Machine Learning Security Project

(CSE 548 Advanced Computer Network Security)

Preparation:

1. Study the provided ML labs:

* CS-ML-00001 – Setup Machine Learning Running Environment on Linux
* CS-ML-00101 – Understanding NSL-KDD Dataset
* CS-ML-00199 – Python Machine Learning Tutorial-Basic Concepts
* CS-ML-00200 – Python Machine Learning Data Processing Modules
* CS-ML-00201 – Feed-Forward Neural Network (FNN)
* CS-ML-00301 – Use Feed-Forward Neural Network (FNN) for Network Traffic Anomaly Detection

1. Download the source codes provided in dropbox link: <https://www.dropbox.com/sh/oao60gnk2a0xoyw/AACBrVE0S4p2L_51X18tlJEVa?dl=0>

Three tasks are presented here. Each team can choose two tasks (any two) to finish. If a team finishs all three tasks, 20% bonus of this project grade will be provided based on its completeness.

# Task 1: Anomaly Detection Accuracy with Unknown Attacks

1. **Goal**: Understanding the prediction of the FeedForward Neural Network model in detecting network attacks
2. **Background**: In supervised training, both normal and attack data (NSL-KDD) are 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 an NN on available normal and only a subset of attack data, how does the trained model react when the other subsets of attacks are passed through in the detection mode.
3. **Setup**: Follow the installation instructions (Appendix A) to install necessary software packages such as:
   * Python 3.5 or later version
   * Install *keras* and *tensorflow* (Install anaconda will install all necessary software packages)
4. **Pre-Requisites**
   * Knowledge of supervised machine learning, python programming, a framework such as *keras* or *tensorflow* to build machine learning programs (study the provided ML labs)
   * Recommended Hardware - Multi-core CPU preferable, i7 or above is preferable (i5 will work but will be slow), at least 8GB RAM
   * Software - Anaconda, Spyder, Python, pip, *keras*. Please refer to installation instructions provided by CS-ML-ML-00001 or the instructions provided in Appendix A at the end of this document
5. **Resources**
   * Datasets - NSL KDD Dataset (<https://www.unb.ca/cic/datasets/nsl.html>). It is also provided in the downloading files of the lab CS-ML-00200 or CS-ML-00301)
   * Running environment (Google Colab, Amazon AWS, or your own computer).
6. **Deliverables**

* Submission requirements:
  + A project report by using the given project report template
  + A short video demo of your work (up to 10 minutes) and answer the following questions

In your report, please provide the following reporting items:

* + When data representing a 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.

Choose A1-A4 based on your understanding/ randomly (please check the CS-ML-00101). And save the model always so that you can predict its performance against different test sets. Refer to Part 7 of Appendix B - Miscellaneous section for instructions on saving and loading a saved model.

* + What is the average accuracy of detecting the new type of attack as normal or attack?

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

* + How is 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 the same attack category or have similar differences w.r.t normal data? Note down what your observations can be made by looking at the data for these attacks.

* + Observe your model’s prediction capability w.r.t 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.

* + 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.

# Task 2: FNN vs. RNN

1. **Goal**: Understanding the performance of ANNs and RNNs in detecting network security attacks
2. **Background**: ANNs are the simplest neural networks. Recurrent Neural Networks (RNNs) are ANNs that can influence the current output in the previous input context.
3. **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)
4. **Pre-Requisites**
   1. Knowledge of supervised machine learning, python programming, a framework such as *keras* or *tensorflow* to build machine learning programs (study the provided ML labs)
   2. Recommended Hardware - Multi-core CPU preferable, 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 provided by CS-ML-ML-00001 and the instructions of Appendix A at the end of this document
5. **Resources**
   1. Datasets - NSL KDD Dataset (<https://www.unb.ca/cic/datasets/nsl.html>). It is also provided in the downloading files of the lab CS-ML-00200 or CS-ML-00301)
   2. Datasets - CIC IDS 2017 Dataset (<https://www.unb.ca/cic/datasets/ids-2017.html>).
6. **Deliverables**

* Submission requirements:
  + A project report by using the given project report template
  + A short video demo of your work (up to 10 minutes) and answer the following questions

In your report, please provide the following reporting items:

* 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 models' prediction accuracy under the same test data.

