Network Analytics: Network Intrusion Detection using Machine Learning

Under guidance of:

Prof. Sri Krishnamurthy
Ashwin Dinoriya

Team 1

Amulya Aankul
Nand Govind Modani
Saksham Agrawal

Index

Amulya Aankul
Nand Govind Modani
Saksham Agrawal

1. Introduction

Along with the benefits, the Internet also created numerous ways to compromise the stability and security of the systems connected to it. Although static defense mechanisms such as firewalls and software updates can provide a reasonable level of security, more dynamic mechanisms such as intrusion detection systems (IDSs) should also be utilized. Intrusion detection systems are typically classified as host based or network based. A host based IDS will monitor resources such as system logs, file systems and disk resources; whereas a network based intrusion detection system monitors the data passing through the network. Different detection techniques can be employed to search for attack patterns in the data monitored. Misuse detection systems try to find attack signatures in the monitored resource. Anomaly detection systems typically rely on knowledge of normal behavior and flag any deviation from this. Intrusion detection systems currently in use typically require human input to create attack signatures or to determine effective models for normal behavior. Support for learning algorithms provides a potential alternative to expensive human input. The main task of such a learning algorithm is to discover directly from (training) data appropriate models for characterizing normal and attack behavior. The ensuing model is then used to make predictions regarding unseen data.

2. About the Data

Software to detect network intrusions protects a computer network from unauthorized users, including perhaps insiders. The intrusion detector learning task is to build a predictive model (i.e. a classifier) capable of distinguishing between ``bad'' connections, called intrusions or attacks, and ``good'' normal connections.

The 1998 DARPA Intrusion Detection Evaluation Program was prepared and managed by MIT Lincoln Labs. The objective was to survey and evaluate research in intrusion detection. A standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment, was provided. The 1999 KDD intrusion detection contest uses a version of this dataset.

Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. They operated the LAN as if it were a true Air Force environment, but peppered it with multiple attacks.

Amulya Aankul
Nand Govind Modani
Saksham Agrawal

The raw training data was about four gigabytes of compressed binary TCP dump data from seven weeks of network traffic. This was processed into about five million connection records. Similarly, the two weeks of test data yielded around two million connection records.

A connection is a sequence of TCP packets starting and ending at some well-defined times, between which data flows to and from a source IP address to a target IP address under some well-defined protocol. Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes.

Attacks fall into four main categories:

- **DOS**: denial-of-service, e.g. syn flood;
- **R2L**: unauthorized access from a remote machine, e.g. guessing password;
- **U2R**: unauthorized access to local superuser (root) privileges, e.g., various ``buffer overflow'' attacks;
- **probing**: surveillance and other probing, e.g., port scanning.

It is important to note that the test data is not from the same probability distribution as the training data, and it includes specific attack types not in the training data. This makes the task more realistic. Some intrusion experts believe that most novel attacks are variants of known attacks and the "signature" of known attacks can be sufficient to catch novel variants. The datasets contain a total of 24 training attack types, with an additional 14 types in the test data only.

Stolfo et al. defined higher-level features that help in distinguishing normal connections from attacks. There are several categories of derived features.

The ``same host'' features examine only the connections in the past two seconds that have the same destination host as the current connection, and calculate statistics related to protocol behavior, service, etc.

The similar ``same service'' features examine only the connections in the past two seconds that have the same service as the current connection.

"Same host" and "same service" features are together called time-based traffic features of the connection records.

Some probing attacks scan the hosts (or ports) using a much larger time interval than two seconds, for example once per minute. Therefore, connection records were also sorted by destination host, and features were constructed using a window of 100 connections to the same host instead of a time window. This yields a set of so-called host-based traffic features.

Unlike most of the DOS and probing attacks, there appear to be no sequential patterns that are frequent in records of R2L and U2R attacks. This is because the DOS and probing attacks involve

many connections to some host(s) in a very short period of time, but the R2L and U2R attacks are embedded in the data portions of packets, and normally involve only a single connection.

Useful algorithms for mining the unstructured data portions of packets automatically are an open research question. Stolfo et al. used domain knowledge to add features that look for suspicious behavior in the data portions, such as the number of failed login attempts. These features are called ``content'' features.

