COMP90073 Security Analytics

Project 2: Machine Learning Based Cyberattack Detection

[redacted]

1. Introduction

There are two main tasks as part of this project. The first task involves the detection of anomalies using unsupervised machine learning techniques whereas the second task involves generating adversarial samples against supervised learning models.

Two datasets are provided, one for each task, that contains data in the form of NetFlow records. The aim of both the tasks is to identify botnet traffic from normal traffic. Also, for Task 2, we need to choose a botnet IP address and explain how to manipulate its features in order to bypass detection.

Section 2 of the report provides an analysis of the data provided as well as a description of the pre-processing and feature generation techniques used. Section 3 and 4 discusses the Methodology and Critical Analysis for Tasks 1 and 2.

2. Dataset

Datasets for both tasks are inspected and pre-processed in the same way as described in this section.

2.1 Analysis and Inspection

Using Splunk, we were able to ingest the test data for Task 1 to inspect and visualize the distribution of various features which in turn helped further towards data preprocessing and feature engineering.

We see the top source IP address to be '**224.134.91.164**' and then so on in Figure 1. (Splunk Command - *source="test_data_ass2.csv" | top src_ip*)

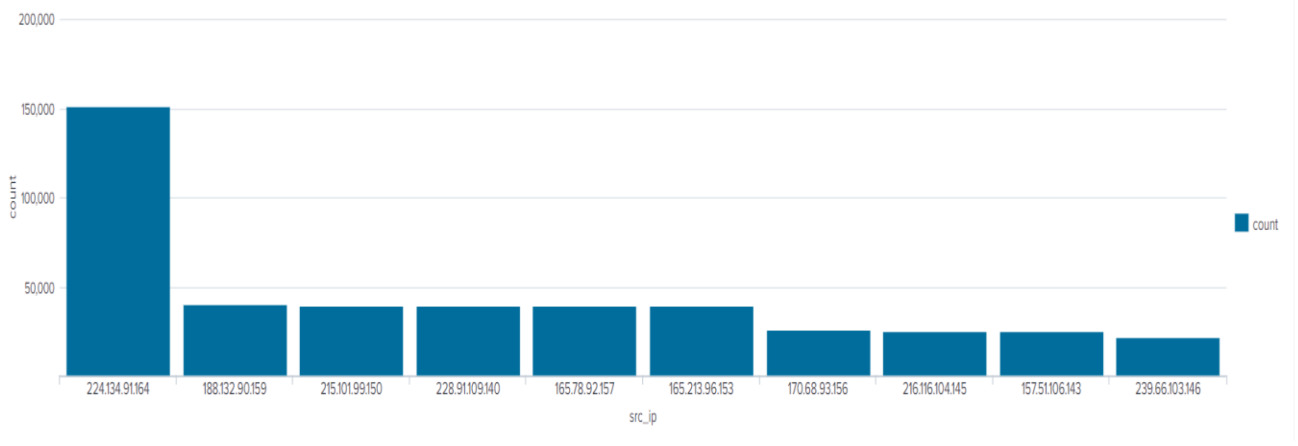

Figure 1: Top 10 source IP addresses

Similarly, we observe the top 10 destination IP addresses in Figure 2 (Splunk Command - *source="test_data_ass2.csv" | top protocol*)

