MLND Capstone Project Report

Machine Learning for Network Intrusion Detection Systems

Naveena Premkrishna

Capital One

Table of Contents

Definition

Project Overview

In recent years, human society has developed an increased dependency on Information and Communication Technologies. With the ever-growing use of internet, computer networks have grown leaps and bound with compounding complexities. Same time, sophisticated and diverse threats to modern computer networks and systems are also increasing. This calls for robust security mechanism and Intrusion detection systems in place to identify and combat malicious activities on computer networks.

Problem Statement

Governments and Organizations have valuable information stored on computer networks, which has made intrusion detection the most critical focus area in every CIO's office. A variety of studies have been carried out in the field of network security, intrusion detection and resolution.

While traditional intrusion detection methods have proven to be efficient at detecting intrusions based on known parameters, they are not very effective in cases involving new types of intrusions.

When an IDS uses filters and signatures to describe attack patterns, the analysis is static. This method of IDS is called signature detection (or Misuse Based Network Intrusion Detection System (MNIDS)). Signature detection is limited to the detection of known attack patterns. For the detection of unknown attacks, heuristic methods must be used. Systems that use these methods offer the possibility of detecting patterns that are not `normal'. These detection methods are called anomaly detection (Anomaly Based Network Intrusion Detection System (ANIDS)).

Machine learning can be applied in ANIDS systems, machine learning solutions can detect possible attacks in real time, so that combating measures can be taken in a timely manner.

The project will have a typical Machine Learning Project workflow

1. Explore the Dataset
2. Preprocess data
    a. Identify features that need log transformation and apply transformation
    b. Perform minmax scaling on numerical features
    c. Identify outliers if any
    d. One hot encode categorical features
    e. Perform Feature selection / Dimensionality Reduction as required
3. Train a 2 different of classifiers on the given dataset.
    a. Ensemble Methods and
    b. Deep Neural Networks
4. Compare results with the benchmark results.
5. Iterate and Fine-tune hyper-parameters to improve the accuracy and FAR.
6. Analyze confusion matrix and suggest future areas of improvements.

Metrics

Accuracy and FAR (False Alarm Rate) will the main evaluation metrics for this project. The reason for choosing these 2 metrics is as below:

1. The benchmark metrics is measured in terms of Accuracy and FAR. In order to produce comparable results, the project needs to generate similar metrics.
2. Accuracy is the most versatile and common metric used in a lot of machine learning models.
3. FAR is an appropriate metric for this domain. The main problem with Anomaly based Intrusion Detection Systems the enormous False Alarms they produce. False Alarms can be of 2 types:
    a. False Negatives: FNs represents attacks that go undetected. These are very expensive and the very purpose of Intrusion Detection Systems is to protect the networks from attacks.
    b. False Positives: FPs exhausts (read as frustrates) the support crew who are on-call, guarding the many corporate and Government networks across the world.

    Hence both these kind of False Alarms needs to be minimized. This makes False Alarm Rate a vital metrics for this domain.

Accuracy

Accuracy is the rate of the correctly classified records to all the records, whether correctly or incorrectly classified
Accuracy = (TP + TN) / (TP + TN + FP + FN)
have used, the `sklearn.metrics.accuracy_score` method to calculate the accuracy of predictions.

False Alarm Rate (FAR)

FAR is the average ratio of the misclassified to classified records either normal or abnormal
FPR (False Positive Rate) = FP / (FP + TN)
FNR (False Negative Rate) = FN / (FN + TP)
FAR (False Alarm Rate) = (FPR + FNR) / 2

I have used, `sklearn.metrics.confusion_matrix` to obtain true positives, true negatives, false positives and false negatives.

By definition a confusion matrix C is such that $C_{(i, j)}$ is equal to the number of observations known to be in group i but predicted to be in group j. Thus in binary classification, the count of true negatives is $C_{(0,0)}$, false negatives is $C_{(1,0)}$, true positives is $C_{(1,1)}$ and false positives is $C_{(0,1)}$.[3] With the 4 values obtained from confusion matrix, I calculate the False Alarm Rate (FAR) with formulas mentioned above.

Confusion Matrix per Attack Type:

In addition to Accuracy and FAR, Confusion Matrix per attack type will be generated and compared.

Analysis

Data Exploration

This project will use the UNSW-NB15 dataset to build a model to detect network anomalies and intrusions. The UNSW-NB15 data set was created using an IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) (Australian Center for Cyber Security (ACCS), 2014) to generate a hybrid of the realistic modern normal activities and the synthetic contemporary attack behaviors from network traffic.

The details of the UNSW-NB15 data set are published in following the papers:

Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set).
"Military Communications and Information Systems Conference (MilCIS), 2015. IEEE, 2015.

Moustafa, Nour, and Jill Slay. "The evaluation of Network Anomaly Detection Systems:
Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data
set." Information Security Journal: A Global Perspective (2016): 1-14.

Feature Description:

On README.md, follow the instructions under “Setting up the project” to setup the project. Then execute the below command for data exploration. The command performs a brief examination of the dataset and generates 2 csv files explore-cat-features.csv and explore-num-features.csv.

```
python -c "from src.exec import *; data_exploration()"
```

Table 1: Features of the UNSW-NB15 dataset (citation: https://www.researchgate.net/publication/312184006)

