**Task II – Structured Data Analysis**

NOTE: code snippets are taken from My work on Jupyter Notebook.

Due to lack of memory , I tried to use smaller dataset. Which is in this case 10%ge of actual dataset.

1. Do an exploratory Data Analysis (EDA) using Python

**Load Dependency Library and Dataset.**

import numpy as np

import pandas as pd

import seaborn as sns

import tensorflow as tf

import matplotlib.pyplot as plt

data = pd.read\_csv(train\_path,names=columns)

data.head()

# transform target column values to attack-types as suggested in dataset description.

attacks\_types = {

'normal': 'normal',

'back': 'dos',

'buffer\_overflow': 'u2r',

'ftp\_write': 'r2l',

'guess\_passwd': 'r2l',

'imap': 'r2l',

'ipsweep': 'probe',

'land': 'dos',

'loadmodule': 'u2r',

'multihop': 'r2l',

'neptune': 'dos',

'nmap': 'probe',

'perl': 'u2r',

'phf': 'r2l',

'pod': 'dos',

'portsweep': 'probe',

'rootkit': 'u2r',

'satan': 'probe',

'smurf': 'dos',

'spy': 'r2l',

'teardrop': 'dos',

'warezclient': 'r2l',

'warezmaster': 'r2l',

}

#Adding Attack Type column

data['Attack Type'] = data.target.apply(lambda r:attacks\_types[r[:-1]])

data.head()

# I found 5 different classes.

‘’’

dos 391458

normal 97278

probe 4107

r2l 1126

u2r 52

‘’’

### Data Preprocessing

print(data.isnull().values.any())

print(data.isnull().sum().sum())

# as no null values found, so null handling opts.

**Uni-Variate Distribution Analysis**

In [19]:

data['srv\_count'].plot.hist()

data['duration'].plot.hist()

data['protocol\_type'].value\_counts().plot.bar()

data['service'].value\_counts().plot.bar()

**Bi-Variate Distribution Analysis**

In [19]:

sns.pairplot(data)

example snippet:

#### **detect outliers is to draw some boxplots:**

data.iloc[:,[0,1,2,3,4,5,6,7,8,9,10]].plot(kind='box', subplots=True, layout=(10,3), sharex=False, sharey=False, figsize=(20, 20), color='deeppink')

data.iloc[:,[10,11,12,13,14,15,16,17,18,19,20]].plot(kind='box', subplots=True, layout=(10,3), sharex=False, sharey=False, figsize=(20, 20), color='deeppink')

data.iloc[:,[20,21,22,23,24,25,26,27,28,29,30]].plot(kind='box', subplots=True, layout=(10,3), sharex=False, sharey=False, figsize=(20, 20), color='deeppink')

data.iloc[:,[30,31,32,33,34,35,36,37,38,39,40]].plot(kind='box', subplots=True, layout=(10,3), sharex=False, sharey=False, figsize=(20, 20), color='deeppink')

**transform highly skewed features suitably**

for handling skewed features :

categorical features : didn’t make any changes as going to try DNN. So, let NN model handle it.

numerical features : made using z-score. Will try in future for for log scale.

**Numerical Feature Scaling using Z-score.**

# Numeric Feature scale with Z-score

numeric\_variables\_zscore\_attributes = dict()

# Numeric Feature scale with Z-score

def encode\_numeric\_zscore(data, name, mean=None, sd=None):

if mean is None:

mean = data[name].mean()

if sd is None:

sd = data[name].std()

#for training

numeric\_variables\_zscore\_attributes[name] = {'mean':mean,'sd':sd}

data[name] = (data[name] - mean) / sd

encode\_numeric\_zscore(data, 'duration')

encode\_numeric\_zscore(data, 'src\_bytes')

encode\_numeric\_zscore(data, 'dst\_bytes')

encode\_numeric\_zscore(data, 'wrong\_fragment')

encode\_numeric\_zscore(data, 'urgent')

encode\_numeric\_zscore(data, 'hot')

encode\_numeric\_zscore(data, 'num\_failed\_logins')

encode\_numeric\_zscore(data, 'num\_compromised')

encode\_numeric\_zscore(data, 'root\_shell')

encode\_numeric\_zscore(data, 'su\_attempted')

encode\_numeric\_zscore(data, 'num\_root')

encode\_numeric\_zscore(data, 'num\_file\_creations')

encode\_numeric\_zscore(data, 'num\_shells')

encode\_numeric\_zscore(data, 'num\_access\_files')

encode\_numeric\_zscore(data, 'num\_outbound\_cmds')

encode\_numeric\_zscore(data, 'count')

encode\_numeric\_zscore(data, 'srv\_count')

encode\_numeric\_zscore(data, 'serror\_rate')

encode\_numeric\_zscore(data, 'srv\_serror\_rate')

encode\_numeric\_zscore(data, 'rerror\_rate')

encode\_numeric\_zscore(data, 'srv\_rerror\_rate')

encode\_numeric\_zscore(data, 'same\_srv\_rate')

encode\_numeric\_zscore(data, 'diff\_srv\_rate')

encode\_numeric\_zscore(data, 'srv\_diff\_host\_rate')

encode\_numeric\_zscore(data, 'dst\_host\_count')

encode\_numeric\_zscore(data, 'dst\_host\_srv\_count')

encode\_numeric\_zscore(data, 'dst\_host\_same\_srv\_rate')

encode\_numeric\_zscore(data, 'dst\_host\_diff\_srv\_rate')

encode\_numeric\_zscore(data, 'dst\_host\_same\_src\_port\_rate')

encode\_numeric\_zscore(data, 'dst\_host\_srv\_diff\_host\_rate')

encode\_numeric\_zscore(data, 'dst\_host\_serror\_rate')

encode\_numeric\_zscore(data, 'dst\_host\_srv\_serror\_rate')

encode\_numeric\_zscore(data, 'dst\_host\_rerror\_rate')

encode\_numeric\_zscore(data, 'dst\_host\_srv\_rerror\_rate')

import json

with open('numeric\_variables\_zscore\_attributes.json','w') as f:

json.dump(numeric\_variables\_zscore\_attributes,f)

NOTE : also Normalize samples individually to unit norm before pushing to Model. Each sample (i.e. each row of the data matrix) with at least one non zero component is rescaled independently of other samples so that its norm (l1, l2 or inf) equals one. why ? to make each sample feature close to one in feature space.