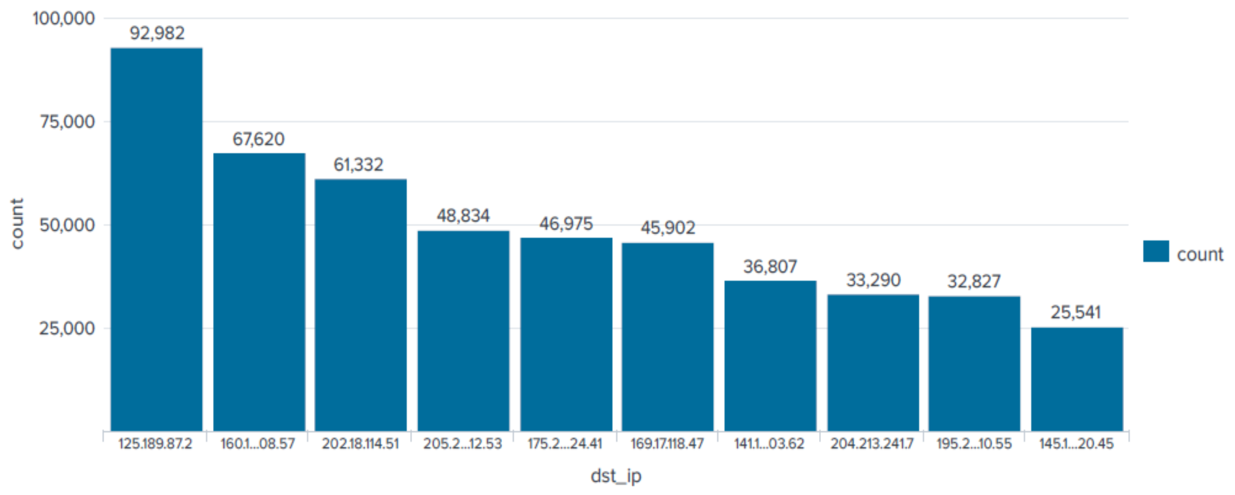

Figure 2: Top 10 destination IP addresses

We also extract the top protocols and observe that the most used protocols are TCP, ICMP and UDP. (Splunk Command - *"test_data_ass2.csv" | top protocol)*

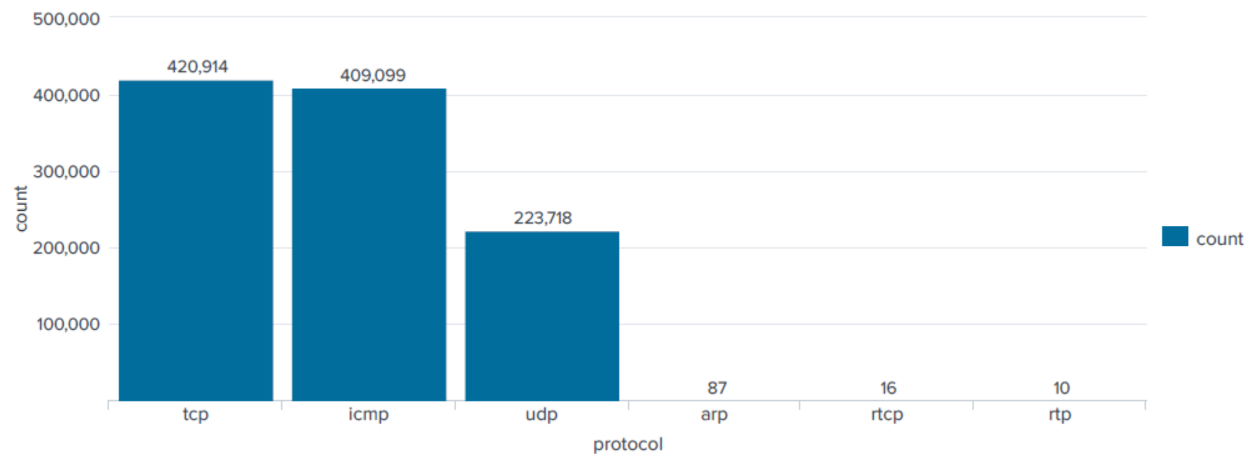

Figure 3: Top protocols

2.2 Initial Preprocessing:

- Since the datasets do not contain a column header, it is manually added according to the data description.
- On inspecting the data in Splunk, it can be seen that there are a few blank and a few 'Nan' fields as well. These are handled by filling them with -1 to ease further processing and feature generation.
- When the original NetFlow data is ingested into a Data Frame, the Date time objects are read as a string rather than as a date-time object which makes it difficult to perform operations on it.

Hence, we use the *Pandas.to_datetime()* method that helps convert a string date-time into a Python date-time object.

- For the data files that contain labels, the labels are provided in a raw format. We categorize the labels into 2 categories. All labels that contain the word 'Botnet' (regex matching) are categorized as '1' (anomalous traffic) and the rest are categorized as '0' (normal traffic).

2.3 Feature Generation & Further Processing:

The main issue with NetFlow data is that most of the information is in the form of categorical features. Trying to transform the categories into one-hot encoded columns, would result into a large sparse matrix causing memory errors [1]. This problem encourages us to extract new features based on reviewed literature.