	No.	Name	Type	Description
Basic *Features*	1	proto	nominal	Transaction protocol
	2	state	nominal	Indicates to the state and its dependent protocol, e.g. ACC, CLO, *CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state)*
	3	dur	Float	Record total duration
	4	sbytes	Integer	Source to destination transaction bytes
	5	dbytes	Integer	Destination to source transaction bytes
	6	sttl	Integer	Source to destination time to live value
	7	dttl	Integer	Destination to source time to live value
	8	sloss	Integer	Source packets retransmitted or dropped
	9	dloss	Integer	Destination packets retransmitted or dropped
	10	service	nominal	http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used *service*
	11	Sload	Float	Source bits per second
	12	Dload	Float	Destination bits per second
	13	Spkts	integer	Source to destination packet count
	14	Dpkts	integer	Destination to source packet count
Content *Features*	15	swin	integer	Source TCP window advertisement value
	16	dwin	integer	Destination TCP window advertisement value
	17	stcpb	integer	Source TCP base sequence number
	18	dtcpb	integer	Destination TCP base sequence number
	19	smeansz	integer	Mean of the ?ow packet size transmitted by the src
	20	dmeansz	integer	Mean of the ?ow packet size transmitted by the dst
	21	trans_depth	integer	Represents the pipelined depth into the connection of http *request/ response transaction*
	22	res_bdy_len	integer	Actual uncompressed content size of the data transferred from *the serverís http service.*
Time *Features*	23	Sjit	Float	Source jitter (mSec)
	24	Djit	Float	Destination jitter (mSec)
	25	Sintpkt	Float	Source interpacket arrival time (mSec)
	26	Dintpkt	Float	Destination interpacket arrival time (mSec)
	27	tcprtt	Float	TCP connection setup round-trip time, the sum of ísynackí and *iackdatí.*
	28	synack	Float	TCP connection setup time, the time between the SYN and the *SYN_ACK packets.*
	29	ackdat	Float	TCP connection setup time, the time between the SYN_ACK and *the ACK packets.*
	30	is_sm_ips_ports	Binary	If source (1) and destination (3)IP addresses equal and port *numbers (2)(4) equal then, this variable takes value 1 else 0*

Additional Generated Features	31	ct_state_ttl	Integer	No. for each state (6) according to specific range of values for source/ destination time to live (10) (11).
	32	ct_flw_http_mthd	Integer	No. of flows that has methods such as Get and Post in http service.
	33	is_ftp_login	Binary	If the ftp session is accessed by user and password then 1 else 0.
	34	ct_ftp_cmd	integer	No of flows that has a command in ftp session.
	35	ct_srv_src	integer	No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).
	36	ct_srv_dst	integer	No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).
	37	ct_dst_ltm	integer	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
	38	ct_src_ ltm	integer	No. of connections of the same source address (1) in 100 connections according to the last time (26).
	39	ct_src_dport_ltm	integer	No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
	40	ct_dst_sport_ltm	integer	No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).
	41	ct_dst_src_ltm	integer	No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26).
Labelled Features	42	attack_cat	nominal	The name of each attack category. In this data set , nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms
	43	Label	binary	0 for normal and 1 for attack records

Exploration summary:

Exploration summary of the dataset's categorical and numeric features:

1. Total number of records in the training dataset: 175341
2. Total number of records in testing dataset: 82332
3. Total number of normal records: 56000
4. Total number of attack records: 119341
5. Percentage of attack records: 68%
6. Some features of the dataset have a very high standard deviation. For example, sbytes, dbytes, rate, sload, dload, sjit, stcpb, dtcpb and response_body_len. A low standard deviation indicates that the data points tend to be close to the mean of the set, while a high standard deviation indicates that the data points are spread out over a wider range of values[1]. These features may need log transformation.
7. The 'proto' feature has 133 unique values. Performing a one shot encoding will drastically increase the number of features.

Table 2: Cursory Investigation of the UNSW-NB15 dataset's categorical features

	count	unique	top	freq
proto	175341	133	tcp	79946
service	175341	13	-	94168
state	175341	9	INT	82275

Table 3: Cursory Investigation of the UNSW-NB15 dataset's numeric features

	count	mean	std	min	25%	50%	75%	max
dur	175,341.00	1.36	6.48	0.00	0.00	0.00	0.67	60.00
spkts	175,341.00	20.30	136.89	1.00	2.00	2.00	12.00	9,616.00
dpkts	175,341.00	18.97	110.26	0.00	0.00	2.00	10.00	10,974.00
sbytes	175,341.00	8,844.84	174,765.64	28.00	114.00	430.00	1,418.00	12,965,233.00
dbytes	175,341.00	14,928.92	143,654.22	0.00	0.00	164.00	1,102.00	14,655,550.00
rate	175,341.00	95,406.19	165,400.98	0.00	32.79	3,225.81	125,000.00	1,000,000.00
sttl	175,341.00	179.55	102.94	0.00	62.00	254.00	254.00	255.00
dttl	175,341.00	79.61	110.51	0.00	0.00	29.00	252.00	254.00
sload	175,341.00	73,454,033.19	188,357,447.00	0.00	13,053.34	879,674.75	88,888,888.00	5,988,000,256.00
dload	175,341.00	671,205.57	2,421,312.39	0.00	0.00	1,447.02	27,844.87	22,422,730.00
sloss	175,341.00	4.95	66.01	0.00	0.00	0.00	3.00	4,803.00
dloss	175,341.00	6.95	52.73	0.00	0.00	0.00	2.00	5,484.00
sinpkt	175,341.00	985.98	7,242.25	0.00	0.01	0.28	55.16	84,371.50
dinpkt	175,341.00	88.22	987.09	0.00	0.00	0.01	51.05	56,716.82
sjit	175,341.00	4,976.25	44,965.85	0.00	0.00	0.00	2,513.30	1,460,480.02
djit	175,341.00	604.35	4,061.04	0.00	0.00	0.00	114.99	289,388.27
swin	175,341.00	116.26	127.00	0.00	0.00	0.00	255.00	255.00
stcpb	175,341.00	969,250,421.91	1,355,264,249.26	0.00	0.00	0.00	1,916,651,334.00	4,294,958,913.00
dtcpb	175,341.00	968,877,027.07	1,353,999,546.23	0.00	0.00	0.00	1,913,674,673.00	4,294,881,924.00
dwin	175,341.00	115.01	126.89	0.00	0.00	0.00	255.00	255.00
tcprtt	175,341.00	0.04	0.08	0.00	0.00	0.00	0.07	2.52
synack	175,341.00	0.02	0.04	0.00	0.00	0.00	0.02	2.10
ackdat	175,341.00	0.02	0.04	0.00	0.00	0.00	0.04	1.52
smean	175,341.00	136.75	204.68	28.00	57.00	73.00	100.00	1,504.00
dmean	175,341.00	124.17	258.32	0.00	0.00	44.00	89.00	1,458.00
trans_depth	175,341.00	0.11	0.78	0.00	0.00	0.00	0.00	172.00
response_body_len	175,341.00	2,144.29	54,207.97	0.00	0.00	0.00	0.00	6,558,056.00
ct_srv_src	175,341.00	9.31	10.70	1.00	2.00	5.00	12.00	63.00
ct_state_ttl	175,341.00	1.30	0.95	0.00	1.00	1.00	2.00	6.00
ct_dst_ltm	175,341.00	6.19	8.05	1.00	1.00	2.00	7.00	51.00
ct_src_dport_ltm	175,341.00	5.38	8.05	1.00	1.00	1.00	5.00	51.00
ct_dst_sport_ltm	175,341.00	4.21	5.78	1.00	1.00	1.00	3.00	46.00
ct_dst_src_ltm	175,341.00	8.73	10.96	1.00	1.00	3.00	12.00	65.00