### display the density plots of these variables and analyze their skewness:

data.iloc[:,[0,1,2,3,4,5,6,7,8,9,10]].plot(kind='density', subplots=True, layout=(10,3), sharex=False, figsize=(20, 20)) #after remove of duplicate features, which might cause 100%ge correlation.

data.iloc[:,[10,11,12,13,14,15,16,17,18,19,20]].plot(kind='density', subplots=True, layout=(10,2), sharex=False, figsize=(20, 20)) #after remove of duplicate features

# which might cause 100%ge correlation.

data.iloc[:,[20,21,22,23,24,25,26,27,28,29,30]].plot(kind='density', subplots=True, layout=(10,2), sharex=False, figsize=(20, 20)) #after remove of duplicate features

# which might cause 100%ge correlation.

data.iloc[:,[30,31,32,33,34,35,36,37,38,39,40]].plot(kind='density', subplots=True, layout=(10,2), sharex=False, figsize=(20, 20)) #after remove of duplicate features

# which might cause 100%ge correlation.

#### **Data correlation using heatmap**

#data = data.dropna('columns')# drop columns with NaN

data = data[[col for col in data if data[col].nunique() > 1]]# keep columns where there are more than 1 unique values

corr = data.corr()

plt.figure(figsize=(15,12))

sns.heatmap(corr)

plt.show()

# more details . by looking at each individuals

data['num\_root'].corr(data['num\_compromised'])

data['srv\_serror\_rate'].corr(data['serror\_rate'])

data['srv\_count'].corr(data['count'])

data['srv\_rerror\_rate'].corr(data['rerror\_rate'])

#### **Drop Independent features which have high correlation score more than 90%ge**

#This variable is highly correlated with num\_compromised and should be ignored for analysis.

#(Correlation = 0.9938277978738366)

data.drop('num\_root',axis = 1,inplace = True)

#This variable is highly correlated with serror\_rate and should be ignored for analysis.

#(Correlation = 0.9983615072725952)

data.drop('srv\_serror\_rate',axis = 1,inplace = True)

#This variable is highly correlated with rerror\_rate and should be ignored for analysis.

#(Correlation = 0.9947309539817937)

data.drop('srv\_rerror\_rate',axis = 1, inplace=True)

#This variable is highly correlated with srv\_serror\_rate and should be ignored for analysis.

#(Correlation = 0.9993041091850098)

data.drop('dst\_host\_srv\_serror\_rate',axis = 1, inplace=True)

#This variable is highly correlated with rerror\_rate and should be ignored for analysis.

#(Correlation = 0.9869947924956001)

data.drop('dst\_host\_serror\_rate',axis = 1, inplace=True)

#This variable is highly correlated with srv\_rerror\_rate and should be ignored for analysis.

#(Correlation = 0.9821663427308375)

data.drop('dst\_host\_rerror\_rate',axis = 1, inplace=True)

#This variable is highly correlated with rerror\_rate and should be ignored for analysis.

#(Correlation = 0.9851995540751249)

data.drop('dst\_host\_srv\_rerror\_rate',axis = 1, inplace=True)

#This variable is highly correlated with srv\_rerror\_rate and should be ignored for analysis.

#(Correlation = 0.9865705438845669)

data.drop('dst\_host\_same\_srv\_rate',axis = 1, inplace=True)

data.drop('service',axis = 1,inplace= True)

### Feature Selection And Feature Engineering

### # after removal of redundant features, remaining features considered to have high variance against dependent feature.

### # for feature engineering. Two types of features found here. One is numeric and another is categorical. Below codes are used to transform categorical features to numeric.