By investigating the correlation between flows, we can find meaningful traffic patterns and node behaviors. One method is by looking at NetFlow sampling [2]. This is justified by the fact that botnets "tend to have a temporal locality behavior" [3]. To trim down the amount of data, we use a schema to sample the NetFlow traffic using a time window. We choose a time window of 2 minutes with a stride of 1 minute taking inspiration from [1]. We then preprocess the bidirectional NetFlow dataset and extract categorical and numerical characteristics that describe the dataset within the given time window.

2.3.1 Feature Set 1

The features that will be extracted should represent NetFlow communications inside a time window. To do so, the time window data are grouped by source IP addresses.

The extracted data in this feature set includes the number of unique occurrences from categorical data such as source ports, destination IPs and ports in the subgroup, as well as numerical NetFlow characteristics - duration of communication, total number of exchanged bytes, number of bytes sent by the source [1].

The five features extracted from each numerical feature include:

- Sum
- Mean
- Standard Deviation
- Maximum value
- Median

This extracts a total of 18 features which are all normalized.

2.3.2 Feature Set 2

Similar to the first set, the time window data are grouped by source IP address. The new set of features extracted here are the normalized subgroup entropy defined as

$$E = -\sum_{x_i \in X} \mathrm{p}(x_i) \log \mathrm{p}(x_i) \quad with \quad p(x_i) = \frac{\#x_i}{\#X}$$

where X is the subgroup of the source address. These features are extracted from categorical NetFlow features only (source ports, destination IPs and the destination ports) [1].

2.3.3 Feature Set 3

The third set of features is extracted by combining the features from feature sets 1 and 2 and then filtering out those features that have a high correlation. We use Pearson correlation since the pre-processed data are all continuous values. Using this we can eliminate those features that have very high correlation scores of greater than 0.8 magnitude. This allows us to only select more independent features since most of these features are a combination of the same original NetFlow data. Hence, we are able to drop 4 different highly correlated features from our combined feature set and use the remaining as the third feature set.

3. Task 1

3.1 Methodology

3.1.1 Isolation Forest Algorithm

The isolation Forest algorithm explicitly isolates anomalies rather than profiling normal instances. It takes advantage of the fact that anomalies are 'few and different', which make them more susceptible to isolation than normal points, which is why we choose this algorithm [4].

To implement the iForest algorithm we use PyOD, which is an open-source Python toolbox for performing scalable outlier detection on multivariate data [5]. The parameters that mainly need to be set are contamination (expected outlier proportion), the number of trees to build (estimators) and the sub-sampling size which is the number of samples to draw from the training data to train each base estimator. We first run the model with default parameters just as a baseline and then tune parameters accordingly using the validation set.

On training the model, we first compute the list of outlier scores of the training data itself. The higher score, the more abnormal it is. We then sort these outlier scores in descending order (say *desc_scores*) and further calculate the expected number of outlier samples (say *exp_out*), using the contamination value that we've set. Hence, the threshold outlier score can be estimated as the outlier score of that index value (*desc_scores[exp_out]*).

The classifier is then used to predict outlier scores on the validation data and for each predicted outlier score, if the score is greater than the threshold, then it is considered an anomaly, else normal. To evaluate the model, F1 measure is used between the ground truth labels and the predicted labels since accuracy as a metric can be somewhat misleading.

The model is tuned appropriately to a certain extent so as to avoid overfitting using the F1 score. The model was found to best perform with the number of estimators at 100, sub-sampling size of 256 for all feature sets, and contamination parameter as mentioned in Table 1. Outlier scores are similarly generated on the test samples after which predictions are made based on the set threshold value.

We extract the required data (index value, timestamps, source and destination IPs, source and destination ports and protocol) of all those test samples that are predicted as anomalous and write it to the CSV file.

This entire process is run similarly for all three feature sets and the results noted as seen in Table 1.

3.1.2 Cluster Based Local Outlier Factor Algorithm (CBLOF):

CBLOF defines the similarity between a point and a cluster in a statistical way that represents the probability that the point belongs to the cluster. The CBLOF score can detect outlier points that are far from any clusters and hence we choose this as our second algorithm [6].

Similarly as before, we use PyOD to implement the algorithm on all the 3 different feature sets generated. The main parameters to be set and tuned are contamination (expected outlier proportion) and the number of clusters. We first run the model with default parameters just as a baseline and then tune parameters accordingly using the validation set. Similar to the procedure followed in the previous algorithm, we train the model, extract outlier scores, find the threshold depending on the contamination factor, and then predict outlier scores on the validation set.