is_ftp_login	175,341.00	0.01	0.13	0.00	0.00	0.00	0.00	4.00
ct_ftp_cmd	175,341.00	0.01	0.13	0.00	0.00	0.00	0.00	4.00
ct_flw_http_mthd	175,341.00	0.13	0.70	0.00	0.00	0.00	0.00	30.00
ct_src_ltm	175,341.00	6.96	8.32	1.00	2.00	3.00	9.00	60.00

Exploratory Visualization

On README.md, follow the instructions under "Setting up the project" to setup the project. Then execute the below command for data visualization. The command creates visualizations of the dataset and generates 2 png files in the histogram_distribution.png and initial_training_heatmap.csv in the result folder.

```
python -c "from src.exec import *; data_visualization()"
```

Pairwise Correlation Matrix Heatmap:

Pairwise correlation matrix visualized in the form of a heatmap, shows the dependence or association between each pair of features. For this dataset, a very low correlation can be observed from the heatmap between pairs of features.

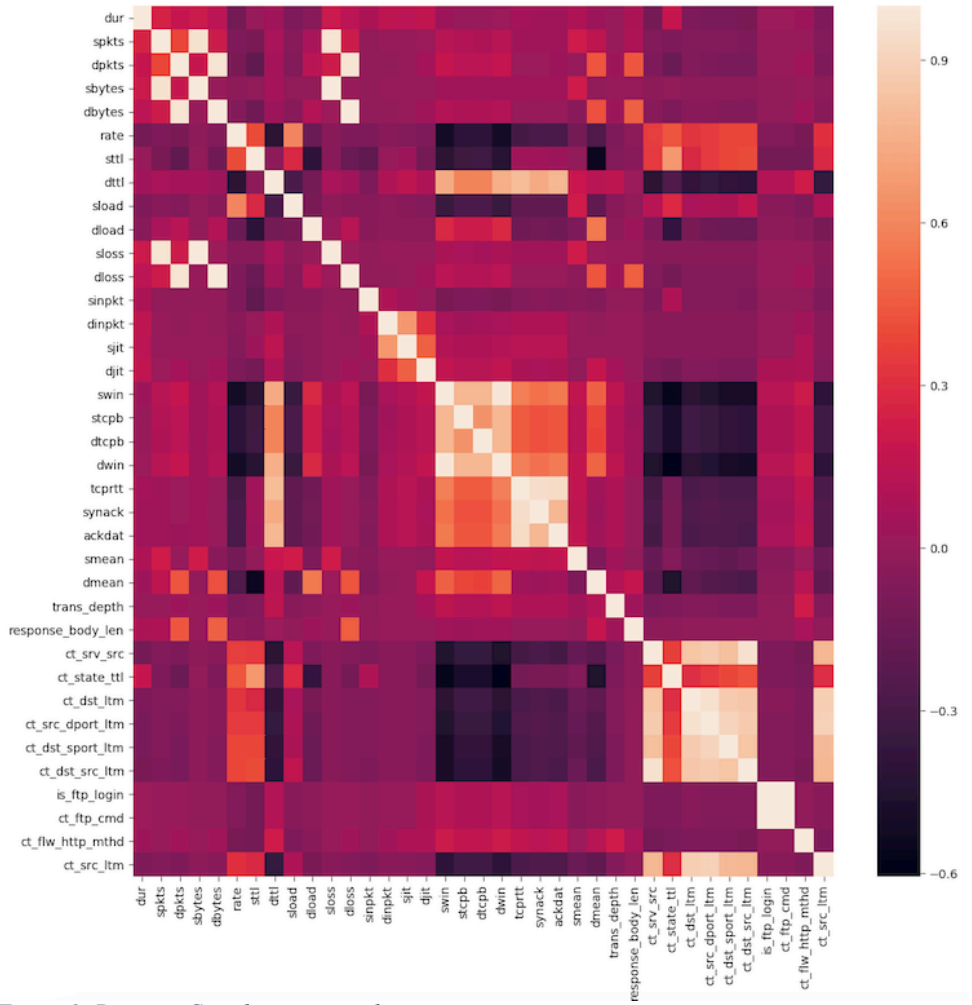

Figure 1: Pairwise Correlation matrix heatmap

Distribution and Histogram:

Histograms and distribution curves of numeric data gives a rough sense of the density of the underlying distribution of the data. It also help determine if logarithmic transformation is required for a feature. For highly-skewed feature distributions, it is common practice to apply logarithmic transformation on the data so that the very large and very small values do not negatively affect the performance of a learning algorithm. Visuals are available in the "/results/histogram_distribution.png" file. From the visuals it is evident that most numeric features need logarithmic transformation. Here are some sample skewed histogram and distribution curves.