### #Finding categorical features

### num\_cols = data.\_get\_numeric\_data().columns

### print(len(num\_cols))

### cate\_cols = list(set(data.columns)-set(num\_cols))

### cate\_cols.remove('target')

### cate\_cols.remove('Attack Type')

### print(len(cate\_cols))

### print(cate\_cols)

### # before feature selection and feature engineering.

### data.dtypes

### #results

**‘’’** duration int64

protocol\_type object

service object

flag object

src\_bytes int64

dst\_bytes int64

land int64

wrong\_fragment int64

urgent int64

hot int64

num\_failed\_logins int64

logged\_in int64

num\_compromised int64

root\_shell int64

su\_attempted int64

num\_root int64

num\_file\_creations int64

num\_shells int64

num\_access\_files int64

num\_outbound\_cmds int64

is\_host\_login int64

is\_guest\_login int64

count int64

srv\_count int64

serror\_rate float64

srv\_serror\_rate float64

rerror\_rate float64

srv\_rerror\_rate float64

same\_srv\_rate float64

diff\_srv\_rate float64

srv\_diff\_host\_rate float64

dst\_host\_count int64

dst\_host\_srv\_count int64

dst\_host\_same\_srv\_rate float64

dst\_host\_diff\_srv\_rate float64

dst\_host\_same\_src\_port\_rate float64

dst\_host\_srv\_diff\_host\_rate float64

dst\_host\_serror\_rate float64

dst\_host\_srv\_serror\_rate float64

dst\_host\_rerror\_rate float64

dst\_host\_srv\_rerror\_rate float64

target object

Attack Type object

dtype: object

### ’’’

### #protocol\_type feature mapping

### pmap = {'icmp':0,'tcp':1,'udp':2}

### data['protocol\_type'] = data['protocol\_type'].map(pmap)

### #flag feature mapping

### fmap = {'SF':0,'S0':1,'REJ':2,'RSTR':3,'RSTO':4,'SH':5 ,'S1':6 ,'S2':7,'RSTOS0':8,'S3':9 ,'OTH':10}

### data['flag'] = data['flag'].map(fmap)

#protocol\_type feature mapping

attackmap = {'normal':0,'dos':1,'probe':2,'r2l':3,'u2r':4}

data['Attack Type'] = data['Attack Type'].map(attackmap)

data['Attack Type'].value\_counts()

# after feature engineering and feature selection

‘’’

duration int64

protocol\_type int64

flag int64

src\_bytes int64

dst\_bytes int64

land int64

wrong\_fragment int64

urgent int64

hot int64

num\_failed\_logins int64

logged\_in int64

num\_compromised int64

root\_shell int64

su\_attempted int64

num\_file\_creations int64

num\_shells int64

num\_access\_files int64

num\_outbound\_cmds int64

is\_host\_login int64

is\_guest\_login int64

count int64

srv\_count int64

serror\_rate float64

rerror\_rate float64

same\_srv\_rate float64

diff\_srv\_rate float64

srv\_diff\_host\_rate float64

dst\_host\_count int64

dst\_host\_srv\_count int64

dst\_host\_diff\_srv\_rate float64

dst\_host\_same\_src\_port\_rate float64

dst\_host\_srv\_diff\_host\_rate float64

target object

Attack Type int64

dtype: object

’’’

data = data.drop(['target',], axis=1)

**Write interpretations & modelling data prep recommendations based on the data summary**

1. Dependent feature distribution is imbalanced. solution : could have handled using SMOTE, if would have followed classic ML models. but here i'm doing DNN , so will leave it to NN models. but yes, if time would have given more, would have tried balancing.
2. Numeric feature distribution has different distribution. solution : each identified feature can be scaled using log-transformation or Z-score. but here again have not tried.
3. Cateforical feature distribution has imbalanced distribution. solution : prefered to not do anything.
4. After observing corrleation score, removed redudant features. also constant features.
5. Also Random Forest Regressor can be tried to remove other irrelevant features. but again as i'm trying DNN, so prefered not to touch. reason : some variable may look undesired, but some times they contribute to model performance.
6. for feature engineering, did label-encoding for categorical features. and for numerical features, didn't make any changes. but it would have been better to do different scaling operations like log scale, z-score as some of features standard deviations are little differnt than other.
7. but did Normalize samples individually to unit norm.Each sample (i.e. each row of the data matrix) with at least one non zero component is rescaled independently of other samples so that its norm (l1, l2 or inf) equals one. why ? to make each sample feature close to one in feature space.

**Create a flag field named ‘Suspicious’ in the dataset defined as -- More than 2 failed logins and loss of data occurs when using file transfer protocol. Use R or Python or SQL**

#Adding Suspicious column

data['Suspicious'] = data.apply(lambda row: 1 if row.num\_failed\_logins > 1 and row.wrong\_fragment > 0 else 0, axis=1)

data.head()

data['Suspicious'].unique()

# Suspicious field only contain single value. So, droping this one as it’ll not contribute to model.

data.drop('Suspicious',axis = 1,inplace= True)

### Modeling using Tensorflow(Keras) that best fits the data.

from \_\_future\_\_ import print\_function

#from sklearn.cross\_validation import train\_test\_split

import pandas as pd

import numpy as np

np.random.seed(1337) # for reproducibility

from keras.preprocessing import sequence

from keras.utils import np\_utils

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Embedding

from keras.layers import LSTM, SimpleRNN, GRU

from keras.datasets import imdb

from keras.utils.np\_utils import to\_categorical

from sklearn.metrics import (precision\_score, recall\_score,f1\_score, accuracy\_score,mean\_squared\_error,mean\_absolute\_error)

from sklearn import metrics

from sklearn.preprocessing import Normalizer

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

import h5py

from keras import callbacks

from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger

y = data[['Attack Type']]

X = data.drop(['Attack Type',], axis=1)

# Split test and train data

X, C, Y, T = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

trainX = np.array(X)

testT = np.array(C)

trainX.astype(float)

testT.astype(float)

scaler = Normalizer().fit(trainX)

trainX = scaler.transform(trainX)

scaler = Normalizer().fit(testT)

testT = scaler.transform(testT)

y\_train1 = np.array(Y)

y\_test1 = np.array(T)

print(y\_train1.shape)

print(y\_test1.shape)

# encode class values as integers

encoder = LabelEncoder()

encoder.fit(y\_train1)

encoded\_Ytr = encoder.transform(y\_train1)

encoded\_Yte = encoder.transform(y\_test1)

# convert integers to dummy variables (i.e. one hot encoded)

y\_train = np\_utils.to\_categorical(encoded\_Ytr)

y\_test = np\_utils.to\_categorical(encoded\_Yte)

#y\_train= to\_categorical(y\_train1)