A complete listing of the set of features defined for the connection records is given in the three tables below. The data schema of the contest dataset is available in machine-readable form.

feature name	*description*	*type*
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong'' fragments	continuous
urgent	number of urgent packets	continuous

Table 1: Basic features of individual TCP connections.

feature name	*description*	*type*
hot	number of ``hot'' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised'' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	discrete
num_root	number of ``root'' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the ``hot'' list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	discrete

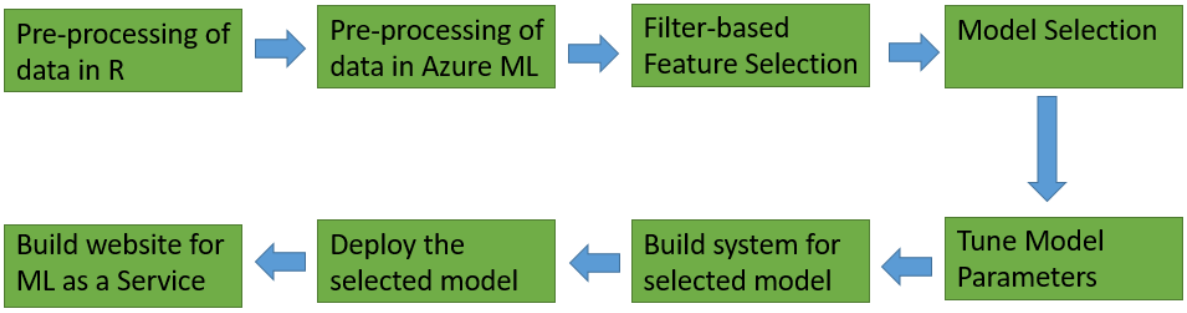
Table 2: Content features within a connection suggested by domain knowledge.

Amulya Aankul
Nand Govind Modani
Saksham Agrawal

feature name	*description*	*type*
count	number of connections to the same host as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-host connections.	
serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-service connections.	
srv_serror_rate	% of connections that have ``SYN'' errors	continuous
srv_rerror_rate	% of connections that have ``REJ'' errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Table 3: Traffic features computed using a two-second time window.

3. Flow Diagram



4. Data-Preprocessing

4.1. Assign column values to the dataset

The original KDD99 dataset does not have column names. To assign column names to the dataset so that it can be identified by Microsoft Azure ML Studio, we created a list of columns and assigned it to the datasets. We used R to achieve this transformation.

4.2. Transformation of labels into binomial classes

The dataset contains labels of different attacks like DOS, R2L, U2R and probing. Since we have to find anomalies, we transformed the data into ‘normal’ and ‘attack’. We used R to achieve this transformation.

Amulya Aankul
Nand Govind Modani
Saksham Agrawal

5. Building Model in Microsoft Azure ML Studio

Now that the data is pre-processed and clean, we will run the algorithm in Microsoft Azure ML Studio. The following steps were involved in building a model:

5.1. Dataset

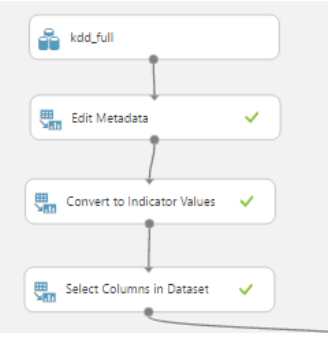
The pre-processed output from R is stored in Microsoft Azure ML cloud storage. It is stored as GenericCSV format. The input dataset contains the following files:

- kddcup.names A list of features.
- kddcup.data.gz The full data set (18M; 743M Uncompressed)
- kddcup.data_10_percent.gz A 10% subset. (2.1M; 75M Uncompressed)
- kddcup.newtestdata_10_percent_unlabeled.gz (1.4M; 45M Uncompressed)
- kddcup.testdata.unlabeled.gz (11.2M; 430M Uncompressed)
- kddcup.testdata.unlabeled_10_percent.gz (1.4M;45M Uncompressed)
- corrected.gz Test data with corrected labels.
- training_attack_types A list of intrusion types.
- typo-correction.txt A brief note on a typo in the data set that has been corrected (6/26/07)

For development purpose, we used the 10% data for training and testing. Once, we figured out the processing and model selection, we chose the full dataset for training the data.