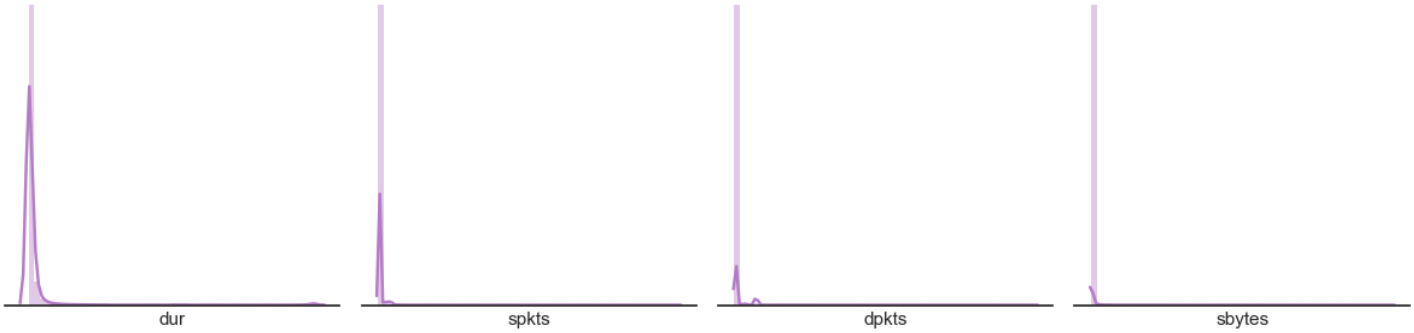

Figure 2: Histogram and distribution curves for some sample skewed features.

Algorithms and Techniques

The characteristics of the problem and problem domain needs to be considered before selecting the algorithms and techniques for training a model.

1. The chosen dataset has a large number of records (175341 training and 82332 testing).
2. There are no training time and memory foot print constraints
3. The problem is a two-class classification problem
4. Accuracy is a very important metric that would determine the performance of the model.
5. Most features are numeric and some are categorical. There are no test features.
6. The accompanying research paper has published some benchmark models and metrics. According to those benchmark models, Decision Tree and Logistic Regression has the best performance.

Model selection process involves some level of trial and error. There is no one-size-fits-all solution. However, all these characteristics of the problem domain listed above can help narrow down the selection process. I will be training 2 models on the dataset and compare the results for this capstone assignment, one based on Decision Tree and another based on Logistic Regression.

Model 1 – RandomForest:

Ensemble algorithm based on randomized decision trees. Ensemble methods are techniques that create multiple models and then combine them to produce improved results. Ensemble methods usually produces more accurate solutions than a single model would[2]. The data science blog post at this link (http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/) has an excellent explanation of Random Forest Algorithm. In this article the author explains how Random Forest algorithm works in the form of a psudocode.

Random Forest construction pseudocode:

1. Randomly select k features from total m features. Where k << m
2. Among the k features, calculate the node d using the best split point.
3. Split the node into daughter nodes using the best split.

4. Repeat 1 to 3 steps until "l" number of nodes has been reached.
5. Build forest by repeating steps 1 to 4 for "n" number times to create "n" number of trees.

Random forest prediction pseudocode:

1. Take the test features and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)
2. Calculate the votes for each predicted target.
3. Consider the high voted predicted target as the final prediction from the random forest algorithm. This concept of voting is known as majority voting.

Model 2 – Deep Neural Network:

Logistic Regression is the building block of all that constitutes Deep Learning. I will be training a DNN based classification model. DNN models are time and resource consuming. My justification for choosing this DNN is to see what levels of accuracy can be achieved with more time and resources. This MIT news blog post at this link (http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414) has a simple explanation of how Neural Networks works without going into the details of the mathematics behind it.

Modeled loosely on the human brain, a neural net consists of thousands or even millions of simple processing nodes that are densely interconnected. Most of today's neural nets are organized into layers of nodes, and they're "feed-forward," meaning that data moves through them in only one direction. An individual node might be connected to several nodes in the layer beneath it, from which it receives data, and several nodes in the layer above it, to which it sends data.

To each of its incoming connections, a node will assign a number known as a "weight." When the network is active, the node receives a different data item — a different number — over each of its connections and multiplies it by the associated weight. It then adds the resulting products together, yielding a single number. If that number is below a threshold value, the node passes no data to the next layer. If the number exceeds the threshold value, the node "fires," which in today's neural nets generally means sending the number — the sum of the weighted inputs — along all its outgoing connections.

When a neural net is being trained, all of its weights and thresholds are initially set to random values. Training data is fed to the bottom layer — the input layer — and it passes through the succeeding layers, getting multiplied and added together in complex ways, until it finally arrives, radically transformed, at the output layer. During training, the weights and thresholds are continually adjusted until training data with the same labels consistently yield similar outputs.

Benchmark

The accompanying research paper authored by the creators of the UNSW-NB15 dataset has outlined some benchmark models and metrics. These will serve as the benchmark for this project.

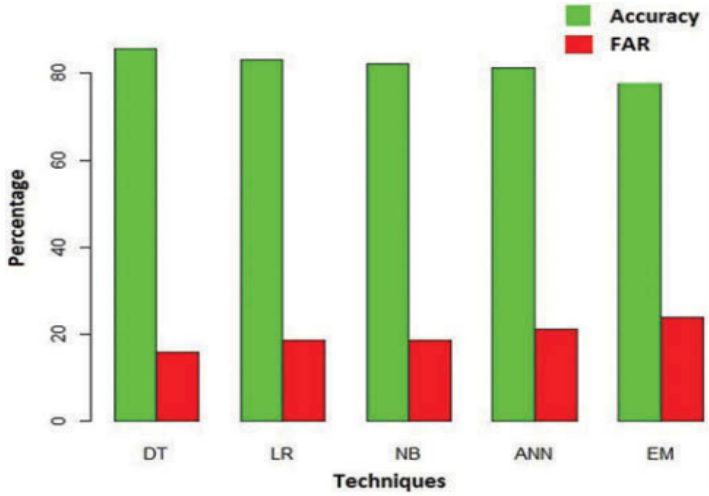

Figure 3: Benchmark metrics visualization

	Techniques	Accuracy	FAR
1	Decision Tree	85.56	15.78
2	Logistic Regression	83.15	18.48
3	Naïve Bayes	83.07	18.56
4	Artificial Neural Networks	81.34	21.13
5	EM Clustering	78.47	23.79

Methodology

Data Preprocessing

Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured. Fortunately, for this dataset, there are no invalid or missing entries we must deal with, however, there are some qualities about certain features that must be adjusted. This preprocessing can help tremendously with the outcome and predictive power of nearly all learning algorithms.