#y\_test= to\_categorical(y\_test1)

X\_train = np.array(trainX)

X\_test = np.array(testT)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

batch\_size = 64

# 1. define the network

model = Sequential()

model.add(Dense(1024,input\_dim=32,activation='relu'))

model.add(Dropout(0.01))

model.add(Dense(5))

model.add(Activation('softmax'))

# try using different optimizers and different optimizer configs

model.compile(loss='categorical\_crossentropy',optimizer='adam',metrics=['accuracy'])

checkpointer = callbacks.ModelCheckpoint(filepath="dnn1layer/checkpoint-{epoch:02d}.hdf5", verbose=1, save\_best\_only=True, monitor='loss')

#csv\_logger = CSVLogger('kddresults/dnn1layer/training\_set\_dnnanalysis.csv',separator=',', append=False)

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test),batch\_size=batch\_size, nb\_epoch=100, callbacks=[checkpointer])#,csv\_logger

# Sample result

‘’’

(330994, 1)

(163027, 1)

(330994, 32)

(163027, 32)

(330994, 5)

(163027, 5)

WARNING:tensorflow:From E:\data\_structure\myenv\lib\site-packages\tensorflow\python\ops\resource\_variable\_ops.py:435: colocate\_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.

Instructions for updating:

Colocations handled automatically by placer.

E:\data\_structure\myenv\lib\site-packages\sklearn\preprocessing\\_label.py:235: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n\_samples, ), for example using ravel().

y = column\_or\_1d(y, warn=True)

E:\data\_structure\myenv\lib\site-packages\sklearn\preprocessing\\_label.py:268: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n\_samples, ), for example using ravel().

y = column\_or\_1d(y, warn=True)

WARNING:tensorflow:From E:\data\_structure\myenv\lib\site-packages\tensorflow\python\ops\math\_ops.py:3066: to\_int32 (from tensorflow.python.ops.math\_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Use tf.cast instead.

E:\data\_structure\myenv\lib\site-packages\ipykernel\_launcher.py:51: UserWarning: The `nb\_epoch` argument in `fit` has been renamed `epochs`.

Train on 330994 samples, validate on 163027 samples

Epoch 1/100

330994/330994 [==============================] - 20s 61us/step - loss: 0.0401 - accuracy: 0.9887 - val\_loss: 0.0208 - val\_accuracy: 0.9912

Epoch 00001: loss improved from inf to 0.04012, saving model to dnn1layer/checkpoint-01.hdf5

Epoch 2/100

330994/330994 [==============================] - 20s 61us/step - loss: 0.0175 - accuracy: 0.9951 - val\_loss: 0.0141 - val\_accuracy: 0.9961

Epoch 00002: loss improved from 0.04012 to 0.01755, saving model to dnn1layer/checkpoint-02.hdf5

Epoch 3/100

330994/330994 [==============================] - 20s 61us/step - loss: 0.0135 - accuracy: 0.9962 - val\_loss: 0.0121 - val\_accuracy: 0.9967

Epoch 00003: loss improved from 0.01755 to 0.01348, saving model to dnn1layer/checkpoint-03.hdf5

Epoch 4/100

330994/330994 [==============================] - 20s 61us/step - loss: 0.0110 - accuracy: 0.9969 - val\_loss: 0.0104 - val\_accuracy: 0.9969

Epoch 00004: loss improved from 0.01348 to 0.01101, saving model to dnn1layer/checkpoint-04.hdf5

Epoch 5/100

330994/330994 [==============================] - 20s 61us/step - loss: 0.0096 - accuracy: 0.9973 - val\_loss: 0.0080 - val\_accuracy: 0.9978

Epoch 00005: loss improved from 0.01101 to 0.00958, saving model to dnn1layer/checkpoint-05.hdf5

Epoch 6/100

330994/330994 [==============================] - 20s 61us/step - loss: 0.0085 - accuracy: 0.9977 - val\_loss: 0.0079 - val\_accuracy: 0.9980

Epoch 00006: loss improved from 0.00958 to 0.00849, saving model to dnn1layer/checkpoint-06.hdf5

Epoch 7/100

330994/330994 [==============================] - 20s 62us/step - loss: 0.0077 - accuracy: 0.9979 - val\_loss: 0.0072 - val\_accuracy: 0.9978

Epoch 00007: loss improved from 0.00849 to 0.00774, saving model to dnn1layer/checkpoint-07.hdf5

Epoch 8/100

330994/330994 [==============================] - 20s 61us/step - loss: 0.0072 - accuracy: 0.9980 - val\_loss: 0.0075 - val\_accuracy: 0.9982

Epoch 00008: loss improved from 0.00774 to 0.00723, saving model to dnn1layer/checkpoint-08.hdf5

Epoch 9/100

330994/330994 [==============================] - 21s 62us/step - loss: 0.0068 - accuracy: 0.9982 - val\_loss: 0.0077 - val\_accuracy: 0.9983

Epoch 00009: loss improved from 0.00723 to 0.00683, saving model to dnn1layer/checkpoint-09.hdf5

Epoch 10/100

330994/330994 [==============================] - 24s 72us/step - loss: 0.0063 - accuracy: 0.9983 - val\_loss: 0.0057 - val\_accuracy: 0.9987

Epoch 00024: loss improved from 0.00399 to 0.00387, saving model to dnn1layer/checkpoint-24.hdf5

Epoch 25/100

330994/330994 [==============================] - 22s 65us/step - loss: 0.0039 - accuracy: 0.9990 - val\_loss: 0.0036 - val\_accuracy: 0.9990

