Machine Learning Digital Assignment 2

Name: Om Ashish Mishra

Registration Number: 16BCE0789

Slot: F2

9. Consider a classification problem in Networking domain and implement it using Multi-Layer Perceptron and Support vector Machines.

1. TITLE/HEADER

**Networking domain**

1. INTRODUCTION

### **Context**

Computer Network Traffic Data - A ~500K CSV with summary of some real network traffic data from the past. The dataset has ~21K rows and covers 10 local workstation IPs over a three month period. Half of these local IPs were compromised at some point during this period and became members of various botnets.

### **Content**

Each row consists of four columns:

* date: yyyy-mm-dd (from 2006-07-01 through 2006-09-30)
* l\_ipn: local IP (coded as an integer from 0-9)
* r\_asn: remote ASN (an integer which identifies the remote ISP)
* f: flows (count of connnections for that day)

1. IMPLEMENTATION
   1. Source Code

# Support Vector Machine (SVM)

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Network.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# 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 = 42)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Fitting SVM to the Training set

from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random\_state = 0)

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('SVM (Training set)')

plt.xlabel('IP')

plt.ylabel('Flow')

plt.legend()

plt.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('SVM (Test set)')

plt.xlabel('IP')

plt.ylabel('Flow')

plt.legend()

plt.show()

# -\*- coding: utf-8 -\*-

"""

Created on Tue Feb 12 10:35:28 2019

@author: OM MISHRA

"""

#Multi Layer Perceptron

"""

import numpy as np

#Input array of People Bank Account

X=np.array([[1,0,1,0],[1,0,1,1],[0,1,0,1]])

#Output

y=np.array([[1],[1],[0]])

#Sigmoid Function

def sigmoid (x):

return (1/(1 + np.exp(-x)))

#Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

return x \* (1 - x)

#Variable initialization

epoch=5000 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer\_neurons = X.shape[1] #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

for i in range(epoch):

#Forward Propogation

hidden\_layer\_input1=np.dot(X,wh)

hidden\_layer\_input=hidden\_layer\_input1 + bh

hiddenlayer\_activations = sigmoid(hidden\_layer\_input)

output\_layer\_input1=np.dot(hiddenlayer\_activations,wout)

output\_layer\_input= output\_layer\_input1+ bout

output = sigmoid(output\_layer\_input)

#Backpropagation

E = y-output

slope\_output\_layer = derivatives\_sigmoid(output)

slope\_hidden\_layer = derivatives\_sigmoid(hiddenlayer\_activations)

d\_output = E \* slope\_output\_layer

Error\_at\_hidden\_layer = d\_output.dot(wout.T)

d\_hiddenlayer = Error\_at\_hidden\_layer \* slope\_hidden\_layer

wout += hiddenlayer\_activations.T.dot(d\_output) \*lr

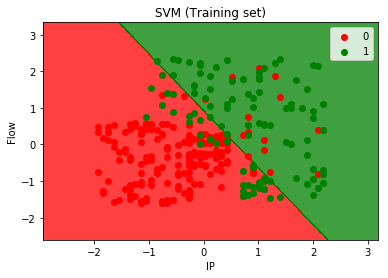
bout += np.sum(d\_output, axis=0,keepdims=True) \*lr

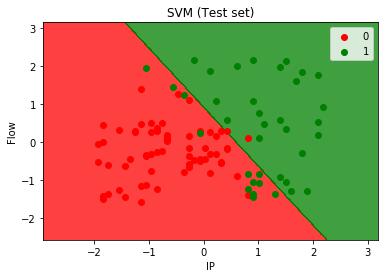
wh += X.T.dot(d\_hiddenlayer) \*lr

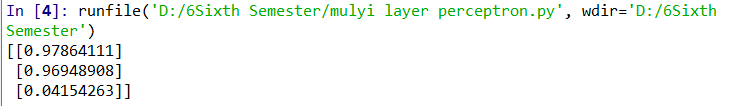
bh += np.sum(d\_hiddenlayer, axis=0,keepdims=True) \*lr

print (output)

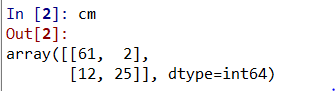
* 1. Snapshots

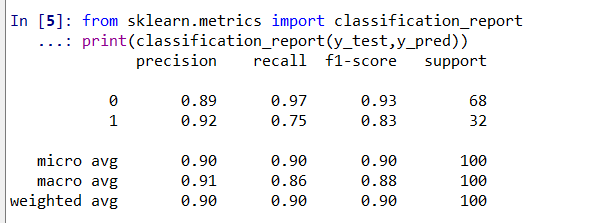






* 1. Conclusion





1. REFERENCES
2. <http://www.crraoaimscs.org/lecture_notes/VR_Lecture_2.pdf>
3. <https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es1999-463.pdf>
4. <https://www.researchgate.net/publication/270345247_Selection_of_Support_Vector_Machines_based_classifiers_for_credit_risk_domain>
5. <https://dl.acm.org/citation.cfm?id=1465151>