5.2. Pre-processing

The original label column, called 'class' has many values and of string type. Each string value corresponds to a different attack. Some attacks do not have many examples in the training set and there are new attacks in the test set. to simplify this sample experiment, we build a model that does not distinguish between different types of attacks. For this purpose, we replace 'class' column with the binary column that has 1 if an activity is normal and 0 if it is an attack. The Studio provides built-in modules to ease this preprocessing step. The binarization of 'class' column is achieved by using **Metadata Editor** to change the type of 'class' column to categorical, getting binary column with **Indicator Values** module and selecting 'class-normal' column with **Project Columns** module. This sequence of steps is shown below:



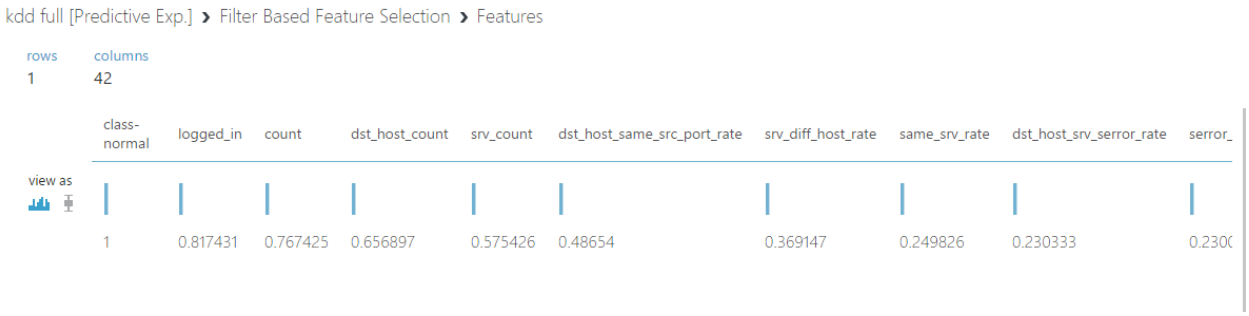
5.3. Feature Selection

Amulya Aankul
Nand Govind Modani
Saksham Agrawal

We have used Filter based feature selection for selecting the features. In general, feature selection refers to the process of applying statistical tests to inputs, given a specified output, to determine which columns are more predictive for the output. The Filter Based Feature Selection module provides multiple feature selection algorithms to choose from; you must select an appropriate method based on the kind of predictive task you are performing and the types of values in your data.

The module outputs a dataset that contains the best feature columns, as ranked by predictive power. It also outputs the names of the features and their scores from the selected metric. Since, it's an anomaly detection being used in network intrusion, we have to identify the malicious packets without compromising the speed and accuracy.

First, we selected all the 41 features and calculated the test results. After some trial and error, we selected the number of features as 15 and calculated the results. The coefficient of selection of top selected features are shown in the below visualization:



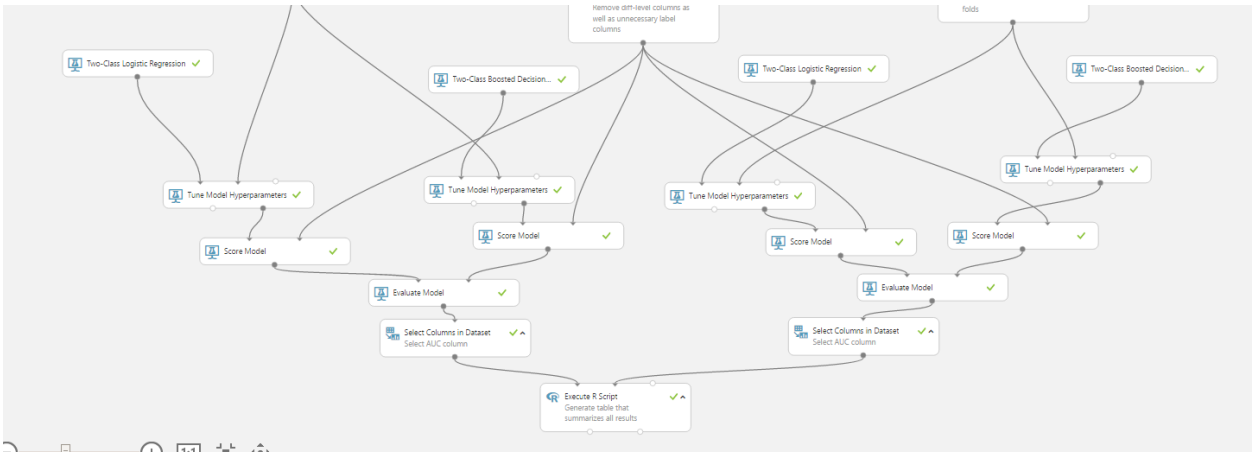
5.4. Evaluating and Selecting model

In Network Intrusion, we need both accuracy and speed. We cannot slow down the transfer of packets. At the same time, we cannot trade-off the accuracy of the system. Therefore, we evaluated several models and compared their accuracy and speed. The evaluation was done by scoring all the models and then comparing the AUC curve and Training Time of all the models in Azure ML Studio. We evaluated four well known models:

a) Two-class Logistic Regression
b) Two-class Boosted Decision Tree
c) Neural Network
d) Support Vector Machines

The workflow is shown below:

Amulya Aankul
Nand Govind Modani
Saksham Agrawal

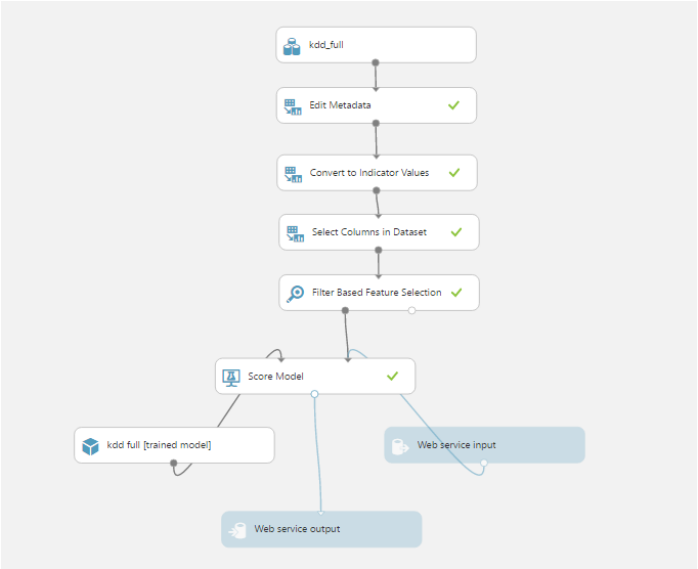


To evaluated the optimum number of features, we have also evaluated the performance of the models when all the features are used vs when only top 15 features are used. We have also used hyper-parameters to tune the results.

The performance of the models for **10% data** can be shown below:

Model	Accuracy (AUC)
Logistic Regression	0.995634
Boosted Decision Tree	0.999093
Neural Network	0.996295
Support Vector Machines	0.994526

From the above results, we conclude that Boosted Decision Tree with 15 features shows the best results. Hence, we have deployed Boosted Decision Tree – 15 features as a web services in the website.



Amulya Aankul
Nand Govind Modani
Saksham Agrawal

6. Machine Learning as a Service

Now that we have deployed the web service, we can hit the Azure ML API to implement the machine learning algorithm as a service. The input is the continuous stream of packet data and the output shows whether the packet is normal or malicious. As a product, it will run in the background and block all the packets that are malicious.