Logarithmic Transformation:

A dataset may sometimes contain features whose values tend to lie near a single number, but will also have a non-trivial number of vastly larger or smaller values than that single number. Algorithms can be sensitive to such distributions of values and can underperform if the range is not properly normalized. With the UNSW-NB15 dataset almost all features fit this description, which is also visible in the distribution curves included in Data Visualization section of this report.

For highly-skewed feature distributions it is common practice to apply a logarithmic transformation on the data so that the very large and very small values do not negatively affect the performance of a learning algorithm. Using a logarithmic transformation significantly reduces the range of values caused by outliers. Care must be taken when applying this transformation however: The logarithm of 0 is undefined, so the values are translated by a small amount above 0 to apply the logarithm successfully.

Outliers Removal:

Detecting outliers in the data is extremely important in the data preprocessing step of any analysis. The presence of outliers can often skew results which take into consideration these data points. There are many "rules of thumb" for what constitutes an outlier in a dataset. For this project I used Tukey's Method for identifying outliers: An outlier step is calculated as 1.5 times the interquartile range (IQR). A data point with a feature that is beyond an outlier step outside of the IQR for that feature is considered abnormal.

Even though outliers are often bad data points, outliers should be investigated carefully. Often they contain valuable information about the process under investigation or the data gathering and recording process. Before considering the possible elimination of these points from the data, we need to understand why they appeared and whether it represents an anomaly or attack related network traffic pattern and so, it is likely that similar values will continue to appear.

Upon applying Tukey's Method to identify Outliers, I found that 125225 records fall into the outlier range. This is more than half of the dataset. So, I decided not to remove outliers from the dataset and to use all records to train the model.

On README.md, follow the instructions under "Setting up the project" to setup the project. Then execute the below command to identify outliers. The command prints the total number of outliers identified using Tukey's method.

```
python -c "from src.exec import *; identify_outliers()"
```

Min Max Scaling

In addition to performing transformations on features that are highly skewed, it is often good practice to perform some type of scaling on numerical features. Applying a scaling to the data does not change the shape of each feature's distribution however, normalization ensures that each feature is treated equally when applying supervised learners.

I have used the sklearn.preprocessing.MinMaxScaler to apply scaling to all the numeric features of the dataset.

Factorizing Categorical Features

There are 3 non numeric features on the dataset, proto, service and state. There are 133, 13 and 9 unique values respectively on each of these features. Typically, learning algorithms expect input to be numeric, which requires that non-numeric features (called *categorical variables*) be converted. One popular way to convert categorical variables is by using the **one-hot encoding** scheme. One-hot encoding creates a *"dummy"* variable for each possible category of each non-numeric feature. But this will increase the number of features on the dataset.

This project's dataset has a total of 155 unique values in all 3 categorical features combined. One-hot encoding will increase the features by 155. So, I am using another method of encoding called Factorizing, which simply assigns a number to each category. This is not the best encoding method because Categorical variable may not be ordinal or have an order. However, for this dataset's case, factorize encode is the best of the 2 options. I have used `pandas.factorize()` method to perform the encoding.

Dimensionality Reduction

Now that the data has been scaled to a more normal distribution and has had any necessary outliers removed, we can now apply PCA to the pre-processed data to discover which dimensions about the data best maximize the variance of features involved. In addition to finding these dimensions, PCA will also report the *explained variance ratio* of each dimension — how much variance within the data is explained by that dimension alone. Note that a component (dimension) from PCA can be considered a new "feature" of the space, however it is a composition of the original features present in the data.

On README.md, follow the instructions under "Setting up the project" to setup the project. Then execute the below command to pre-process the dataset. The command creates a csv file called explained_variance.csv.

```
python -c "from src.exec import *; data_preprocessing()"
```

I have used `sklearn.decomposition.PCA` to apply PCA transformation to the dataset. I have used 42 dimensions, the same number of dimensions as the features. Upon analyzing the explained variance ratio of each of the 42 dimensions, the first 3 dimensions maximizes the variance of the features. Also the first 16 dimensions entirely captures all the variance in the dataset. So, I have reduced the dataset to 16 dimensions using PCA transformation

Table 4: Explained Variance of the top 16 dimensions

Dimensions	Explained Variance
Dimension 1	0.9848
Dimension 2	0.0102
Dimension 3	0.0013
Dimension 4	0.0008
Dimension 5	0.0006
Dimension 6	0.0004
Dimension 7	0.0003
Dimension 8	0.0003
Dimension 9	0.0002

Dimension 10	0.0002
Dimension 11	0.0001
Dimension 12	0.0001
Dimension 13	0.0001
Dimension 14	0.0001
Dimension 15	0.0001
Dimension 16	0.0001

Implementation

As discussed in the algorithms and techniques section of this paper, I will be training 2 models using Random Forest Ensemble and Multi-layer Perceptron to compare the results

Random Forest Ensemble:

I have used the sklearn.ensemble.RandomForestClassifier, class to train a model on the UNSW-NB15 training dataset. For this model I have used the default parameters of the classifier. This step is fairly simple and straight forward. The data was preprocessed, clean and ready for training. I only had to define random state to produce consistent results between multiple runs.

This Random Forest model reported an Accuracy of 84.59% and a False Alarm Rate of 16.8%. It took around 7 seconds to train and less than 1 second to predict.