* 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 can remember the immediate previous input. However, they are a specific type of RNNs that can relate the current input to n previous inputs. Report what those RNNs are and how they achieve the capability mentioned above.

* For one type of RNN identified in (b), what observations have been made with increased long-range dependency mapping concerning 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 prediction accuracy. 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.

* 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.

# Task 3: Unsupervised Training for Detecting Network Attacks

1. **Goal**: Understanding SOMs, unsupervised neural networks under port scans
2. **Background**: Self Organizing Map (SOM) is an ANN trained in unsupervised mode. They can group similar nodes while reducing the dimension. However, there are pros and cons of SOMs.
3. **Pre-Requisites**
   1. Knowledge of supervised machine learning, python programming, a framework such as *keras* or *tensorflow* to build machine learning programs (study the provided ML labs)
   2. Recommended Hardware - Multi-core CPU preferable, 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 provided by CS-ML-ML-00001 and the instructions of Appendix A at the end of this document
4. **Resources**
   1. Datasets - CIC IDS 2017 Dataset (<https://www.unb.ca/cic/datasets/ids-2017.html>).
5. **Deliverables**

* Submission requirements:
  + A project report by using the given project report template
  + A short video demo of your work (up to 10 minutes) and answer the following questions

In your report, please provide the following reporting items:

* + 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 w.r.t the domain they are used. Explain how unsupervised training mode will help in network attack detection. And how to achieve it.
  + 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.
  + 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.
  + What are the true positive and false positive rates observed using SOM for network attack detection?   
    Hint: When detecting attacks in network flows, a positive result means the model has detected an attack, and a 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 have been detected as attacks by the model, but they are not part of any attacks. Likewise, True Negatives are the normal flows that the model has detected 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 help measure an intrusion detection system's abilities. Observe these metrics for SOM.

# Appendix

# 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 parts 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 test how many flows to check in previous steps  \_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**

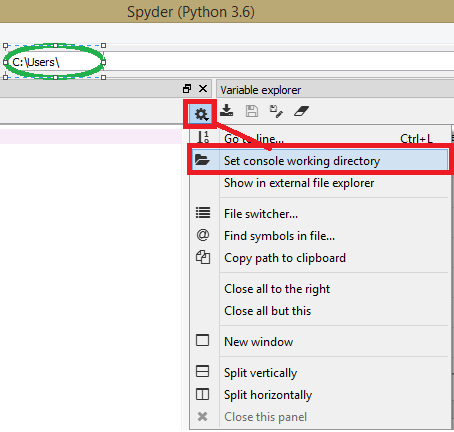
|  |
| --- |
| # Training the SOM. You would need download minisom.py file  from minisom import MiniSom  som = MiniSom(x = 10, y = 10, input\_len = len(X\_som\_scaled[0]), sigma = 1.0, learning\_rate = 0.1, decay\_function=None, random\_seed=None)  # X\_som\_scaled is the result obtained after feature 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") |

# Tips

1. Open Spyder. Within Spyder, open your .py file
2. Once the file has been loaded, please make sure you set the console working directory to the path this file is in, as below. This way, you do not have to give the entire path, just the dataset file's relative path.



1. Now, run your py code 1 part at a time after ensuring you have the following changes
2. In **Part 1**, start with using the **KDDTrain+.txt** file. Make sure this file is beside your .py file so that you can use **pd.read\_csv(‘KDDTrain+.txt’, header=None)**.
3. The dataset file does not have a header, so header=None is passed to pandas in the above line.
4. Now, since the last 2 columns are label-related columns in this dataset file, they have to be excluded in the X variable. **X = dataset.iloc[:, 0:-2].values**.
5. The label\_column should be made to hold the label column, which is the 2nd column from right (-2). **label\_column = dataset.iloc[:, -2].values**.
6. However, the label\_column has string values, so convert them to binary using the list and then convert that list to numpy array, 'y' (0 => Normal, 1 => Attack).
7. Then, use the label encoder to encode the string values to int in X. And then the categorical encoder to encode the categorical columns. These 2 operations need to be done on columns 1, 2, and 3 in X. After doing categorical encoding. You will see the total number of columns in X has increased.
8. Now, **In Part 2**, in line classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu', input\_dim = 11)), pass **input\_dim=len(X\_train[0])**
9. Then, as a start, set the code to run for only 10 epochs, and you would see the epoch results printing while the model is being trained as below.

