Python Project Document

150101027- I.N.Dilip Kumar
150101051- Hareesh Reddi
150101032- L.Sai Shobith

April 10, 2017

Contents

1 Introduction **3**

2 Algorithm **4**

3 Algorithm Analysis **5**

4 Code Description **6**
4.1 Section1 6
4.2 Section2 6
4.3 Section3 6
4.4 Section4 7

1 Introduction

In the given attack data file, each category has 10 directories. We select 7 directories for the training and use the other 3 directories for testing.
The problem statement is:

Split the Attack data of each category (Hydra-FTP, Hydra-SSH, Adduser, Java-Meterpreter, Meterpreter and Webshell) into 70% training data and 30 % test data. For instance there are are 10 folders in "Adduser" attack. Therefore, 7 of these folders are to be used for training and 3 folders are to be used for testing.

For the Normal data, files in "Training_Data_Master" folder are to be used as training data and files in "Validation_Data_Master" folder are to be used as test data.

Write a python script to find the frequency of occurences of all unique 3-grams, 5-grams and 7-grams system call sequences in the training data for both Attack data (across all categories of attack) and Normal data.

Perform the same task on files in the T̈raining_Data_Masterẗo obtain all the unique 3-grams, 5-grams and 7-grams.

Once you have obtained the frequencies of all the unique n-grams terms in the training data,use the top 30% n-grams terms with the highest frequency to create a data set.

Apply the same procedure to generate the test dataset from the test files of the attack data (for all attack types) and the normal files in the "Validation_Data_Master" using the top 30% 3-grams terms with highest frequencies obtained during the training phase. The classifier model developed during the training phase will finally be validated on the Test dataset.

2 Algorithm

1. Select the first seven folders for each category and find all the text files in all the directories.
2. Find all the seven grams for each file.
3. Find the frequency of each seven gram and keep them in a dictionary.
4. Using the seven grams, find the 5 grams and 3 grams by taking the first three numbers for 3 gram and first five for 5 gram.
5. Find the frequency for each of the 5 grams and for 3 grams and keep them in a dictionary.
6. Sort the dictionary based on the values as frequencies.
7. Select the top 30 percent of the n-grams and do this for each category and for the training data set.
8. Concat all of the top 30 percent n-grams and keep them in a single dictionary and let us call it feature file.
9. Now for each text file in the test data and validation data master create another text file which contains the 7 gram in the feature file and if it is in the text file write it's frequency and if it is not present write 0 as the frequency.

3 Algorithm Analysis

1. O(No.Of.TextFiles) is the time complexity of first step
2. O(No.Of.SevenGrams) is the time complexity of second step
3. O(No.Of.SevenGrams) is the time complexity for finding the frequencies and O(NlogN) where N is No.Of.SevenGrams for sorting the dictionaries
4. O(No.Of.ThreeGrams + No.Of.FiveGrams)<=O(No.Of.SevenGrams) for the fourth step
5. O(No.Of.ThreeGrams + No.Of.FiveGrams) is the time complexity of fifth step
6. O(NlogN) where N is No.Of.SevenGrams is time complexity for sorting the dictionaries
7. O(N) where N is No.Of.SevenGrams is time complexity for selecting the top 30% N-Grams from the dictionaries
8. O(N) where N is No.Of.SevenGrams is time complexity for concatenating the dictionaries
9. O(N*No.Of.TextFiles) is the time complexity for the ninth step

4 Code Description

This is the code description

4.1 Section1

The unify(Adduser) will return three dictionaries of 7 tuples, 5tuples, 3tuples as keys and their frequencies as values

```
def unify(specific_string):
    dictionary_7 = collections.OrderedDict()
    dictionary_5 = collections.OrderedDict()
    dictionary_3 = collections.OrderedDict()
    list_of_text_files = get_Attack_Data_Master_files(specific_string)
    for text_file in list_of_text_files:
        grams_7_list = list(ngrams(text_file.split(), 7))
        for item_gram in grams_7_list:
            if item_gram in dictionary_7:
                dictionary_7[item_gram] = dictionary_7[item_gram] + 1
            else:
                dictionary_7[item_gram] = 1
        grams_5_list = list(ngrams(text_file.split(), 5))
        for item_gram in grams_5_list:
            if item_gram in dictionary_5:
                dictionary_5[item_gram] = dictionary_5[item_gram] + 1
            else:
                dictionary_5[item_gram] = 1
        grams_3_list = list(ngrams(text_file.split(), 3))
        for item_gram in grams_3_list:
            if item_gram in dictionary_3:
                dictionary_3[item_gram] = dictionary_3[item_gram] + 1
            else:
                dictionary_3[item_gram] = 1

    length(dictionary_7, dictionary_5, dictionary_3, specific_string)
    return dictionary_7, dictionary_5, dictionary_3
```