On README.md, follow the instructions under "Setting up the project" to setup the project. Then execute the below command to train a simple Random Forest model. The command trains a model on the training dataset and generates metrics on the test dataset and records the results in the csv file called metrics.csv in 'results' folder.

```
python -c "from src.exec import *; train_random_forest_model()"
```

Multi-Layer Perceptron:

I have used Keras Sequential Model with Tenserflow backend to construct and train the Mutilayer Perceptron Model. The network has 3 layers.

1. A regular densely-connected neural network layer with 256 units. With 'relu' activiation. Relu stands for Rectified Linear Unit. It is the most popular activation function in Neural Networks. It works most of the time as a general approximator.
2. A Dropout layer - In machine learning, regularization is way to prevent over-fitting. Regularization reduces over-fitting by adding a penalty to the loss function such that it does not learn interdependent set of features weights. Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons. I have used 40% dropout rate which means 40% of the unit will be dropped out during training.
3. Another densely connected NN layer with 2 units representing the binary nature of the label (attack and normal)

To compile a Sequential Model, 3 parameters needs to be passed.

1. Loss function: Since this is a binary classification problem, I have used the binary_crossentrophy loss function.
2. Optimizer: AdaGrad is a popular optimization method that allows different step sizes for different features. It increases the influence of rare but informative features.
3. Metrics: As discussed in the metrics section of this paper, accuracy and False Alarm Rates will be the 2 primary metrics to evaluate the model.

Some challenges faced:

1. Accuracy is available as an out-of-the-box metrics function in keras, whereas, FAR is not. So had to write a custom metrics functions to compute false positives, false negatives, true positives and true negatives, which I later use to compute FAR after the training.

2. Since I had very limited experience training neural networks, it was challenge to determine the input_shape parameter for the first dense layer and the number of units on the last dense layer. But after a few trial and error and google search, I was able to initialize a basic NN with the right set of inputs.

This MLP model reported an accuracy of 82.07% and FAR of 19.54%. It took around 5 seconds to train and 2 second to evaluate.

On README.md, follow the instructions under "Setting up the project" to setup the project. Then execute the below command to train a Multilayer Perceptron model. The command trains a model on the training dataset and generates metrics on the test dataset and records the results in the csv file called metrics.csv in 'results' folder.

```
python -c "from src.exec import *; train_mlp_model()"
```

Refinement

This section will elaborate the improvements and refinements I have made on the 2 data models to achieve optimal performance.

Random Forest Ensemble:

When evaluating different hyperparameters for estimators, there is a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally. This way, knowledge about the test set can "leak" into the model and evaluation metrics no longer report on generalization performance. To solve this problem, yet another part of the dataset can be held out as a so-called "validation set". However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.
A solution to this problem is a procedure called cross-validation. In the basic cross-validation approach, called k-fold CV, the training set is split into k smaller sets (or 'folds'). For each of the k "folds":

1. A model is trained using k-1 of the folds as training data;
2. the resulting model is validated on the remaining part of the data

The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data.

Instead of manually adjusting/tuning the parameters of the classifier to find the optimal set of parameters, I have used sklearn.model_selection.GridSearchCV class to perform an exhaustive search over specified set of parameter values for Random Forest Classifier. Below is the list of hyperparameters tuned/searched to find the optimal set.

1. Maximum Features:
    a. Sqrt
    b. Log2
2. Minimum Sample Split:
    a. 2, 8, 14 and 16
3. Criterion:
    a. Gini
    b. Entrophy
4. Crossvalidation:
    a. 5

Contrary to expectations, the Grid search is not able to find better parameters than the default Random Forest parameters. The reported metrics are slightly lower than the metrics reported by the Random Forest Model trained with default parameter values. The reason could be loss of data for validation. Even though cross validation does not waste too much data, it still have some data loss resulting to decreased performance.

This Refined Random Forest model reported an Accuracy of 83.95% and a False Alarm Rate of 17.62%. It took around 15 minutes to train and less than 5 second to predict.

On README.md, follow the instructions under "Setting up the project" to setup the project. Then execute the below command to train a simple Random Forest model. The command trains and tunes a model on the training dataset and generates metrics on the test dataset and records the results in the csv file called metrics.csv in 'results' folder.

```
python -c "from src.exec import *; train_random_forest_model_refined()"
```

Multi-Layer Perceptron:

To refine the MPL model, I had to manually adjust the below list of hyperparameters and finally arrived at the best performing network settings.

1. Loss function:
    a. mean_squared_error,
    b. categorical_crossentrophy
    c. binary_corssentrophy
2. Optimizer:
    a. Adam
    b. Adamax
    c. Adagrad
3. Network layers
    a. Multiple sets of dense and dropout layers
4. Drop-out rate
    a. Ranging from .2 to .6
5. Activation
    a. Relu and softmax
6. Batch size
    a. Ranging from 1000 to 5000
7. Epochs
    a. Ranging from 2 to 500
8. Validation Size
    a. Ranging from 0.1 to 0.4
9. Check pointer Monitor parameter
    a. Loss, val_loss, acc, val_acc
10. Early stopping patience parameter
    a. Ranging from 50 to 75

The optimized set of Hyperparameters are as below:

1. Loss function: binary_corssentrophy
2. Optimizer: Adagrad
3. Network layers: a sandwich of 8 sets of Dense and Dropout layer with 512, 384, 256, 128, 64, 32, 16, 8 units and finally a 2 unit Dense layer at the end
4. Dropout Rate: 0.6
5. Activation: 'softmax' for the final dense layer and 'relu' for all the other dense layers.
6. Batch Size: 900
7. Epochs: 750
8. Validation Size: 0.3
9. Check pointer Monitor parameter: loss
10. Early stopping patience parameter: 75

This Refined MLP model reported an accuracy of 89.91% and FAR of 10.46%. It roughly took around 45 minutes to train and 5 second to evaluate test data.