Epoch 00025: loss improved from 0.00387 to 0.00386, saving model to dnn1layer/checkpoint-25.hdf5

Epoch 26/100

330994/330994 [==============================] - 21s 63us/step - loss: 0.0038 - accuracy: 0.9990 - val\_loss: 0.0040 - val\_accuracy: 0.9991

Epoch 00026: loss improved from 0.00386 to 0.00383, saving model to dnn1layer/checkpoint-26.hdf5

Epoch 27/100

330994/330994 [==============================] - 20s 62us/step - loss: 0.0038 - accuracy: 0.9990 - val\_loss: 0.0035 - val\_accuracy: 0.9992

Epoch 00027: loss improved from 0.00383 to 0.00379, saving model to dnn1layer/checkpoint-27.hdf5

Epoch 28/100

330994/330994 [==============================] - 21s 63us/step - loss: 0.0036 - accuracy: 0.9991 - val\_loss: 0.0042 - val\_accuracy: 0.9991

Epoch 00028: loss improved from 0.00379 to 0.00359, saving model to dnn1layer/checkpoint-28.hdf5

Epoch 29/100

330994/330994 [==============================] - 21s 62us/step - loss: 0.0037 - accuracy: 0.9990 - val\_loss: 0.0036 - val\_accuracy: 0.9992

Epoch 00029: loss did not improve from 0.00359

Epoch 30/100

330994/330994 [==============================] - 20s 62us/step - loss: 0.0035 - accuracy: 0.9991 - val\_loss: 0.0038 - val\_accuracy: 0.9990

Epoch 00030: loss improved from 0.00359 to 0.00351, saving model to dnn1layer/checkpoint-30.hdf5

Epoch 31/100

330994/330994 [==============================] - 21s 63us/step - loss: 0.0035 - accuracy: 0.9991 - val\_loss: 0.0040 - val\_accuracy: 0.9990

Epoch 00031: loss improved from 0.00351 to 0.00345, saving model to dnn1layer/checkpoint-31.hdf5

Epoch 32/100

330994/330994 [==============================] - 20s 62us/step - loss: 0.0034 - accuracy: 0.9991 - val\_loss: 0.0039 - val\_accuracy: 0.9990

Epoch 00032: loss improved from 0.00345 to 0.00343, saving model to dnn1layer/checkpoint-32.hdf5

Epoch 33/100

330994/330994 [==============================] - 21s 63us/step - loss: 0.0033 - accuracy: 0.9992 - val\_loss: 0.0037 - val\_accuracy: 0.9991

Epoch 00033: loss improved from 0.00343 to 0.00327, saving model to dnn1layer/checkpoint-33.hdf5

Epoch 00096: loss did not improve from 0.00192

Epoch 97/100

330994/330994 [==============================] - 21s 63us/step - loss: 0.0020 - accuracy: 0.9995 - val\_loss: 0.0027 - val\_accuracy: 0.9994

Epoch 00097: loss did not improve from 0.00192

Epoch 98/100

330994/330994 [==============================] - 20s 61us/step - loss: 0.0020 - accuracy: 0.9995 - val\_loss: 0.0030 - val\_accuracy: 0.9992

Epoch 00098: loss did not improve from 0.00192

Epoch 99/100

330994/330994 [==============================] - 21s 65us/step - loss: 0.0019 - accuracy: 0.9995 - val\_loss: 0.0035 - val\_accuracy: 0.9992

Epoch 00099: loss did not improve from 0.00192

Epoch 100/100

330994/330994 [==============================] - 22s 65us/step - loss: 0.0019 - accuracy: 0.9995 - val\_loss: 0.0029 - val\_accuracy: 0.9994

Epoch 00100: loss did not improve from 0.00192

’’’

**Error Vs Accuracy Curve**

[error vs accuracy curve.png](error%20vs%20accuracy%20curve.png)

**Test Result :**

model.evaluate(X\_test, y\_test)

# result

‘’’

[0.0038743916104425854, 0.9991780519485474]

[error %ge, Accuracy %ge]

‘’’

### 5. Do the hyperparameter tuning and show the convergence of the model using an error vs. epochs graph

from sklearn.model\_selection import GridSearchCV

#from keras.models import Sequential

#from keras.layers import Dense

from keras.wrappers.scikit\_learn import KerasClassifier

#### **Tune Batch Size and Number of Epochs**

def create\_model(init\_mode='uniform'):

# create model

model = Sequential()

model.add(Dense(1024,input\_dim=32,activation='relu'))

model.add(Dropout(0.01))

model.add(Dense(5))

model.add(Activation('softmax'))

# try using different optimizers and different optimizer configs

model.compile(loss='categorical\_crossentropy',optimizer='adam',metrics=['accuracy'])

return model

# create model

model = KerasClassifier(build\_fn=create\_model, verbose=0)

# define the grid search parameters

batch\_size = [10, 20, 40, 60, 80, 100]

epochs = [10, 50, 100]

param\_grid = dict(batch\_size=batch\_size, epochs=epochs)

grid = GridSearchCV(estimator=model, param\_grid=param\_grid, n\_jobs=-1, cv=3)

grid\_result = grid.fit(X\_train, y\_train)

# summarize results

print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))

means = grid\_result.cv\_results\_['mean\_test\_score']

stds = grid\_result.cv\_results\_['std\_test\_score']

params = grid\_result.cv\_results\_['params']

for mean, stdev, param in zip(means, stds, params):

print("%f (%f) with: %r" % (mean, stdev, param))

#### **Tune the Training Optimization Algorithm**

def create\_model(optimizer='adam'):