Epoch 1/10

94479/94479 [==============================] - 17s 181us/step - loss: 0.0815 - acc: 0.9727

1. See if increasing the number of epochs is giving you better accuracy.
2. In **Part 3**, use the test subset you obtained after splitting the dataset file. You can also use the KDDTest+.txt file (if you chose to use this file, you need to do the same data pre-processing on this dataset file too - repeat Part 1 on this dataset file). You will see the confusion matrix results printed below. Change the threshold from 0.9 to 0.5 in line **y\_pred = (y\_pred > 0.9),** and you will see the confusion matrix results change too.   
   To view the confusion matrix on spyder, add print(cm) after obtaining cm. Note that depending on the order of the parameters you pass, the confusion matrix will change. (Example, cm = confusion\_matrix(y\_pred, y\_test) will give different results and changes the TP, TN, FP, FN position in the matrix)



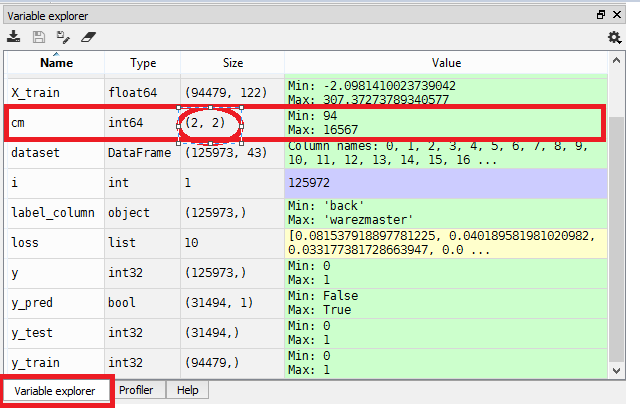
For confusion matrix obtained using cm = confusion\_matrix(y\_test, y\_pred), with 0 being negative and 1 being positive, the true and false positives and negatives are mapped as below. Note that, depending on the positive class that we chose, the matrix changes.

Predicted 0 Predicted 1

Actual 0 TN FN

Actual 1 FP TP

You can also view **cm** variable, variable explorer on Spyder, as below.

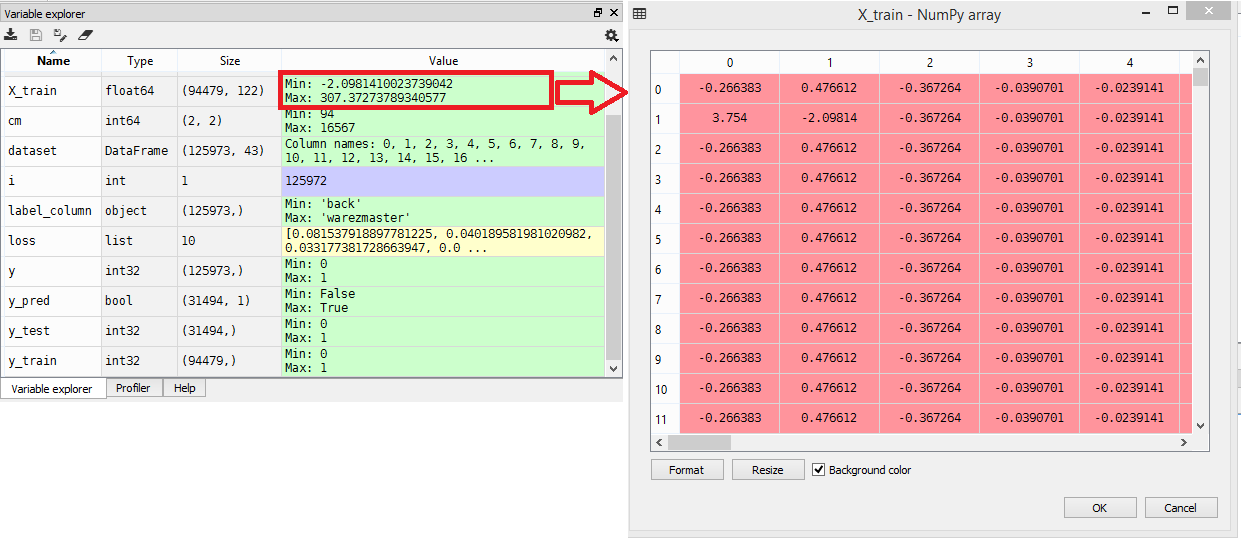


13. For Part 4, try plotting the accuracy and loss using the below lines in 2 different plots -

**plt.plot(classifierHistory.history[‘acc’])**

**plt.plot(classifierHistory.history[‘loss’])**

14. You can also view the X\_train values by clicking on the X\_train variable in the variable explorer. This variable explorer is above your ipython console on Spyder.