On README.md, follow the instructions under "Setting up the project" to setup the project. Then execute the below command to train a Multilayer Perceptron model. The command trains and refines a model on the training dataset and generates metrics on the test dataset and records the results in the csv file called metrics.csv in 'results' folder.

```
python -c "from src.exec import *; train_mlp_model_refined()"
```

Results

Model Evaluation and Validation

This is the benchmark metrics from the benchmark section of this paper.

	Techniques	Accuracy	FAR
1	Decision Tree	85.56	15.78
2	Logistic Regression	83.15	18.48
3	Naïve Bayes	83.07	18.56
4	Artificial Neural Networks	81.34	21.13
5	EM Clustering	78.47	23.79

These are the results obtained from the models trained as part of this project

	Techniques	Accuracy	FAR
1	Random Forest Ensemble	85.56	15.78
2	Random Forest Ensemble – refined using grid search	83.15	18.48
3	Multilayer Perceptron	83.95	17.62
4	Multilayer Perceptron – refined manually	89.91	10.46

Clearly the refined multilayer perceptron has outperformed all other models including the benchmark models.

Justification

This model has been specifically trained on data generated from a network in Australian Centre for Cyber Security (ACCS). The model has learnt the specific traffic patterns of this particular network. The methods and techniques applied to train the model can be used on other network traffic datasets with similar features. However the model itself cannot be used on other datasets, because, each network is unique with its own topology and traffic patterns.

So, to justify the efficiency of the refined MLP model, I took one hundred thousand records from one of the raw datasets and predicted the labels using the refined MLP model. **The model produced an accuracy of 83.16% and FAR was at 32.98%.** This is slightly lower that the models results on test dataset. There could be 2 reasons for the degraded performance on the raw dataset.

1. The raw dataset was missing the 'rate' parameter found on both training and testing datasets. So I had to create another MPL model with the same set of hyperparametes as the refined MPL model but without the 'rate' feature. This information loss due to the missing parameter could have caused this degradation in performance.
2. The raw dataset, unlike the training and test dataset, could contain noise resulting in slightly lower performance.

On README.md, follow the instructions under "Setting up the project" to setup the project. Then execute the below command to run predictions on 100000 records from the raw Dataset. Make sure to downloas the UNSW-NB15_4.csv file from UNSW-NB15 dataset link and place it in the repo under '/dataset' folder.

```
python -c "from src.exec import *; predict_from_raw_dataset()"
```

Conclusion

Free-Form Visualization

Below is a visualization of the findings.

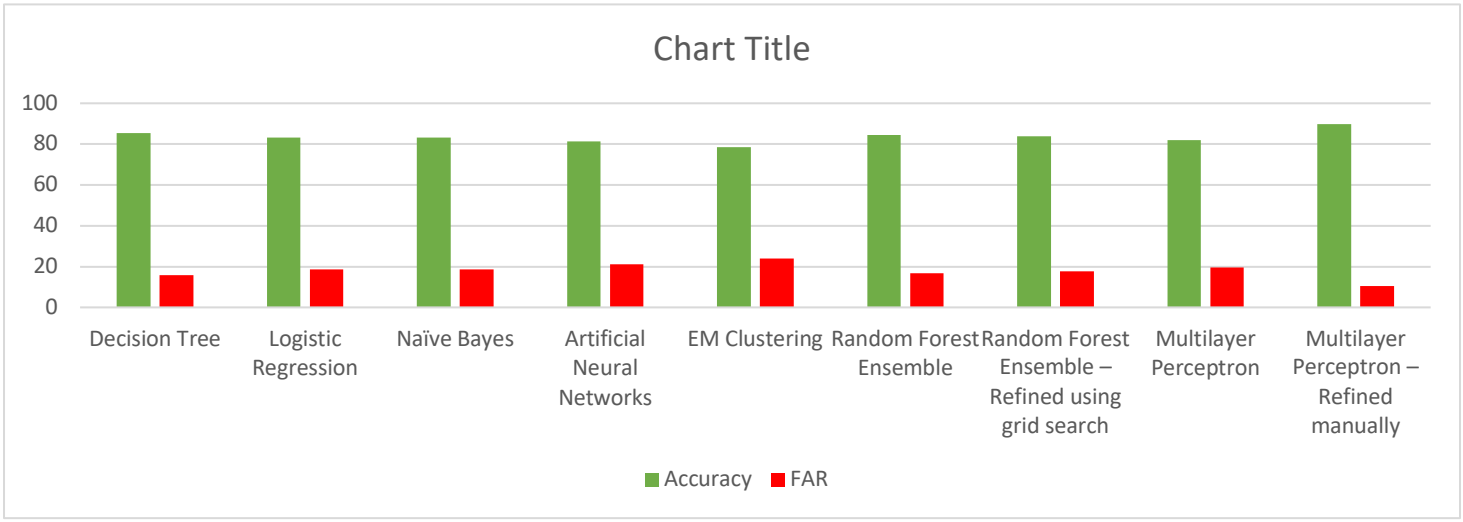

Figure 4: Visualization of results along with benchmark metrics

Confusion Matrix:

Since we have the attack category available as part of the dataset, I used it to generate a confusion matrix per attack category. This visualization shows that most of the false negatives are from fuzzer attack. This could be an indication to search for more appropriate features to identify fuzzer attacks.

