Machine Learning Security Projects

This document lists projects for students as part of their advanced network security course. Project 1 is required by every student and projects 2 and 3 are optional (they are bonus projects).

# Project 1: Anomaly Detection Accuracy with Unknown Attacks

1. **Goal**: Understanding the prediction of FeedForward Neural Network model in detecting network attacks
2. **Setup**: Follow the installation instructions to install necessary software packages such as:
   1. Python 3.5 or later version
   2. Install keras and tensorflow (Install anaconda will install all necessary software packages)
3. **Pre-Requisites**
   1. Knowledge of supervised machine learning, python programming, a framework such as keras or tensorflow to build machine learning programs
   2. Recommended Hardware - Multi-core CPU preferrable, i7 or above is preferrable (i5 will work but will be slow), at least 8GB RAM
   3. Software - Anaconda, Spyder, Python, pip, keras. Please refer to installation instructions at the end of this document
4. **Resources**
   1. Dataset - [NSL KDD Dataset](https://www.unb.ca/cic/datasets/nsl.html)
   2. Time - 20 days
5. **Background**: In supervised training, both normal and attack data is passed. However, attack data is not definite, the data you have is usually a very small subset of the possible attack data. If we train a NN on available normal and only a subset of attack data, how does the trained model react to when the other subset of attacks are passed through it while in detection mode.
6. **Requirements: address the following questions.**
   1. When data representing new type of attack is passed, does your model predict it to be normal or attack ?

Hint: *If the dataset has 4 attack types -> A1, A2, A3, and A4 along with normal data N, start with* ***subsetA*** *comprising of A1 and A2 for training the model along with all the normal data N. A3, A4 then will be the new or unknown attack type to the trained model.*

* 1. What is the average accuracy that it detects the new type of attack to be as normal or attack ?

Hint: *With the model set to perform binary classification, the predicted values of the model can be 0 -> Normal or 1 -> Attack. This prediction is associated with the accuracy, accuracy of the prediction. Note down the accuracy of the prediction, the prediction is normal or attack, for the above unknown attack, A3.*

* 1. How are the untrained subset of attacks different from the trained ones?

Hint: *How are A3 and A4 different from A1 and A2. Do A1 and A3 belong to same attack category or have similar differences wrt normal data ? Note down what your observations can be made by looking at the data for these attacks.*

* 1. Observe your model’s prediction capability wrt change in the attack types it was trained on

Hint: *Create 2 more subsets -> subsetB: {A1, A3}, and subsetC: {A2, A4} and observe the accuracy of the prediction when trained on these subset of attack types. Observe if the model predicts better when it was trained on a specific subset of attack types.*

* 1. Does the prediction accuracy relate to the attacks being similar? If so, what is the similarity ?

Hint: *If the prediction accuracy was found better in (d) for a particular subset, look at the data and the attack types for any similarities that would have made the prediction better.*

*7. Deliverables*

1. *Submit the project report by using the given project report template*
2. *Running environment (ThoTh Lab or your own computer).*

# Projects 2 and 3 are optional (bonus projects)

# Project 2: FNN vs RNN

1. **Goal**: Understanding the performance of ANNs and RNNs in detecting network security attacks
2. **Setup**: Follow the installation instructions to install necessary software packages such as:
   1. Python 3.5 or later version
   2. Install keras and tensorflow (Install anaconda will install all necessary software packages)
3. **Pre-Requisites**
   1. Knowledge of supervised machine learning, python programming, a framework such as keras or tensorflow to build machine learning programs
   2. Recommended Hardware - Multi-core CPU preferrable, i7 or above is preferrable (i5 will work but will be slow), at least 8GB RAM
   3. Software - Anaconda, Spyder, Python, pip, keras. Please refer to installation instructions at the end of this document
4. **Resources**
   1. Dataset - [NSL KDD Dataset](https://www.unb.ca/cic/datasets/nsl.html), [CIC IDS 2017](https://www.unb.ca/cic/datasets/ids-2017.html)
   2. Time - 20 days
5. **Background**: ANNs are simplest neural networks. Recurrent Neural Networks (RNNs) are ANNs with the ability to influence the current output in the context of previous input.
6. **Deliverables**
   1. Are RNNs better than ANNs in detecting the time-series data such as network flows? Please justify your answer with results.

Hint: *Create a simple ANN model and a simple RNN model and compare the prediction accuracy of the models under the same test data.*

* 1. What types of RNNs have capabilities to identify long range dependencies and how they achieve it?

Hint: *RNNs have short term memory. In other words, they are capable of remembering the immediate previous input. However, they are specific type of RNNs that are capable of relating the current input to n previous inputs. Report what those RNNs are and how they achieve the above mentioned capability.*

* 1. For one type of RNN identified in (b), what observations have been made with increased long range dependency mapping in regards to the accuracy of the chosen RNN.

Hint: *Chose one of the RNNs identified in (b) and change n from 1 to 5 and observe the accuracy of the prediction. We need to understand how increasing n from 1 to 5 impacts the model’s training time, prediction time and prediction accuracy. Plot the results you observed for different n values.*

* 1. Were RNNs found to be better (or NOT better) than ANNs, does it have anything to do with the dataset used?