# create model

model = Sequential()

model.add(Dense(1024,input\_dim=32,activation='relu'))

model.add(Dropout(0.01))

model.add(Dense(5))

model.add(Activation('softmax'))

# try using different optimizers and different optimizer configs

model.compile(loss='categorical\_crossentropy',optimizer='adam',metrics=['accuracy'])

return model

# create model

model = KerasClassifier(build\_fn=create\_model, epochs=100, batch\_size=10, verbose=0)

# define the grid search parameters

optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']

param\_grid = dict(optimizer=optimizer)

grid = GridSearchCV(estimator=model, param\_grid=param\_grid, n\_jobs=-1, cv=3)

grid\_result = grid.fit(X\_train, y\_train)

# summarize results

print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))

means = grid\_result.cv\_results\_['mean\_test\_score']

stds = grid\_result.cv\_results\_['std\_test\_score']

params = grid\_result.cv\_results\_['params']

for mean, stdev, param in zip(means, stds, params):

print("%f (%f) with: %r" % (mean, stdev, param))

#### **Tune Learning Rate and Momentum**

def create\_model(learn\_rate=0.01, momentum=0):

# create model

model = Sequential()

model.add(Dense(1024,input\_dim=32,activation='relu'))

model.add(Dropout(0.01))

model.add(Dense(5))

model.add(Activation('softmax'))

# try using different optimizers and different optimizer configs

model.compile(loss='categorical\_crossentropy',optimizer='adam',metrics=['accuracy'])

return model

# create model

model = KerasClassifier(build\_fn=create\_model, epochs=100, batch\_size=10, verbose=0)

# define the grid search parameters

learn\_rate = [0.001, 0.01, 0.1, 0.2, 0.3]

momentum = [0.0, 0.2, 0.4, 0.6, 0.8, 0.9]

param\_grid = dict(learn\_rate=learn\_rate, momentum=momentum)

grid = GridSearchCV(estimator=model, param\_grid=param\_grid, n\_jobs=-1, cv=3)

grid\_result = grid.fit(X\_train, y\_train)

# summarize results

print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))

means = grid\_result.cv\_results\_['mean\_test\_score']

stds = grid\_result.cv\_results\_['std\_test\_score']

params = grid\_result.cv\_results\_['params']

for mean, stdev, param in zip(means, stds, params):

print("%f (%f) with: %r" % (mean, stdev, param))

#### **Tune the Number of Neurons in the Hidden Layer**

def create\_model(neuron=1):

# create model

model = Sequential()

model.add(Dense(1024,input\_dim=32,activation='relu'))

model.add(Dropout(0.01))

model.add(Dense(5))

model.add(Activation('softmax'))

# try using different optimizers and different optimizer configs

model.compile(loss='categorical\_crossentropy',optimizer='adam',metrics=['accuracy'])

return model

# create model

model = KerasClassifier(build\_fn=create\_model, epochs=100, batch\_size=10, verbose=0)

# define the grid search parameters

neurons = [1, 5, 10, 15, 20, 25, 30]

param\_grid = dict(neurons=neurons)

grid = GridSearchCV(estimator=model, param\_grid=param\_grid, n\_jobs=-1, cv=3)

grid\_result = grid.fit(X\_train, y\_train)

# summarize results

print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))

means = grid\_result.cv\_results\_['mean\_test\_score']

stds = grid\_result.cv\_results\_['std\_test\_score']

params = grid\_result.cv\_results\_['params']

for mean, stdev, param in zip(means, stds, params):

print("%f (%f) with: %r" % (mean, stdev, param))

### Serialize[¶](http://localhost:8888/notebooks/data_structure/case%20study%20L%26T/case%20study/Deloite%20Test%202-Copy1.ipynb#Serielize)

model.save("dnn1layer/dnn1layer\_model.hdf5")

### Deploy on Azure DevOps

#NOTE: don’t have Azure machine for deployment. With that , below code for Flask application for model service.

from \_\_future\_\_ import print\_function

import pandas as pd

import numpy as np

np.random.seed(1337) # for reproducibility

from keras.preprocessing import sequence

from keras.utils import np\_utils

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Embedding

from keras.layers import LSTM, SimpleRNN, GRU

from keras.utils.np\_utils import to\_categorical

from sklearn.preprocessing import Normalizer

from sklearn.preprocessing import MinMaxScaler

import h5py

from numpy import loadtxt

from keras.models import load\_model

import tensorflow as tf

import keras

# Flask Application for REST service

from flask import Flask, request, Response, jsonify, redirect, render\_template, url\_for

import json

app = Flask(\_\_name\_\_)

FIELDS = {} # mandatory fields

attackmap = {'normal':0,'dos':1,'probe':2,'r2l':3,'u2r':4}

attackmap\_rev = {v: k for k, v in attackmap.items()}

# load numeric\_variables\_zscore\_attributes for preprocessing.

with open('numeric\_variables\_zscore\_attributes.json','r') as f:

num\_var\_zs\_attr = json.load(f)

# load model

config = tf.ConfigProto(

device\_count={'CPU': 1},

intra\_op\_parallelism\_threads=1,

allow\_soft\_placement=True

)

config.gpu\_options.allow\_growth = True

config.gpu\_options.per\_process\_gpu\_memory\_fraction = 0.6

session = tf.Session(config=config)

keras.backend.set\_session(session)

model = load\_model('dnn1layer/dnn1layer\_model.hdf5')

def create\_validation\_logic(body):

if not body:

return None

return body

def preprocess(body):

# for now body : let be a sample data for test.

#dict\_ = body

# load and evaluate a saved model

with open("pred\_input.json",'r') as f:

dict\_ = json.load(f)

data = pd.DataFrame([dict\_])

