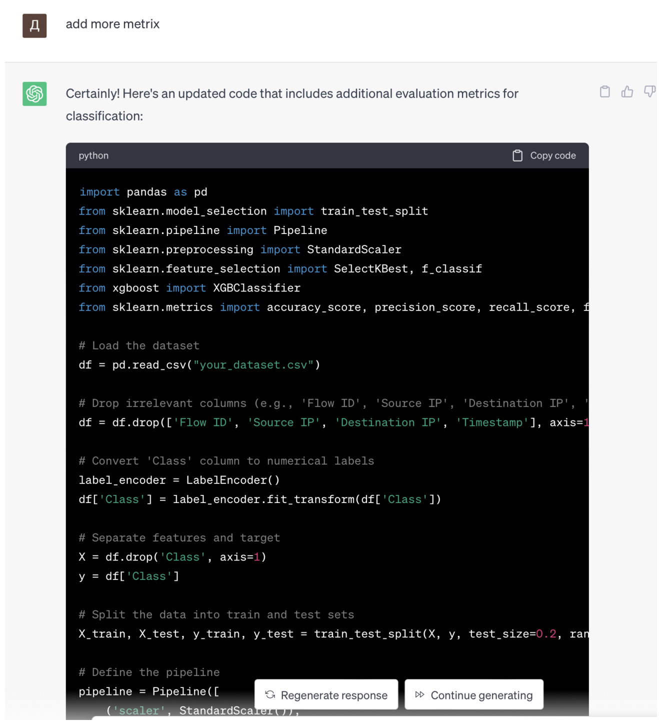
Certainly! Here's an updated code that includes more advanced preprocessing techniques and uses a pipeline with feature scaling, feature selection, and a more complex classification algorithm (XGBoost):

python Copy code

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
df = pd.read_csv("your_dataset.csv")

# Drop irrelevant columns (e.g., 'Flow ID', 'Source IP', 'Destination IP', '
df = df.drop(['Flow ID', 'Source IP', 'Destination IP', 'Timestamp'], axis=1

# Convert 'Class' column to numerical labels
label_encoder = LabelEncoder()
df['Class'] = label_encoder.fit_transform(df['Class'])

# Separate features and target
X = df.drop('Class', axis=1)
y = df['Class']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Define the pipeline
pipeline = Pipeline([
```

Regenerate response Continue generating

Figure 9: Request for getting better solution



Д add more metrix

Certainly! Here's an updated code that includes additional evaluation metrics for classification:

python Copy code

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f

# Load the dataset
df = pd.read_csv("your_dataset.csv")

# Drop irrelevant columns (e.g., 'Flow ID', 'Source IP', 'Destination IP', '
df = df.drop(['Flow ID', 'Source IP', 'Destination IP', 'Timestamp'], axis=1

# Convert 'Class' column to numerical labels
label_encoder = LabelEncoder()
df['Class'] = label_encoder.fit_transform(df['Class'])

# Separate features and target
X = df.drop('Class', axis=1)
y = df['Class']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Define the pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
```

Regenerate response Continue generating

Figure 10: Request for adding more metrics

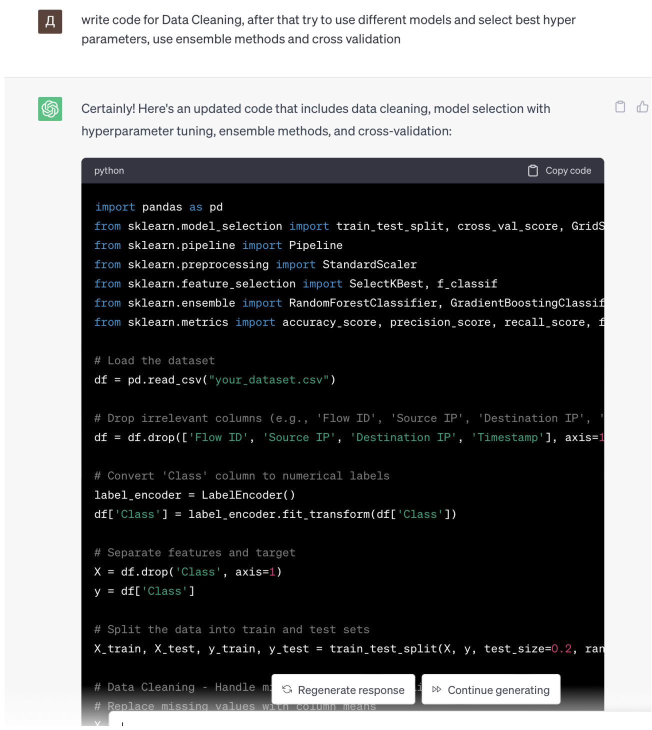


Figure 11: Request for Data cleaning and using ensemble methods

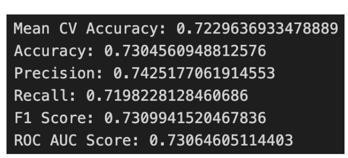


Figure 12: Result

2.4 Conclusion

If a person knows how machine learning and data preprocessing work, he could lead chat GPT in the right direction and sometimes ask it to change or add something fro improving the final score for certain task.