Hint: Using the second dataset, answer (a) again, and justify your findings with observations you made with the second dataset. You can split the Port Scans (Friday afternoon) data in the second dataset into training and prediction data and observe the results.

# Project 3: Unsupervised Training for Detecting Network Attacks

1. **Goal**: Understanding SOMs, unsupervised neural networks under port scans
2. **Pre-Requisites**
   1. Knowledge of supervised machine learning, python programming, a framework such as keras or tensorflow to build machine learning programs
   2. Hardware - Multi-core CPU preferrable, i7 or above is preferable (i5 will work but will be slow), at least 8GB RAM
   3. Software - Anaconda, Spyder, Python, pip, keras. Please refer to installation instructions at the end of this document
3. **Resources**
   1. Dataset - [CIC IDS 2017](https://www.unb.ca/cic/datasets/ids-2017.html) (Friday afternoon has port scans)
   2. Time - 20 days
4. **Background**: Self Organizing Map (SOM) is an ANN trained in unsupervised mode. They are capable of grouping similar nodes while reducing the dimension. However, there are pros and cons of SOMs.
5. **Deliverables** 
   1. What are the pros and cons of using an unsupervised training mode for network attack detection ?

Hint: *All types of training modes have pros and cons wrt the domain they are used in. Explain how unsupervised training mode will help in network attack detection. And how to achieve it.*

* 1. Observe the performance and accuracy of SOMs while attempting to identify port scans from normal user activity.

Hint: *The dataset has data split into 5 days. While the first day data is all normal data, subsequent days have seen attacks. Using only Afternoon data from Day 5, split the data into training and detection data and observe the performance. Use 50% for training and 50% for testing the model after training is completed.*

* 1. Are SOMs a good choice for network attack detection ? Explain and Why and Why Not ?

Hint: *Based on the results you get, determine if SOM can be used for network attack detection.*

* 1. What are the true positive and false positive rates observed using SOM for network attack detection

Hint: *When attempting to detect attacks in network flows, a positive result means, the model has detected an attack, and negative indicates that the model has detected the incoming flow to be normal. Not all positives are True. Similarly, not all negatives are True. True Positives are those flows that have been detected as attacks and are indeed part of an attack. False Positives are those that have been detected as attacks by the model, but they are not part of any attacks. Likewise, True Negatives are the normal flows that have been detected by the model as normal, and false negatives are the attack flows that have been detected as normal. These four metrics, True Positive, False Positive, True Negative and False Negative together help measure the abilities of an intrusion detection system. Observe these metrics for SOM.*

# Installation Instructions:

1. Install anaconda from anconda.org 3.6 (or above) python version
2. Make sure all of anaconda is up to date with the latest releases
3. You may start up Anaconda Prompt and run the commands below
   1. conda update conda
   2. conda update --all
   3. conda install mingw libpython
   4. pip install tensorflow
   5. pip install keras
   6. conda install python-graphviz

If you just do pip install graphviz on the anaconda prompt then your py files will still not be able to find graphviz.

1. Launch Anaconda Navigator and then select Spyre or Launch Spyder directly
2. With Spyder you should now be able to write your python programs.

# Code Snippets

Put together all the snippets from part 1 to 3 to get a working code. Part 4 gives you code snippets for visualizing your results or findings. Depending on the project assigned, replace snippets in Part 3 with associated snippets in Part 5 and 6. Part 7 provides additional snippets that you can play with.

**Part 1 - Data Preprocessing**

|  |
| --- |
| # To load a dataset file in Python, you can use Pandas. Import pandas using the line below  import pandas as pd  # Import numpy to perform operations on the dataset  import numpy as np  # Import dataset.  # Replace ‘dataset\_file\_path’ # with the file path such as “C:\Users\...\dataset.csv’.  # The file can be a .txt as well.  # If the dataset file has header, then keep header=0 otherwise use header=none  dataset = pd.read\_csv(dataset\_file\_path, header=0)  X = dataset.iloc[:, 0:-1].values  y = dataset.iloc[:, -1].values |

|  |
| --- |
| # Encoding categorical data  from sklearn.preprocessing import LabelEncoder, OneHotEncoder  labelencoder\_X\_1 = LabelEncoder()  X[:, 1] = labelencoder\_X\_1.fit\_transform(X[:, 1])  onehotencoder = OneHotEncoder(categorical\_features = [1])  X = onehotencoder.fit\_transform(X).toarray()  X = X[:, 1:] |

|  |
| --- |
| # Splitting the dataset into the Training set and Test set  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0) |

|  |
| --- |
| # Perform feature scaling - normalization. For ANN you can use StandardScaler, for RNNs  # recommended is MinMaxScaler.  from sklearn.preprocessing import StandardScaler  sc = StandardScaler()  X\_train = sc.fit\_transform(X\_train)  X\_test = sc.transform(X\_test) |

|  |
| --- |
| # Perform feature scaling. For ANN you can use StandardScaler, for RNNs recommended is  # MinMaxScaler.  from sklearn.preprocessing import MinMaxScaler  sc = MinMaxScaler(feature\_range = (0, 1))  X\_train = sc.fit\_transform(X\_train) |