After fine-tuning, using F1 score as the metric, we find that **8 clusters** gives the best scores in all three feature sets and contamination values as mentioned in Table 1, per feature set. Outlier scores are similarly generated on the test samples, (using tuned parameters) after which predictions are made based on the threshold value obtained. We extract the required data of all those test samples that are predicted as anomalous and write it to the corresponding CSV file.

Table 1: F1 scores (%) for each algorithm - feature set with contamination parameter on validation data

Algorithm	**Feature Set 1**	**Feature Set 2**	**Feature Set 3**
iForest – Default Parameters	43.41% (10% contamination)	49.75% (10% contamination)	49.83% (10% contamination)
iForest – Tuned Parameters	72.82% (35% contamination)	74.35% (35% contamination)	70.74% (20% contamination)
CBLOF – Default Parameters	56.52% (10% contamination)	41.81% (10% contamination)	32.42% (10% contamination)
CBLOF – Tuned Parameters	57.33% (12.5% contamination)	75.54% (17.5% contamination)	55.02% (35% contamination)

3.2 Analysis & Inference

On analyzing the validation results, it can be seen that hyperparameter tuning, especially on the contamination parameter, makes a considerable difference in results in both algorithms when compared to the default parameters. This is because we do not know the proportion of outliers and hence have to guess through a trial and error like search method. Also, the different feature sets and algorithms peak at different contamination values due to the differences in the way the model learns and differences in data distribution.

Further, there is a general trend visible, where the performance of feature set 2, outperforms those of the other feature sets in both algorithms. Feature Set 2 is extracted from categorical NetFlow data only (IP addresses and Ports) using the normalized sub-group entropy. This seems to prove more effective on the validation data but we cannot conclude that it is the best since we do not have test labels considering this is an unsupervised problem and the distributions of data (and contamination) may be different as compared to the training and validation set.

On an overall basis, we see that iForest performs better than the CBLOF algorithm which can be attributed to the fact that the effectiveness of CBLOF highly depends on the clustering method (in this case k-means). Such methods may not be optimized for anomaly detection [6]. Also, iForest uses the isolation characteristics of iTrees which enables it to build partial models and exploit subsampling to an extent which is not feasible in other existing models which could contribute to the higher performance.

From a scaling point of view, the iForest algorithm can scale up to handle extremely large and high dimensional datasets with linear time complexity and low memory requirement as seen at execution with the large training data [4]. For the CBLOF algorithm, once the clusters are obtained, clustering-based methods need only compare any object against the clusters to determine whether the object is an outlier. This process is typically fast as well because the number of clusters are usually small compared to the total number of objects [6].

4. Task 2

4.1 Methodology

Adversarial Robustness Toolbox (ART) is a Python library supporting developers and researchers in defending Machine Learning against adversarial threats. ART provides the tools to build and deploy defenses and test them with adversarial attacks [7]. Hence, we use ART to facilitate the generation of adversarial samples in our problem.

Since we have the labels for the train, validation and test data, we can use a strong supervised classifier to discriminate botnet traffic from regular traffic. We use feature set 3 (see Section 2) to train a supervised classifier since it contains features extracted from both the numerical and categorical data of the original NetFlow records.

We initially considered using a one-class Support Vector Machine (SVM) but from inspection, found that the proportion of outliers in the training data is around **46%**. Since the training data is not too imbalanced towards either of the two classes, we use a Support Vector Classifier (SVC) instead, to classify instances as anomalous or normal.

We fit the model (wrapped into the ART toolbox) on the training data and use the validation data to tune the hyper-parameters. The metric that is optimized here again is the F1 score. We use the default L2 regularization when training the SVC to avoid overfitting. Through fine tuning, we find the best value of gamma to be 0.001. Gamma is the coefficient of the kernel used in the SVC. Using the tuned parameters, we are able to achieve F1 scores of 88.1% on the validation set and **90%** on the test set.

We make use of the Fast Gradient Method (FGM) which takes the wrapped SVC classifier as input to generate adversarial samples of the complete test data. We then generate predictions of these adversarial samples using the original SVC classifier and evaluate them to see if there is a reduction in F1 scores. As expected, we see a huge fall in the score from the original 90% to **23.57%**.