# Numeric Feature scale with Z-score

def encode\_numeric\_zscore(data, name, mean=None, sd=None):

if mean is None:

mean = data[name].mean()

if sd is None:

sd = data[name].std()

data[name] = (data[name] - mean) / sd

encode\_numeric\_zscore(data, 'duration',num\_var\_zs\_attr['duration']['mean'],num\_var\_zs\_attr['duration']['sd'])

encode\_numeric\_zscore(data, 'src\_bytes',num\_var\_zs\_attr['src\_bytes']['mean'],num\_var\_zs\_attr['src\_bytes']['sd'])

encode\_numeric\_zscore(data, 'dst\_bytes',num\_var\_zs\_attr['dst\_bytes']['mean'],num\_var\_zs\_attr['dst\_bytes']['sd'])

encode\_numeric\_zscore(data, 'wrong\_fragment',num\_var\_zs\_attr['wrong\_fragment']['mean'],num\_var\_zs\_attr['wrong\_fragment']['sd'])

encode\_numeric\_zscore(data, 'urgent',num\_var\_zs\_attr['urgent']['mean'],num\_var\_zs\_attr['urgent']['sd'])

encode\_numeric\_zscore(data, 'hot',num\_var\_zs\_attr['hot']['mean'],num\_var\_zs\_attr['hot']['sd'])

encode\_numeric\_zscore(data, 'num\_failed\_logins',num\_var\_zs\_attr['num\_failed\_logins']['mean'],num\_var\_zs\_attr['num\_failed\_logins']['sd'])

encode\_numeric\_zscore(data, 'num\_compromised',num\_var\_zs\_attr['num\_compromised']['mean'],num\_var\_zs\_attr['num\_compromised']['sd'])

encode\_numeric\_zscore(data, 'root\_shell',num\_var\_zs\_attr['root\_shell']['mean'],num\_var\_zs\_attr['root\_shell']['sd'])

encode\_numeric\_zscore(data, 'su\_attempted',num\_var\_zs\_attr['su\_attempted']['mean'],num\_var\_zs\_attr['su\_attempted']['sd'])

encode\_numeric\_zscore(data, 'num\_root',num\_var\_zs\_attr['num\_root']['mean'],num\_var\_zs\_attr['num\_root']['sd'])

encode\_numeric\_zscore(data, 'num\_file\_creations',num\_var\_zs\_attr['num\_file\_creations']['mean'],num\_var\_zs\_attr['num\_file\_creations']['sd'])

encode\_numeric\_zscore(data, 'num\_shells',num\_var\_zs\_attr['num\_shells']['mean'],num\_var\_zs\_attr['num\_shells']['sd'])

encode\_numeric\_zscore(data, 'num\_access\_files',num\_var\_zs\_attr['num\_access\_files']['mean'],num\_var\_zs\_attr['num\_access\_files']['sd'])

encode\_numeric\_zscore(data, 'num\_outbound\_cmds',num\_var\_zs\_attr['num\_outbound\_cmds']['mean'],num\_var\_zs\_attr['num\_outbound\_cmds']['sd'])

encode\_numeric\_zscore(data, 'count',num\_var\_zs\_attr['count']['mean'],num\_var\_zs\_attr['count']['sd'])

encode\_numeric\_zscore(data, 'srv\_count',num\_var\_zs\_attr['srv\_count']['mean'],num\_var\_zs\_attr['srv\_count']['sd'])

encode\_numeric\_zscore(data, 'serror\_rate',num\_var\_zs\_attr['serror\_rate']['mean'],num\_var\_zs\_attr['serror\_rate']['sd'])

encode\_numeric\_zscore(data, 'srv\_serror\_rate',num\_var\_zs\_attr['srv\_serror\_rate']['mean'],num\_var\_zs\_attr['srv\_serror\_rate']['sd'])

encode\_numeric\_zscore(data, 'rerror\_rate',num\_var\_zs\_attr['rerror\_rate']['mean'],num\_var\_zs\_attr['rerror\_rate']['sd'])

encode\_numeric\_zscore(data, 'srv\_rerror\_rate',num\_var\_zs\_attr['srv\_rerror\_rate']['mean'],num\_var\_zs\_attr['srv\_rerror\_rate']['sd'])

encode\_numeric\_zscore(data, 'same\_srv\_rate',num\_var\_zs\_attr['same\_srv\_rate']['mean'],num\_var\_zs\_attr['same\_srv\_rate']['sd'])

encode\_numeric\_zscore(data, 'diff\_srv\_rate',num\_var\_zs\_attr['diff\_srv\_rate']['mean'],num\_var\_zs\_attr['diff\_srv\_rate']['sd'])

encode\_numeric\_zscore(data, 'srv\_diff\_host\_rate',num\_var\_zs\_attr['srv\_diff\_host\_rate']['mean'],num\_var\_zs\_attr['srv\_diff\_host\_rate']['sd'])

encode\_numeric\_zscore(data, 'dst\_host\_count',num\_var\_zs\_attr['dst\_host\_count']['mean'],num\_var\_zs\_attr['dst\_host\_count']['sd'])

encode\_numeric\_zscore(data, 'dst\_host\_srv\_count',num\_var\_zs\_attr['dst\_host\_srv\_count']['mean'],num\_var\_zs\_attr['dst\_host\_srv\_count']['sd'])

encode\_numeric\_zscore(data, 'dst\_host\_same\_srv\_rate',num\_var\_zs\_attr['dst\_host\_same\_srv\_rate']['mean'],num\_var\_zs\_attr['dst\_host\_same\_srv\_rate']['sd'])