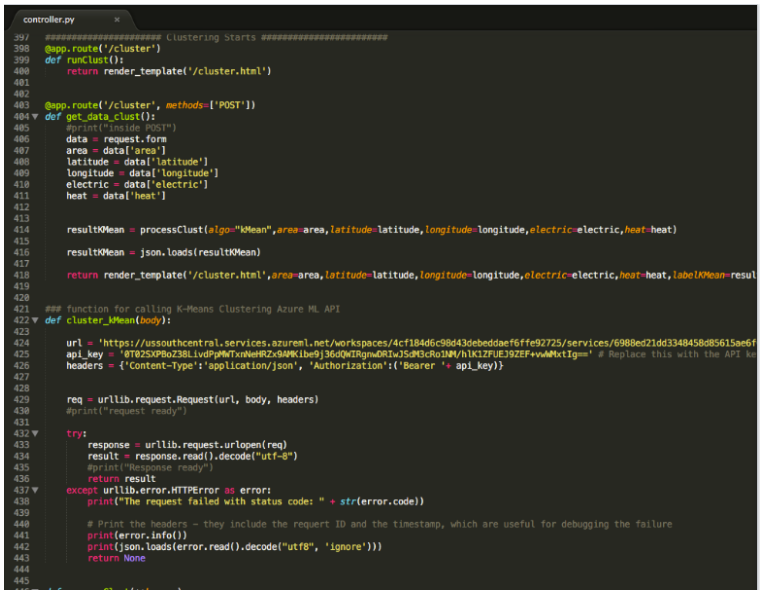
In the website, we have used HTML5, CSS3, Bootstrap and jQuery in the backend and Python Flask at the backend.

Request-Response Batch

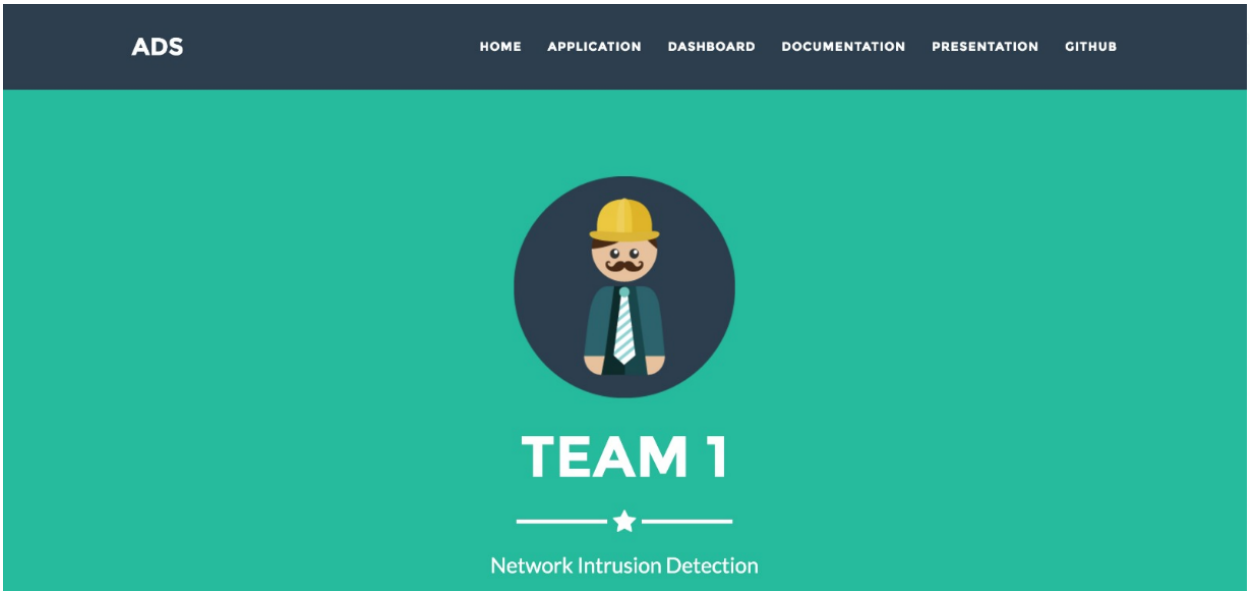
C# Python Python 3+ R

```
import urllib.request
import json

data = {
        "Inputs": {
                "input1":
                [
                    {
                            'Norm_Consumption': "1",
                            'Dew_PointF': "1",
                            'TemperatureF': "1",
                            'month': "1",
                            'Conditions': "Clear",
                            'Weekday': "1",
                            'Base_hour_Flag': "false",
                            'Wind_SpeedMPH': "1",
                            'type': "Dist_Heating",
                    }
                ],
        },
    "GlobalParameters":  {
    }
}
```



Amulya Aankul
Nand Govind Modani
Saksham Agrawal



Amulya Aankul
Nand Govind Modani
Saksham Agrawal