Out of the many botnet IP addresses, we randomly choose one, such that it has been correctly classified as anomalous by our original SVC discriminator but is now being misclassified as normal since we have added perturbations to it using the FGM. The randomly chosen IP address is ‘**147.148.92.184**’. We observe the original (preprocessed) feature data for the sample compared to the perturbed adversarial sample and calculate the feature-wise difference (δ) between the two. These are recorded in the corresponding CSV file.

Table 1: F1 scores (%) on the different datasets

Dataset	**F1 Score**
Training (Normal Samples)	92.79%
Validation (Normal Samples)	88.17%
Test (Normal Samples)	90.00%
Test (Adversarial Samples)	23.57%

4.2. Analysis & Inference

From Table 2, we can see that the SVC works quite well in classifying traffic as anomalous or normal in general proving to be a strong discriminator. This is credited to the fact that SVCs work relatively well as there is a clear margin of separation between the classes. It also has a regularization parameter that avoids overfitting and the model has a high generalization ability which is why we choose this classifier [8]. While the F1 score on the validation set is 88.17%, the score of 90% on the test set shows that the model is not overfit to the training instances.

The FGM is an evasion attack that aims to add minimum perturbation (δ) to the input, in order to cause the target model to misclassify. We see a huge drop in scores on evaluating the adversarial test samples as compared to the normal test samples because the adversarial samples have been manipulated using the FGM to contain some perturbation (noise). Hence, an IP address that was initially considered an anomaly, would now be able to bypass the detection system as normal. Due to the large number of such misclassifications, the performance of the model drops down to around 24% as seen in Table 2.

From the CSV generated, we see a general trend that in reducing the value of features like duration, total number of bytes between source and destination and total number of bytes sent from source to destination, the network traffic data can be manipulated in order to satisfy the modified (adversarial) features.

One of the main differences between generating adversarial samples in computer vision and in cyber-data is that in computer vision, the features are extracted by the convolutional layers and adversarial samples are generated automatically based on them. Whereas in cybersecurity data, the feature engineering needs to be manually done and then fed to the adversarial sample generator.

6. Assumptions

- Splunk is used for data inspection only as advised. All feature generation methods have been executed in Python. While Splunk has an interactive GUI, it is slow in ingesting and processing large amounts of data considering the constraints of the local machine being used.
- An assumption in task 1 is that each NetFlow record is considered as a complete event. Our model outputs all anomalous traffic over the span of the complete test data. Due to constraints on time, we are unable to analyze each anomalous event to extract its exact lifespan and type of attack.
- The CSV generated in task 2 describes the feature-wise data values after pre-processing only. Once we generate adversarial samples of the test data, it is not possible to extract the exact values

of original NetFlow records, since each feature generated is a combination of original features in some cases. Since we cannot reverse engineer to the original features, we calculate differences in the preprocessed test sample and preprocessed adversarial test sample and relate them back to the original NetFlow data only, discussing a trend rather than an exact difference.

References

[1] A. Delplace, S. Hermoso, and K. Anandita, “Cyber Attack Detection thanks to Machine Learning Algorithms,” 2020.

[2] C. Wagner, J. François, R. State, and T. Engel, “Machine learning approach for IP-flow record anomaly detection,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6640 LNCS, no. PART 1, pp. 28–39, 2011.

[3] S. García, M. Grill, J. Stiborek, and A. Zunino, “An empirical comparison of botnet detection methods,” *Comput. Secur.*, vol. 45, pp. 100–123, 2014.

[4] F. Tony Liu, K. Ming Ting, and Z.-H. Zhou, “Isolation Forest ICDM08,” *Icdm*, 2008.

[5] Y. Zhao, Z. Nasrullah, and Z. Li, “PyOD: A python toolbox for scalable outlier detection,” *J. Mach. Learn. Res.*, vol. 20, pp. 1–7, 2019.

[6] J. Han, M. Kamber, and J. Pei, “Data Mining Techniques, Third Edition,” p. 847, 2011.

[7] M.-I. Nicolae *et al.*, “Adversarial Robustness Toolbox v1.0.0,” pp. 1–34, 2018.

[8] N. H. Farhat, “Support-Vector Networks,” *IEEE Expert. Syst. their Appl.*, vol. 7, no. 5, pp. 63–72, 1992.