encode\_numeric\_zscore(data, 'dst\_host\_diff\_srv\_rate',num\_var\_zs\_attr['dst\_host\_diff\_srv\_rate']['mean'],num\_var\_zs\_attr['dst\_host\_diff\_srv\_rate']['sd'])

encode\_numeric\_zscore(data, 'dst\_host\_same\_src\_port\_rate',num\_var\_zs\_attr['dst\_host\_same\_src\_port\_rate']['mean'],num\_var\_zs\_attr['dst\_host\_same\_src\_port\_rate']['sd'])

encode\_numeric\_zscore(data, 'dst\_host\_srv\_diff\_host\_rate',num\_var\_zs\_attr['dst\_host\_srv\_diff\_host\_rate']['mean'],num\_var\_zs\_attr['dst\_host\_srv\_diff\_host\_rate']['sd'])

encode\_numeric\_zscore(data, 'dst\_host\_serror\_rate',num\_var\_zs\_attr['dst\_host\_serror\_rate']['mean'],num\_var\_zs\_attr['dst\_host\_serror\_rate']['sd'])

encode\_numeric\_zscore(data, 'dst\_host\_srv\_serror\_rate',num\_var\_zs\_attr['dst\_host\_srv\_serror\_rate']['mean'],num\_var\_zs\_attr['dst\_host\_srv\_serror\_rate']['sd'])

encode\_numeric\_zscore(data, 'dst\_host\_rerror\_rate',num\_var\_zs\_attr['dst\_host\_rerror\_rate']['mean'],num\_var\_zs\_attr['dst\_host\_rerror\_rate']['sd'])

encode\_numeric\_zscore(data, 'dst\_host\_srv\_rerror\_rate',num\_var\_zs\_attr['dst\_host\_srv\_rerror\_rate']['mean'],num\_var\_zs\_attr['dst\_host\_srv\_rerror\_rate']['sd'])

#protocol\_type feature mapping

pmap = {'icmp':0,'tcp':1,'udp':2}

data['protocol\_type'] = data['protocol\_type'].map(pmap)

#flag feature mapping

fmap = {'SF':0,'S0':1,'REJ':2,'RSTR':3,'RSTO':4,'SH':5 ,'S1':6 ,'S2':7,'RSTOS0':8,'S3':9 ,'OTH':10}

data['flag'] = data['flag'].map(fmap)

#protocol\_type feature mapping

attackmap = {'normal':0,'dos':1,'probe':2,'r2l':3,'u2r':4}

data['Attack Type'] = data['Attack Type'].map(attackmap)

data['Attack Type'].value\_counts()

reqd\_cols\_for\_pred = ['duration','protocol\_type','flag','src\_bytes','dst\_bytes','land','wrong\_fragment','urgent','hot','num\_failed\_logins','logged\_in','num\_compromised','root\_shell','su\_attempted','num\_file\_creations','num\_shells','num\_access\_files','is\_host\_login','is\_guest\_login','count','srv\_count','serror\_rate','rerror\_rate','same\_srv\_rate','diff\_srv\_rate','srv\_diff\_host\_rate','dst\_host\_count','dst\_host\_srv\_count','dst\_host\_diff\_srv\_rate','dst\_host\_same\_src\_port\_rate','dst\_host\_srv\_diff\_host\_rate']

data = data[reqd\_cols\_for\_pred]

print(data.shape)

testT = np.array(data)

testT.astype(float)

scaler = Normalizer().fit(testT)

testT = scaler.transform(testT)

X\_test = np.array(testT)

return X\_test

@app.route('/')

def home():

return render\_template('home.html')

@app.route('/get\_intrusion',methods = ['POST'])

def get\_intrusion():

with session.as\_default():

with session.graph.as\_default():

body = request.get\_json()

#validation\_out = create\_validation\_logic(body)

validation\_out = True # for test

if validation\_out:

# preprocess data

X\_test = preprocess(body)

# make a prediction

result = 'No prediction happen.'

result = model.predict(X\_test)

return render\_template('home.html', prediction\_text="Intrusion Type : {}".format(attackmap\_rev[result.argmax()]))

#return jsonify({"Intrusion Type":attackmap\_rev[result.argmax()]}), 201

#return '{"error": "Bad request"}', 400

return render\_template('home.html', prediction\_text="Bad Request.")

@app.route('/get\_intrusion\_api', methods=['POST'])

def get\_intrusion\_api():

with session.as\_default():

with session.graph.as\_default():

body = request.get\_json()

#validation\_out = create\_validation\_logic(body)

validation\_out = True # for test

if validation\_out:

# preprocess data

X\_test = preprocess(body)

# make a prediction

result = 'No prediction happen.'

result = model.predict(X\_test)

return jsonify({"Intrusion Type":attackmap\_rev[result.argmax()]}), 201

return '{"error": "Bad request"}', 400

if \_\_name\_\_ == '\_\_main\_\_':

#app.run(debug=True)

app.run(host="0.0.0.0", port=8080)

# Deploy models with Azure Machine Learning

Service deployed in below URL.

https://intrusion-detection.azurewebsites.net

user can try this. But default it is set to hard coded sample input. Means body part needs to be configured required. Otherwise it’s stable version.

From REST call:

Request URL : https://intrusion-detection.azurewebsites.net/get\_inrusion\_api

Method : POST

**Generate interpretations and recommendations for a client looking for solution/ process change requirements to deal with the network intrusion incidents.**

Wrote details on Interpretation & Recommendation.ppt