Figure 1: Fig1

4.2 Section2

The Sorted() function will sort the dictionary based on their values in decreasing order and returns the list of keys in that order

```
Sorted_List_Training = sorted(dictionary_Training, key=dictionary_Training.__getitem__, reverse = True)
Sorted_List_Adduser = sorted(dictionary_Adduser, key=dictionary_Adduser.__getitem__, reverse = True)
Sorted_List_HydraFTP = sorted(dictionary_HydraFTP, key=dictionary_HydraFTP.__getitem__, reverse = True)
Sorted_List_HydraSSH = sorted(dictionary_HydraSSH, key=dictionary_HydraSSH.__getitem__, reverse = True)
Sorted_List_JavaMeterpreter = sorted(dictionary_JavaMeterpreter, key=dictionary_JavaMeterpreter.__getitem__, reverse = True)
Sorted_List_Meterpreter = sorted(dictionary_Meterpreter, key=dictionary_Meterpreter.__getitem__, reverse = True)
Sorted_List_WebShell = sorted(dictionary_WebShell, key=dictionary_WebShell.__getitem__, reverse = True)
```

Figure 2: Fig2

4.3 Section3

The Sorted_List_Adduser is a list which contains the tuples in sorted order of their frequencies The same process above is done for each type of attack

```
Sorted_List_All_Top30=collections.OrderedDict()
for s in Sorted_List_Adduser[:int(0.3*len(Sorted_List_Adduser))]:
    if s in Sorted_List_All_Top30:
        continue
    else:
        Sorted_List_All_Top30[s] = 1
for s in Sorted_List_HydraFTP[:int(0.3*len(Sorted_List_HydraFTP))]:
    if s in Sorted_List_All_Top30:
        continue
    else:
        Sorted_List_All_Top30[s] = 1
for s in Sorted_List_HydraSSH[:int(0.3*len(Sorted_List_HydraSSH))]:
    if s in Sorted_List_All_Top30:
        continue
    else:
        Sorted_List_All_Top30[s] = 1
for s in Sorted_List_JavaMeterpreter[:int(0.3*len(Sorted_List_JavaMeterpreter))]:
    if s in Sorted_List_All_Top30:
        continue
    else:
        Sorted_List_All_Top30[s] = 1
for s in Sorted_List_Meterpreter[:int(0.3*len(Sorted_List_Meterpreter))]:
    if s in Sorted_List_All_Top30:
        continue
    else:
        Sorted_List_All_Top30[s] = 1
for s in Sorted_List_WebShell[:int(0.3*len(Sorted_List_WebShell))]:
    if s in Sorted_List_All_Top30:
        continue
    else:
        Sorted_List_All_Top30[s] = 1
for s in Sorted_List_Training[:int(0.3*len(Sorted_List_Training))]:
    if s in Sorted_List_All_Top30:
        continue
    else:
        Sorted_List_All_Top30[s] = 1
```

Figure 3: Fig3

4.4 Section4

A dictionary is formed with tuples whose frequencies are top 30 percent in the sorted dictionary for each kind of attack and concatenated into one and is called the feature set Now the feature set is tested against each file in the Adduser folder and a file is created in which if the tuple in the feature set is present in the n-grams of the selected file then the n-gram and its frequency in the selected file is printed, if it is not present then it will print the n-gram and 0 as the frequency The same is done for 5 gram and 3 gram

```
for file_object in  sec_get_Attack_Data_Master_files("Adduser"):
    new_file_path = get_new_file_ob(file_object,"7")
    list_new_file ={}
    file_data = file_object.read()
    for x in list(ngrams(file_data.split(), 7)):
        if x in list_new_file:
            list_new_file[x] += 1
        else:
            list_new_file[x] = 1
    with open(new_file_path, "w") as f:
        for key,value in Sorted_List_All_Top30.items():
            if key in list_new_file:
                f.write('%s : %d\n' % (str(key), list_new_file[key]))
            else:
                f.write('%s : 0\n' % str(key))
    new_file_path = get_new_file_ob(file_object,"5")
    list_new_file ={}
    file_data = file_object.read()
    for x in list(ngrams(file_data.split(), 5)):
        if x in list_new_file:
            list_new_file[x] += 1
        else:
            list_new_file[x] = 1
    with open(new_file_path, "w") as f:
        for key,value in Sorted_List_All_Top30_5tuple.items():
            if key in list_new_file:
                f.write('%s : %d\n' % (str(key), list_new_file[key]))
            else:
                f.write('%s : 0\n' % str(key))
    new_file_path = get_new_file_ob(file_object,"3")
    list_new_file ={}
    file_data = file_object.read()
    for x in list(ngrams(file_data.split(), 3)):
        if x in list_new_file:
            list_new_file[x] += 1
        else:
            list_new_file[x] = 1
    with open(new_file_path, "w") as f:
        for key,value in Sorted_List_All_Top30_3tuple.items():
            if key in list_new_file:
                f.write('%s : %d\n' % (str(key), list_new_file[key]))
            else:
                f.write('%s : 0\n' % str(key))
```

Figure 4: Fig4