**Part 2 - Building FNN**

|  |
| --- |
| # Importing the Keras libraries and packages  import keras  from keras.models import Sequential  from keras.layers import Dense |

|  |
| --- |
| # Initialising the ANN  classifier = Sequential()  # Adding the input layer and the first hidden layer  classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu', input\_dim = 11))  # Adding the second hidden layer  classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu'))  # Adding the output layer  classifier.add(Dense(units = 1, kernel\_initializer = 'uniform', activation = 'sigmoid'))  # Compiling the ANN  classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])  # Fitting the ANN to the Training set  classifierHistory = classifier.fit(X\_train, y\_train, batch\_size = 10, epochs = 100) |

**Part 3 - Making predictions and evaluating the model**

|  |
| --- |
| # Predicting the Test set results  y\_pred = classifier.predict(X\_test)  y\_pred = (y\_pred > 0.9) |

|  |
| --- |
| # Making the Confusion Matrix  from sklearn.metrics import confusion\_matrix  cm = confusion\_matrix(y\_test, y\_pred) |

**Part 4 - Visualizing**

|  |
| --- |
| # Import matplot lib libraries for plotting the figures.  import matplotlib.pyplot as plt  # You start with creating a new figure  plt.figure()  # You can plot lines as below. Try plotting the actual and predicted values, y\_test, and y\_pred  plt.plot(x, y, color='red', label=test)  # You can add a title to the figure using the below statement  plt.title('Test Plot)  # Specify where to put the legend  plt.legend(loc='lower right')  # Save the figure  plt.savefig("TestFigure.png")  # View the figure  plt.show() |

**Part 5 - Building RNN**

|  |
| --- |
| # Converting to a dastructure with timesteps for feeding into RNN (sequences)  \_nTimesteps = 2  X\_train\_sequence = []  y\_train\_sequence = []  for i in range(\_nTimesteps, np.shape(X\_train)[0]):  X\_train\_sequence.append(X\_train[i-\_nTimesteps:i, :])  y\_train\_sequence.append(y\_train[i-\_nTimesteps:i, 0])  X\_train\_sequence = np.array(X\_train\_sequence)  y\_train\_sequence = np.array(y\_train\_sequence)  # Initializing the RNN  classifier = Sequential()  # Adding the first LSTM layer and some Dropout regularisation  classifier.add(LSTM(units = 20, return\_sequences = True, input\_shape = (X\_train.shape[1], 1)))  classifier.add(Dropout(0.2)) |

**Part 6 - SOM**

|  |
| --- |
| # Training the SOM. You would need download minisom.py file  from minisom import MiniSom  som = MiniSom(x = 10, y = 10, input\_len = len(X\_som\_scaled[0]), sigma = 1.0, learning\_rate = 0.1, decay\_function=None, random\_seed=None)  # X\_som\_scaled is the result obtained after feature scaling X using MinMaxScaler  som.random\_weights\_init(X\_som\_scaled)  som.train\_batch(data = X\_som\_scaled, num\_iteration = 100)  # Visualizing the results  from pylab import bone, pcolor, colorbar, plot, show  bone()  pcolor(som.distance\_map().T)  colorbar()  markers = ['s', 'o']  colors = ['g', 'r']  for i, x in enumerate(X\_som\_scaled):  w = som.winner(x)  plot(w[0] + 0.5,  w[1] + 0.5,  markers[int(y\_som[i])],  markeredgecolor = colors[int(y\_som[i])],  markerfacecolor = 'None',  markersize = 10,  markeredgewidth = 2)  show()  # Getting all outliers found by SOM  distance\_map = som.distance\_map()  outliers = []  for i in range(0, 10):  for j in range(0, 10):  if distance\_map[i, j] == 1:  outliers.append([i, j])  print(outliers)  # Get the mappings  mappings = som.win\_map(X\_som\_scaled) |

**Part 7 - Miscellaneous**

|  |
| --- |
| # You can plot the training and validation loss incurred during each epoch  loss = classifierHistory.history['loss']  val\_loss = classifierHistory.history['val\_loss']  epochs = range(100)  plt.figure()  plt.plot(epochs, loss, 'bo', label='Training loss')  plt.plot(epochs, val\_loss, 'b', label='Validation loss')  plt.title('Training and Validation loss')  plt.xlabel('epochs')  plt.ylabel('loss')  plt.legend()  plt.savefig('Loss\_testModel.png')  plt.show()  # Save the model  # serialize model to JSON  model\_json = classifier.to\_json()  with open( testModel".json", "w") as json\_file:  json\_file.write(model\_json)  # serialize weights to HDF5  classifier.save\_weights("testModel.h5")  print("Saved model to disk")  # load json and create model  json\_file = open("testModel.json", 'r')  loaded\_model\_json = json\_file.read()  json\_file.close()  loaded\_model = model\_from\_json(loaded\_model\_json)  # load weights into new model  loaded\_model.load\_weights("testModel.h5")  print("Loaded model from disk") |