CMSC 25422: ML for Computer Systems Final Project Report

Author: Robert Xing | CNetID: rkxing

Project: NetML Malware Detection IoT Leaderboard Results

Premise + Objective

For my final project, I've elected to attempt to reproduce the best results for the NetML Malware Detection Challenge, the objective being to beat nPrintML's results on both the easy and hard (top and fine) levels of classification.

Data

This project uses the data provided by the organizers of the challenge, namely the NetML dataset, which contains two levels of problems: a top-level classification problem that has two target classes of malware, and a fine-level problem that has nineteen classes. The testing set is split up further into two sets, one denoted the 'std' set and the other the 'challenge' set. I've tested my model against both of these sets for the most comprehensive results. Labels–or "annotations"–are also provided by the organizers, to be used for supervised learning on the training set and validation of results on the testing sets. All of the relevant files have been included in my project repository under the 'data' folder, and the original files can also be found here.

Metrics + Benchmarking

The metric to optimize as used by nPrint is the *Balanced Accuracy Score*, for which nPrintML achieved a 92.4 on the top-level set and a 86.1 on the fine-level set. The table below is taken from the paper which presents these results: (paper found here)

Malware Detection for IoT Traces (§5.4.1)	netML IoT [6, 28]	2 19	-4 -t -u	10	92.4 86.1	99.5 96.9	93.2 84.1	99.9 (True Positive Rate) 39.7 (Balanced F1)

Methodology

I tried three different classification models for this challenge, which were:

- Random Forest
- Gaussian Naive Bayes
- Stochastic Gradient Descent

First, exclusively considering the NetML top-level training set (all of the top-level data except for that in test-std and test-challenge), I trained and validated each of these models on an 80-20 train-test split in addition to cross-validation to find the best hyperparameters for each of them. Comparing the testing results, I found that the RandomForestClassifier had the best overall performance on the top-level data.

Training accuracy across the three models was as follows:

In [5]:
```
from sklearn.metrics import balanced_accuracy_score

y_pred = rf_clf.predict(X_test_scaled)
b_acc = balanced_accuracy_score(y_test, y_pred)

print(f'Random Forest Balanced Accuracy: {b_acc:.4f}')
```

```
Random Forest Balanced Accuracy: 0.9966
```

In [9]:
```
y_pred = gnb_clf.predict(X_test_scaled)
b_acc = balanced_accuracy_score(y_test, y_pred)

print(f'GaussianNB Balanced Accuracy: {b_acc:.4f}')
```

```
GaussianNB Balanced Accuracy: 0.6640
```

In [13]:
```
y_pred = sgd_clf.predict(X_test_scaled)
b_acc = balanced_accuracy_score(y_test, y_pred)

print(f'SGD Balanced Accuracy: {b_acc:.4f}')
```

```
SGD Balanced Accuracy: 0.9800
```

I then selected Random Forest to be the candidate for which to try on the fine-level data. Unfortunately, even after extensive experimentation with hyperparameters, the best result I could attain with Random Forest on the 19-class problem was significantly below nPrintML's benchmark of 86.1. Other supervised models performed even worse, and unsupervised models like neural networks were too computationally expensive for me to test on my hardware alone.

In [18]:
```
y_pred = rf_clf_fine.predict(X_test_scaled)
b_acc = balanced_accuracy_score(y_test, y_pred)

print(f'Random Forest (Fine) Balanced Accuracy: {b_acc:.4f}')
```

```
Random Forest (Fine) Balanced Accuracy: 0.5726
```

Satisfied with my results for the top-level data, I then finally moved to test my best model on the test-std and test-challenge sets on both the top and fine levels.

Results

The average balanced accuracy score across test-std and test-challenge on the top-level (2-class) data was **98.44, beating nPrintML's benchmark.**

In [19]:
```
test_std_folder = DATA_PATH + 'test-std_set'
test_std_anno_file_top = DATA_PATH + 'test-std_anno/' + 'top_submission_test-std_anno.json.gz'

# reuse get_training_data() function to load test-std data
X_test, y_test, _, _ = get_training_data(test_std_folder, test_std_anno_file_top)

scaler = StandardScaler()
X_test_scaled = scaler.fit_transform(X_test)

y_pred = rf_clf.predict(X_test_scaled)
b_acc = balanced_accuracy_score(y_test, y_pred)

print(f'Balanced Accuracy (std, Top): {b_acc:.4f}')
```

```
Loading training set ...
Reading 1_test-std_set.json.gz
Balanced Accuracy (std, Top): 0.9837
```

In [20]:
```
test_challenge_folder = DATA_PATH + 'test-challenge_set'
test_challenge_anno_file_top = DATA_PATH + 'test-challenge_anno/' + 'top_submission_test-challenge_anno.json.gz'

X_test, y_test, _, _ = get_training_data(test_challenge_folder, test_challenge_anno_file_top)

scaler = StandardScaler()
X_test_scaled = scaler.fit_transform(X_test)

y_pred = rf_clf.predict(X_test_scaled)
b_acc = balanced_accuracy_score(y_test, y_pred)

print(f'Balanced Accuracy (challenge, Top): {b_acc:.4f}')
```

```
Loading training set ...
Reading 0_test-challenge_set.json.gz
Balanced Accuracy (challenge, Top): 0.9851
```

The balanced accuracy for the fine-level (19-class) data, however, was <50, barely half of what nPrintML achieved.

The overall result of this project is thus that I successfully developed a model that can identify malware vs. benign traffic (binary classification) with high accuracy, but underperforms on multi-label malware classification problems.