Solution for the 11th International Cybersecurity Data Mining Competition (CDMC 2020)

The solution was prepared by team from Sberbank Cybersecurity department:
Roman Leksutin RVLeksutin@sberbank.ru
Kirill Cherkashin KICerkashin@sberbank.ru
Artem Sokolov Sokolov.A.Ily@Sberbank.ru

Solution created using Python v3.7

Task 1. Android Malware Detection

Feature selection

In this task, we are dealing with some undefined features (2493 features) which describe Android Malware categories.

For this solution during preprocessing step we dropped features that has only 1 unique value (1038 features) and features with linear dependency (116 features). Also, we found out that all the remaining features (excluding hashes, which values we failed to find useful) do not allow us to separate all given classes (there are some samples with similar feature representation but with different labels). Dropping duplicates did not lead to an improvement in model performance, so we kept them in the training dataset.

We tried some other feature selection techniques, including selection by SHAP values, feature importance with Lightgbm, but it led to lower performance scores.

Therefore we used 1339 features for building models.

Experiments with models

For experiments and performance evaluation we did 0.7/0.3 stratified split for training and hold-out set (all the preprocessing steps we did after the split). Most of our experiments in this task were centered around the family classification task which was a little harder, considering a lot of classes with only one sample in the training set.

We used the macro F1 score as our main metric for model selection both for validation and hold-out datasets.

Some of our most noticeable results:

- *AutoML*. We used LightAutoML (LAMA) model developed in our organization (in closed Beta right now, will go opensource in December). It gave us considerably high scores (0.84) on the hold-out set, but was susceptible to overfitting;
- Solution based on *Lightgbm*. Gave us standard deviation on cross-validation close to zero (most stable and robust model), but overall lower score (0.8).
- Most of our models using One-vs-Rest strategy to handle multiple classes, but we also tried *ClassifierChains* method hoping to get a performance boost from hidden dependencies between classes. It also failed to give us better results;
- We tried different *combinations of models*, including stacking and/or blending of Lightgbm, KNN, RandomForest, Logistic regression, Naive Bayes, Multilayer perceptron (feedforward neural net), but did not gain any performance increase.

Best model

Our best model was Random Forest. We first tweaked hyperparameters manually. We tried to play with `{n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, class_weight, max_features, max_samples}` and then used grid_search for most impactful after analysis (n_estimators, max_depth, and class_weight) and ended up with

```
{'n_estimators':250, 'max_depth':None, 'class_weight':None}
```

On hold-out set it gave us 0.82 but was more stable on cross-validation than AutoML.

In order to reproduce the best solution, it is necessary to download the Testing and Training set into task_1/data folder, install the necessary requirements

```
pip install -r requirements.txt
```

and then run the script from task_1 folder

```
python train_predict_task1.py
```

Task 2. IOT Malware Detection

Feature engineering

In this task, we are trying to distinguish some set of IOT Malware Families given their encrypted and encoded byte sequences.

The main challenge was to find the best way to represent the given data in a vectorized form. Unfortunately, we were not able to use domain knowledge of ELF files structure to extract some meaningful features because byte sequences were encrypted, so we tried text-based and image-based approaches.

In this solution, we decoded radix64 strings back into bytes and used default python bytes representation in integers. Each sample of our data was converted to a list of integers in the range from 0 to 255. Then we made a vocabulary of all subsequences of those bytes occurred in files with the following lengths: `[1, 2, 4, 8, 16]` (using sklearn CountVectorizer with params: `{max_df=0.9, min_df=0.1}`). The resulted vocabulary was used for fitting of TfIdfVectorizer with the following parameters: `{ngram_range=(1, 16), max_df=0.8, vocabulary=vocab, min_df=0.2}`.

A column with CPU architecture we vectorized using One-Hot-Encoding technique.

We also tried to use radix64, ASCII symbols (including and excluding '\x') as tokens, and also used bytes in decimals as numerical features, but those approaches gave us worse results.

Experiments with models

Given target distribution (rarest class has only 4 samples in train data) we used 4-kfold split, one split for hold-out, and the other for cross-validation. For evaluation F1 score was used as well.

Most of our models performed relatively well on most of the classes, except Xorddos and Pnscan, so the main challenge was to build a model that can distinguish these rare malware families.

Among the noticeable experiments we have:

- *Neural Net* with LSTM on top of the 1-d convolution (image-based approach). Inspired by this paper we tried to fit 1-d convolutional network on integer bytes representation and feed final layer to LSTM (you can check out our PyTorch implementation at task_2/convnet.py). It gave us reasonable results (~0.8), but mostly failed on rare classes, even with weighted loss and oversampling;
- *AutoML* (same model as in the first task). Did not gave us reasonable scores (best was around 0.73);
- Different combinations of *classical algorithms*, including RandomForest and GradientBoosting.

Best model

The only algorithm which dealt really well with scarce training data of rare classes was KNN with the number of neighbors set to 1 and distance metric to 'manhattan' (parameters was tuned on cross-validation using grid-search). On hold-out set this solution gave us 0.95 f1 score.

In order to reproduce the best solution, it is necessary to download the Testing and Training set into task_2/data folder, install the necessary requirements

```
pip install -r requirements.txt
```

and then run the script form task_2 folder

```
python train_predict_task2.py
```