Lab 4
Machine Learning for Cybersecurity

Due: October 29, 2022

Setup

The goal of this lab is to introduce basic machine learning concepts as applied to cybersecurity problems. An incomplete Jupyter notebook will be provided as a starting point for each exercise. You will need to turn in the completed Jupyter notebooks (with corresponding output) for credit.

Problem 1: Supervised ML Pipeline

In this first exercise, you will build a supervised machine learning pipeline for classifying whether or not a particular set of network traffic corresponds to normal traffic or fuzzing traffic. The data comes from UNSW ADFA-NB15-Dataset. The source has a good overview of the dataset.

Much of the preprocessing has already been done, but you will have to do some additional processing to get the data ready.

1. Load data from training and testing sets from `data/exercise1/`.
2. Keep the two dataframes separate and create train/test data and labels. This will be used to experiment with the case where there are different types of activities in the training versus test set.
3. Create a new training/test split by combining the dataframes into one. Then split the dataframe randomly into train/test data and labels. This will be used to experiment with the case where there are largely the same types of activities in the training and test set.
4. Train a classifier using various test/train splits and random forest with default parameters

5. or each of the train/test splits and associated random forest, predict labels on the training data and testing data as well as get metrics like precision, recall, f1 and accuracy.

Questions

1. What are the results of precision and recall when we keep train and test data separate (i.e. after Step 2)? Describe.

2. What changes in the result on the test data once we combine the data in Step 3. Does it produce a better or worse classifier. Explain.

Problem 2: Anomaly detection

In this exercise, you will run some anomaly detection experiments. You will take some normal data, inject some malicious datapoints corresponding to worm activity, and run isolation forests in order to try to detect the malicious activity. The data comes from the same source as Problem 1.

1. Load data from normal and malicious sets from `data/exercise2/`.

2. Create 15 datasets, where ith dataset consists of: all normal data, ith fraction of malicious data.

3. For each dataset run isolation forests and evaluate on

    - rank the anomalousness of each datapoint using the isolation forest
    - record the list index of each attack datapoint when sorting from most to least unusual

Questions

1. Why is there no separate training and test set?

2. What is the metric measuring? What would be a perfect score? Bonus: What is the expected performance of an outlier detector that assigns a random score to each datapoint?

3. How well does the isolation forest perform compared to a perfect score? Bonus: How well does the isolation forest perform compared to a random detector?

4. What are some issues that would prevent this model from being practically deployed?

5. What might happen if we inject five attack datapoints at a time? What might happen if we inject 100 attack datapoints at a time?

6. What is the effect of the parameters max_features and max_samples? What other parameters could you adjust to change performance?

7. **Optional Bonus.** What are some alternative anomaly detection models one could use instead of an isolation forest? Try one of these alternatives and compare performance.

Problem 3: Using Syscalls for Host-based Intrusion Detection

In exercise 3, you will run a series of experiments on syscall log data. This data is mostly unprocessed. The data is the AFDA Linux Dataset and comes from a UNSW Sydney project around Host Intrusion Detection Systems [1]. In this dataset, each system call trace is provided in a separate text file. System call parameters have been removed, and each system call is replaced by a numeric ID. Thus, an example system call trace will look like this: 6 6 63 6 42 ...

Part 1

1. Load data: All of the data under `data/exercise3/Training_Data_Master` `data/exercise3/Validation_Data_Master` are normal and data under `data/exercise3/Attack_Data_Master` is malicious. For the purpose of this lab, simply use Training_Data_Master and Validation_Data_Master as a single pool of normal data. Each system call trace is stored as a single file. Treat each trace as a separate datapoint.

2. Feature Extraction: Tokenize and create a dataset where each datapoint corresponds to (normalized) counts of system call n-grams. Try various sizes of ngram i.e. for a sequence "6 6 63 6 42", a 3-gram would be: "6 6 63", "6 63 6", "63 6 42".

3. Create a train/test split and use this to train a classifier.

4. Print inference and result metrics: precision, recall, f1-measure and accuracy.

Part 2

Create several new test datasets where you have randomly subsampled the number of attack datapoints.

- 10 datasets where 25% of the attack datapoints are removed from the original test set
- 10 datasets where 50% of the attack datapoints are removed from the original test set
- 10 datasets where 75% of the attack datapoints are removed from the original test set
- 10 datasets where 90% of the attack datapoints are removed from the original test set
- 10 datasets where 95% of the attack datapoints are removed from the original test set

Report five sets of precision, recall, f1-measure, and accuracy corresponding to the following:

- Average precision, recall, f1-measure, accuracy for datasets where 25% of attack datapoints removed
- Average precision, recall, f1-measure, accuracy for datasets where 50% of attack datapoints removed
- Average precision, recall, f1-measure, accuracy for datasets where 75% of attack datapoints removed
- Average precision, recall, f1-measure, accuracy for datasets where 90% of attack datapoints removed
- Average precision, recall, f1-measure, accuracy for datasets where 95% of attack datapoints removed

Questions

1. In Part 1, what size of ngrams gives the best performance? What are the tradeoffs as you change the size?
2. In Part 1, how does performance change if we use simple counts as features (i.e., 1-grams) as opposed to counts of 2-grams? What does this tell you about the role of sequences in prediction for this dataset?
3. How does performance change as a function of class prior in Part 2?

Submission

Lab reports must be in readable English and not raw dumps of log-files. You are encouraged to use the report template provided on Canvas. Please submit your lab report and all of your code in a zip/tar file on on Canvas as lab4_EID.tar.gz or lab4_EID.zip.

References

[1] G. Creech and J. Hu, "A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguousand Discontiguous System Call Patterns," in IEEE Transactions on Computers, vol. 63, no. 4, pp. 807-819, April 2014, doi: 10.1109/TC.2013.13.