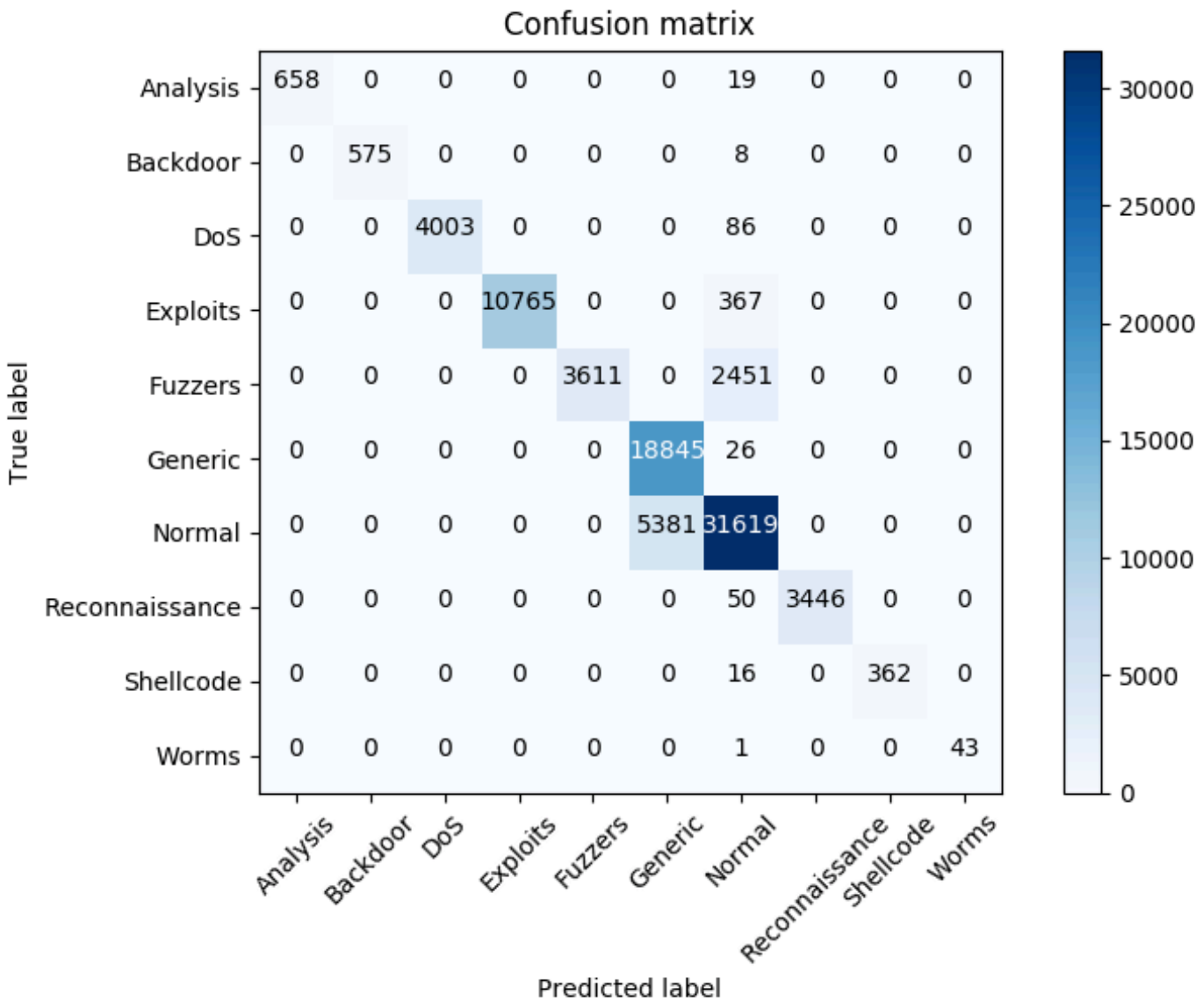

Figure 5: Confusion matrix of attack category predictions

On README.md, follow the instructions under "Setting up the project" to setup the project. Then execute the below command to generate the above confusion matrix. The command also saves the confusion matrix in the results folder.

```
python -c "from src.exec import *; confusion_matrix()"
```

Reflection

In this paper, we discussed about how, with the advent of advanced machine learning algorithms, Heuristic Machine Learning models can be used for anomaly based intrusion detection. Most of the research and studies in this area are focused on classifying network traffic into normal and attack categories using supervised learning. Supervised Learning algorithms trains

models based on labeled datasets. However, when considering real world scenarios, there are 2 main short comings in this approach.

First is the unavailability of labeled dataset. When it comes to training a model on network traffic, there is no one-size-fit-all. Each network is unique with its own topology and traffic patterns. This mandates that models be trained with data from the same network on which the model is intended to be deployed. That said, it is fairly easier to collect normal traffic data. But on the other hand it is close to impossible to collect sample traffic patterns representing all the different kinds of attacks and threats. Hence, supervised learning is not a suitable approach for Anomaly Based NIDSs.

Secondly, with ANIDSs, the main goal is to raise alarm as soon as anomalous traffic patterns are detected and NOT merely classifying each connection as benign or malicious. Hence, the problem has be looked from a time series perspective, which can trigger alarms in real-time and enable actions in-time to mitigate threats.

Improvement

My next steps in this area will be around creating a comprehensive solution for network datasets which does not have proper labeling. I intend to apply LSTM RNN and Unsupervised learning algorithms and perform time-series analysis and clustering.

Citations

The details of the UNSW-NB15 data set are published in following the papers:

Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). "Military Communications and Information Systems Conference (MilCIS), 2015. IEEE, 2015.

Moustafa, Nour, and Jill Slay. "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set." Information Security Journal: A Global Perspective (2016): 1-14.

Free use of the UNSW-NB15 dataset for academic research purposes is hereby granted in perpetuity. Use for commercial purposes is strictly prohibited. Nour Moustafa and Jill Slay have asserted their rights under the Copyright.

To whom intend the use of the UNSW-NB15 data set have to cite the above two papers.

For more information, please contact the authors:
1. Nour Moustafa: e-mail nour.abdelhameed@student.adfa.edu.au
2. Jill Slay: e-mail j.slay@adfa.edu.au

References

1. https://en.wikipedia.org/wiki/Standard_deviation
2. https://www.toptal.com/machine-learning/ensemble-methods-machine-learning
3. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
4. https://datascience.stackexchange.com/questions/13746/how-to-define-a-custom-performance-metric-in-keras
5. http://scikit-learn.org/stable/modules/cross_validation.html
6. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
7. https://cs.stanford.edu/~ppasupat/a9online/1107.html
8. http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
9. http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/
10. http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414