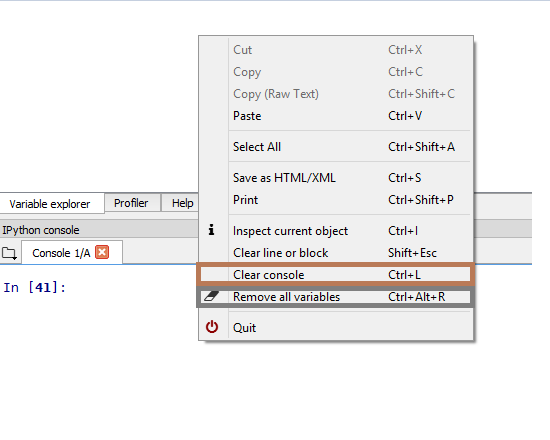
You can see that when you do one-hot encoding, each categorical column that you performed one-hot encoding on is expanded to multiple columns to represent the column value in binary format. One hot encoding is not necessary, but it will make training faster. Alternatively, you can use a script to create a new dataset file replacing the strings with integer values. And this new file can be used directly.

15. You can also remove features (columns) from the input file that you think are not necessary.For instance, you can choose specific features out of all 41 features in the NSL-KDD file as below -

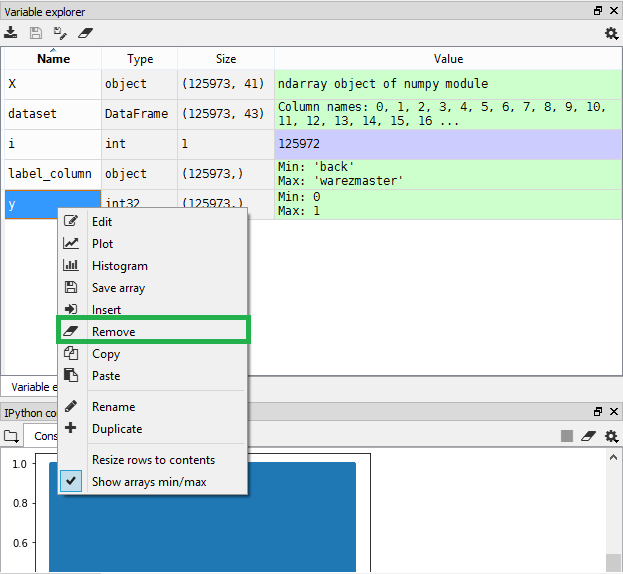
extracted\_columns = dataset.iloc[:, [2,3,4,5,6,7,11,13,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,41]].values

X\_train = np.array(extracted\_columns)

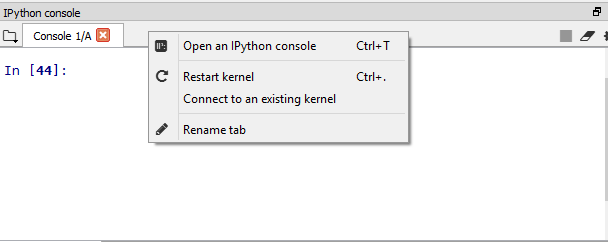
16. If anytime you need to clear all the variables, and restart the execution, you can clear the console and/or variables on Spyder by right-clicking on the iPython console and selecting the appropriate action as in the below snapshot.



17. You can also clear individual variables from the variable explorer by right-clicking on the variable and selecting remove, and then re-running the corrected code. You can also try other options like plot etc, as necessary.



18. You can also right-click on the empty area beside the iPython console tab on Spyder and restart the kernel of that console (say, to retrieve modules that were installed while spyder is running). You can also open another iPython console.



19. You can get help documentation by placing the cursor before the keyword you need help documentation on and then pressing CTRL+I. Please see the below snapshot:

