Bộ môn An toàn Thông tin – Khoa Mạng máy tính và Truyền thông
Trường Đại học Công nghệ Thông tin – Đại học Quốc gia thành phố Hồ Chí Minh

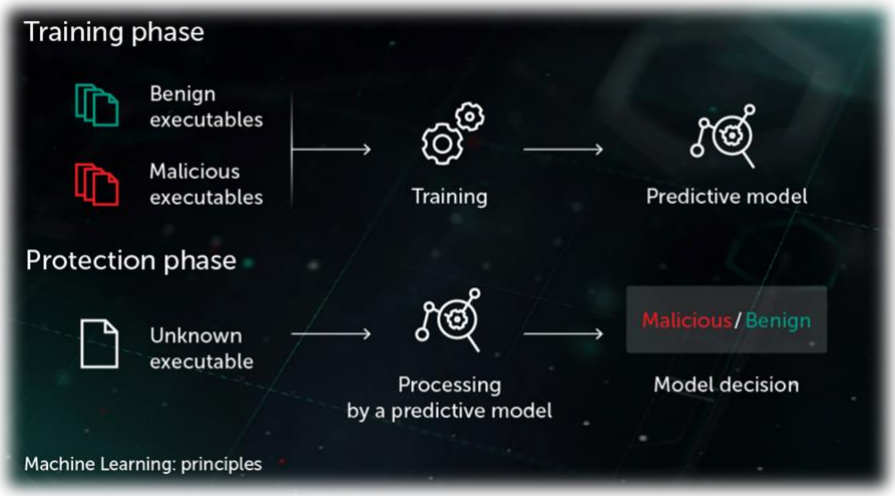
2

Lab

Machine Learning based Malware Detection

Python

Thực hành môn Phương pháp học máy trong an toàn thông tin



Tháng 10/2020

Lưu hành nội bộ

<Nghiêm cấm đăng tải trên internet dưới mọi hình thức>

Mọi góp ý về tài liệu, vui lòng gửi về email inseclab@uit.edu.vn

A. TỔNG QUAN

1. Mục tiêu

- Áp dụng thành tựu của Khoa học dữ liệu vào lĩnh vực An toàn thông tin.
- Cách tạo mẫu dữ liệu đầu vào và xây dựng các bộ phát hiện mã độc.
- Phân tích tĩnh mã độc.
- Phân tích động mã độc.
- Sử dụng mã độc để phát hiện các loại tập tin.
- Đo độ giống nhau giữa hai chuỗi.
- Đo độ giống nhau giữa hai tập tin.
- Trích xuất N-grams.
- Chọn N-gram tốt nhất.
- Xây dựng bộ phát hiện mã độc tĩnh.
- Giải quyết sự mất cân bằng.

2. Thời gian thực hành

- Thực hành tại lớp: **5** tiết tại phòng thực hành.
- Hoàn thành báo cáo kết quả thực hành: tối đa **14** ngày.

3. Môi trường thực hành

> Google Colab (https://research.google.com/colaboratory/) – *khuyến khích không bắt buộc* và thư mục tải Datasets (password: infected)

- YARA
- Pefile
- PyGitHub
- Cuckoo Sandbox
- Natural Language Toolkit (NLTK)
- imbalanced-learn

B. THỰC HÀNH

1. Phân tích tĩnh mã độc

a) Tính toán hàm băm của một mẫu

Tải tập tin mẫu cần thử nghiệm tại:
https://www.python.org/ftp/python/3.10.0/python-3.10.0-amd64.exe.

B1. Thêm các thư viện và chọn tập tin muốn băm

```
import sys
```

```
import hashlib

filename = "python-3.10.0-amd64.exe"
```

B2. Khởi tạo đối tượng MD5 và SHA256

```
BUF_SIZE = 65536
md5 = hashlib.md5()
sha256 = hashlib.sha256()
```

B3. Tiếp tục đọc tập tin lên và đưa vào hàm băm

```
with open(filename, "rb") as f:
    while True:
        data = f.read(BUF_SIZE)
        if not data:
            break
        md5.update(data)
        sha256.update(data
```

B4. In kết quả băm ra

```
print("MD5: {0}".format(md5.hexdigest()))
print("SHA256: {0}".format(sha256.hexdigest())
```

```
❯ python3 hash_file.py
MD5: c3917c08a7fe85db7203da6dcaa99a70
SHA256: cb580eb7dc55f9198e650f016645023e8b2224cf7d033857d12880b46c5c94ef
```

® **Bài tập (yêu cầu làm)**

1. *Sinh viên so sánh kết quả băm với VirusTotal và website Python.*

b) YARA

Là một ngôn ngữ lập tình dùng để phân loại mẫu. Một rule bao gồm tên và điều kiện.

```
rule my_rule_name { condition: false }
```

Rule trên sẽ không khớp với bất kỳ tập tin nào. Ngược lại rule sau thì có đối với mọi mẫu:

```
Rule my_rule_name { condition: true }
```

Rule sau sẽ khớp với tất cả tập tin trên 100KB

```
Rule over_100kb { condition: filesize > 100KB }
```

Một ví dụ kiểm tra tập tin có phải là PDF hay không? Ta dựa vào sequence `25 50 44 46` chuỗi byte ở đầu tập tin.

```
rule is_a_pdf {
    strings:
        $pdf_magic = {25 50 44 46}
    condition:
        $pdf_magic at 0
}
```

Cài đặt YARA tại đây: https://yara.readthedocs.io/en/stable/

B1. Tạo tập tin `rules.yara` và thêm rule sau:

```
rule is_a_pdf
{
    strings:
        $pdf_magic = {25 50 44 46}
    condition:
        $pdf_magic at 0
}
rule dummy_rule1
{
    condition:
        false
}
rule dummy_rule2
{
    condition:
        true
}
```

B2. Chọn tập tin PDF kiểm tra bằng lệnh

```
Yara rules.yara target_file
```

Kết quả như sau:

```
❯ Yara rules.yara Lab\ 2\ -\ Integrating\ Security\ and\ Automation.pdf
is_a_pdf Lab 2 - Integrating Security and Automation.pdf
dummy_rule2 Lab 2 - Integrating Security and Automation.pdf
```

c) Kiểm tra PE header

Tập tin Portable executable (PE) là một tập tin phổ biến của Windows. Tập tin PE chứa `.exe`, `.dll`, và `.sys`. Tất cả tập tin PE được phân biệt bằng PE header, là phần đầu của section của cấu trúc mã Windows, được parse ra mã subsequent.

Các trường ở PE header thường được sử dụng làm thuộc tính để phát hiện mã độc. Python hỗ trợ module `pefile` hỗ trợ trích xuất thuộc tính.

Cài đặt bằng lệnh:

```
pip install pefile
```

Tải tập tin mẫu cần thử nghiệm tại:

https://www.python.org/ftp/python/3.10.0/python-3.10.0-amd64.exe.

B1. Import thư viên pefile và thêm tập tin PE muốn parse

```
import pefile
desired_file = "python-3.10.0-amd64.exe"
pe = pefile.PE(desired_file)
```

B2. Liệt kê các import của tập tin PE

```
for entry in pe.DIRECTORY_ENTRY_IMPORT:
    print(entry.dll)
    for imp in entry.imports:
        print("\t", hex(imp.address), imp.name)
```

Một phần kết quả:

```
❯ python3 parse_pe.py
b'ADVAPI32.dll'
        0x44b000 b'RegCloseKey'
        0x44b004 b'RegOpenKeyExW'
        0x44b008 b'OpenProcessToken'
        0x44b00c b'AdjustTokenPrivileges'
        0x44b010 b'LookupPrivilegeValueW'
        0x44b014 b'InitiateSystemShutdownExW'
        0x44b018 b'GetUserNameW'
        0x44b01c b'RegQueryValueExW'
        0x44b020 b'RegDeleteValueW'
        0x44b024 b'CloseEventLog'
        0x44b028 b'OpenEventLogW'
        0x44b02c b'ReportEventW'
        0x44b030 b'ConvertStringSecurityDescriptorToSecurityDescriptorW'
        0x44b034 b'DecryptFileW'
        0x44b038 b'CreateWellKnownSid'
        0x44b03c b'InitializeAcl'
        0x44b040 b'SetEntriesInAclW'
        0x44b044 b'ChangeServiceConfigW'
        0x44b048 b'CloseServiceHandle'
```

B3. Liệt kê các section của tập tin PE

```
for section in pe.sections:
    print(
        section.Name,
        hex(section.VirtualAddress),
        hex(section.Misc_VirtualSize),
        section.SizeOfRawData,
    )
```

```
❯ python3 parse_pe.py
b'.text\x00\x00\x00' 0x1000 0x49937 301568
b'.rdata\x00\x00' 0x4b000 0x1ed60 126464
b'.data\x00\x00\x00' 0x6a000 0x1730 2560
b'.wixburn' 0x6c000 0x38 512
b'.rsrc\x00\x00\x00' 0x6d000 0x165fc 91648
b'.reloc\x00\x00' 0x84000 0x3dfc 15872
```

B4. In tất cả thông tin dump từ PE

```
print(pe.dump_info())
```

```
❯ python3 parse_pe.py | more
----------DOS_HEADER----------

[IMAGE_DOS_HEADER]
0x0         0x0   e_magic:                       0x5A4D
0x2         0x2   e_cblp:                        0x90
0x4         0x4   e_cp:                          0x3
0x6         0x6   e_crlc:                        0x0
0x8         0x8   e_cparhdr:                     0x4
0xA         0xA   e_minalloc:                    0x0
0xC         0xC   e_maxalloc:                    0xFFFF
0xE         0xE   e_ss:                          0x0
0x10        0x10  e_sp:                          0xB8
0x12        0x12  e_csum:                        0x0
0x14        0x14  e_ip:                          0x0
0x16        0x16  e_cs:                          0x0
0x18        0x18  e_lfarlc:                      0x40
0x1A        0x1A  e_ovno:                        0x0
0x1C        0x1C  e_res:
0x24        0x24  e_oemid:                       0x0
0x26        0x26  e_oeminfo:                     0x0
0x28        0x28  e_res2:
:...skipping...
----------DOS_HEADER----------

[IMAGE_DOS_HEADER]
0x0         0x0   e_magic:                       0x5A4D
0x2         0x2   e_cblp:                        0x90
0x4         0x4   e_cp:                          0x3
0x6         0x6   e_crlc:                        0x0
0x8         0x8   e_cparhdr:                     0x4
0xA         0xA   e_minalloc:                    0x0
0xC         0xC   e_maxalloc:                    0xFFFF
0xE         0xE   e_ss:                          0x0
0x10        0x10  e_sp:                          0xB8
0x12        0x12  e_csum:                        0x0
0x14        0x14  e_ip:                          0x0
0x16        0x16  e_cs:                          0x0
0x18        0x18  e_lfarlc:                      0x40
0x1A        0x1A  e_ovno:                        0x0
0x1C        0x1C  e_res:
0x24        0x24  e_oemid:                       0x0
0x26        0x26  e_oeminfo:                     0x0
0x28        0x28  e_res2:
0x3C        0x3C  e_lfanew:                      0x110

----------NT_HEADERS----------

[IMAGE_NT_HEADERS]
0x110       0x0   Signature:                     0x4550
```

d) Featurizing the PE header

Ta sẽ trích xuất thuộc tính từ PE header xây dựng bộ dữ liệu `malware/benign` để phân loại.

Trong phần này benign và malicious được cung cấp từ `PE SamplesDataset`. Giải nén thành hai thư mục `Benign PE Samples` và `Malicious PE Samples`

B1. Import pefile và hai thư viện

```
import pefile
from os import listdir
from os.path import isfile, join

directories = ["Benign PE Samples", "Malicious PE Samples"]
```

B2. Định nghĩa hai phương thức thu thập tên của sections và chuẩn hoá chúng.

```
def get_section_names(pe):
    """Gets a list of section names from a PE file."""
    list_of_section_names = []
    for sec in pe.sections:
        normalized_name = sec.Name.decode().replace("\x00","").lower()
        list_of_section_names.append(normalized_name)
    return list_of_section_names
```

B3. Ta định nghĩa một phương thuận tiện trong tiền xử lý import

```
def preprocess_imports(list_of_DLLs):
    """Normalize the naming of the imports of a PE file."""
    return [x.decode().split(".")[0].lower() for x in list_of_DLLs]
```

B4. Chúng ta định nghĩa hàm thu thập import từ tập tin

```
def get_imports(pe):
    """Get a list of the imports of a PE file."""
    list_of_imports = []
    for entry in pe.DIRECTORY_ENTRY_IMPORT:
        list_of_imports.append(entry.dll)
    return preprocess_imports(list_of_imports)
```

B5. Cuối cùng, duyệt quá tất cả tập tin và tạo danh sách thuộc tính

```
imports_corpus = []
num_sections = []
section_names = []
for dataset_path in directories:
    samples = [f for f in listdir(dataset_path) if
isfile(join(dataset_path, f))]
    for file in samples:
        file_path = dataset_path + "/" + file
        try:
```

B6. Ngoài việc thu thập thuộc tính, ta còn thu thập số lượng section của tập tin

```
        try:
            pe = pefile.PE(file_path)
            imports = get_imports(pe)
            n_sections = len(pe.sections)
            sec_names = get_section_names(pe)
            imports_corpus.append(imports)
            num_sections.append(n_sections)
            section_names.append(sec_names)
```

B7. Trong trường hợp không parse được tập tin PE, thêm try-catch

```
        except Exception as e:
            print(e)
            print("Unable to obtain imports from " + file_path)
```

® **Bài tập (yêu cầu làm)**

2. *Sinh viên cho biết quả của đoạn code trên*

2. Phân tích động mã độc

Không giống như phân tích tĩnh, phân tích động phân tích hành vi của mẫu đang được chạy. Lợi ích của phân tích động là ta có thể bypass obfuscation (qua mặt rối mã). Vì phần mềm độc hại thì không an toàn nên các mẫu sẽ chạy trên máy ảo (VM) hay còn được gọi là sandboxing.

® **Bài tập (yêu cầu làm)**

3. *Sinh viên tự tìm hiểu, cài đặt (https://cuckoo.sh/docs/introduction/index.html), thực hiện và trình bày phân tích động một tập tin PE.*

3. Sử dụng máy học phát hiện loại tập tin

Một kỹ thuật mà kẻ tấn công hay sử dụng là lén đưa các tập hại bằng cách ản đi phần mở rộng. Ví dụ một tập lệnh độc hại PowerShell (.ps1) , quản trị viên thiết lập hệ thống ngắn chặn thực thi các phần đuôi mở rộng ps1. Tuy nhiên, kẻ tấn công đã ẩn phần mở rộng, chỉ có kiểm tra nội dung tập tin mới biết được.

e) Scraping GitHub cho các loại tập tin đặc biệt

Để quản lý tập dữ liệu, ta sẽ trích xuất dữ liệu Github cho các tập tin cụ thể mà ta quan tâm.

Cài đặt thư viện:

```
pip install PyGitHub
```

B1. Import thư viện PyGitHub để gọi API của Github và sử dụng module base64 để mã hoá và giải mã tập tin.

```
import os
from github import Github
import base64
```

B2. Ta phải cung cấp thông tin chứng thực và đưa ra một truy vấn JavaScript trong repository

```
username = "your_github_username"
password = "your_token"
target_dir = "/path/to/JavascriptSamples/"
g = Github(username, password)
repositories = g.search_repositories(query="language:javascript")
n = 5
i = 0
```

B3. Duyệt danh sách repository trả về

```
for repo in repositories:
    reponame = repo.name
    target_dir_of_repo = target_dir + "\\" + reponame
    print(reponame)
    try:
```

B4. Tạo thư mục lưu trữ

```
        os.mkdir(target_dir_of_repo)
        i += 1
        contents = repo.get_contents("")
```

B5. Ta thêm tất cả các thư mục của repository vào hàng đợi để liệt kê tất cả các tập tin trong thư mục

```
        while len(contents) > 1:
            file_content = contents.pop(0)
            if file_content.type == "dir":
                contents.extend(repo.get_contents(file_content.path))
            else:
```

B6. Nếu kiểm tra một tập tin không phải là thư mục thì kiểm tra phần mở rộng có phải là `.js`

```
                st = str(file_content)
                filename = st.split('"')[1].split('"')[0]
                extension = filename.split(".")[-1]
                if extension == "js":
```

B7. Nếu là `.js` thì sẽ ghi ra tập tin

```
                    filecontents = repo.get_contents(file_content.path)
                    file_data = base64.b64decode(filecontents.content)
                    filename = filename.split("/")[-1]
                    file_out = open(target_dir_of_repo + "/" + filename, "wb")
                    file_out.write(file_data)
    except:
        pass
    if i == n:
        break
```

B8. Di chuyển tất cả tập tin Javascript vào cùng 1 thư mục

® Bài tập (yêu cầu làm)

4. Tương tự sinh viên hãy làm các câu truy vấn về Python và Powershell

f) Phân loại tập tin theo kiểu

Ta sẽ phân loại các tập tin JavaScript, Python, hoặc PowerShell từ phần trên đã thu thập.

B1. Thêm các thư viện cần thiết và đường dẫn tập dữ liệu để chia train và test

```
import os
from sklearn.feature_extraction.text import HashingVectorizer,
TfidfTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.pipeline import Pipeline

javascript_path = "JavascriptSamples/"
python_path = "PythonSamples/"
powershell_path = "PowerShellSamples/"
```

B2. Tiếp theo, đọc tất cả các kiểu tập tin. Tạo một mảng nhãn có giá trị -1, 0 , 1 đại diện JavaScript, Python và PowerShell

```
corpus = []
labels = []
file_types_and_labels = [(javascript_path, -1), (python_path, 0),
(powershell_path, 1)]
for files_path, label in file_types_and_labels:
    files = os.listdir(files_path)
    for file in files:
        file_path = files_path + "/" + file
        try:
            with open(file_path, "r") as myfile:
                data = myfile.read().replace("\n", "")
        except:
            pass
        data = str(data)
        corpus.append(data)
        labels.append(label)
```

B3. Tiếp theo chia bộ dữ liệu train-test, sau đó NLP cơ bản trên các tập tin và dung phân loại random forest

```
X_train, X_test, y_train, y_test = train_test_split(
```

```
    corpus, labels, test_size=0.33, random_state=11
)
text_clf = Pipeline(
    [
        ("vect",   HashingVectorizer(input="content",   ngram_range=(1,
3))),
        ("tfidf", TfidfTransformer(use_idf=True,)),
        ("rf", RandomForestClassifier(class_weight="balanced")),
    ]
)
```

B4. Cuối cùng là train và in ra kết quả phân loại

```
text_clf.fit(X_train, y_train)y_test_pred =
text_clf.predict(X_test)print(accuracy_score(y_test,
y_test_pred))print(confusion_matrix(y_test, y_test_pred)
```

® **Bài tập (yêu cầu làm)**

5. *Sinh viên cho biết quả của đoạn code trên*

g) Đo lường sự giống nhau giữa hai chuỗi

Để kiểm tra hai tập tin có sự giống nhau hay không ta sử dụng các hàm băm tiêu chuẩn như SHA256 hay MD5. Tuy nhiên, trường hợp khắt khe hơn là ta muốn biết 2 tập tin giống nhau đến mức độ nào ta dung một thuật toán băm ssdeep. Cài đặt thư viện:

```
pip3 install ssdeep
```

B1. Thêm thư viện ssdeep và tạo 3 mẫu string

```
import ssdeep
str1 = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua."
str2 = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore Magna aliqua."
str3 = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore aliqua."
str4 = "Something completely different from the other strings."
```

B2. Băm chuỗi

```
hash1 = ssdeep.hash(str1)
hash2 = ssdeep.hash(str2)
hash3 = ssdeep.hash(str3)
hash4 = ssdeep.hash(str4)

print(hash1)
print(hash2)
print(hash3)
print(hash4)
```

B3. Kết quả điểm số cho sự tương đồng giữa các chuỗi

```
print(ssdeep.compare(hash1, hash1))
print(ssdeep.compare(hash1, hash2))
print(ssdeep.compare(hash1, hash3))
print(ssdeep.compare(hash1, hash4))
```

h) Đo lường mức độ giống nhau giữa hai tập tin

Tiếp tục dùng ssdeep để so sánh hai tập tin binary. Tải tập tin mẫu cần thử nghiệm tại: https://www.python.org/ftp/python/3.10.0/python-3.10.0-amd64.exe.

B1. Đầu tiên tạo một bản sao từ tập tin `python-3.10.0-amd64.exe` thành `python-3.10.0-amd64-fake.exe` bằng cách thêm vài null bytes

```
truncate -s +1 python-3.10.0-amd64-fake.exe
```

B2. Dùng hexdump để xem sự khác nhau giữa hai tập tin trước và sau

```
┌──(kali㉿kali)-[~]
└─$ hexdump -C python-3.10.0-amd64.exe | tail -5
01b010e0  10 9c 34 66 02 d3 51 8c  b1 64 19 f3 55 12 0e 74  |..4f..Q..d..U..t|
01b010f0  38 71 4c 2e 1c db 44 d4  f3 81 31 a5 9c 2e c6 06  |8qL...D...1.....|
01b01100  4f 33 c6 8a 9a 5e 16 52  8c 4b 55 10 2b cd 45 61  |O3...^.R.KU.+.Ea|
01b01110  a5 00 00 00 00 00 00 00                           |........|
01b01118

┌──(kali㉿kali)-[~]
└─$ hexdump -C python-3.10.0-amd64-fake.exe | tail -5
01b010e0  10 9c 34 66 02 d3 51 8c  b1 64 19 f3 55 12 0e 74  |..4f..Q..d..U..t|
01b010f0  38 71 4c 2e 1c db 44 d4  f3 81 31 a5 9c 2e c6 06  |8qL...D...1.....|
01b01100  4f 33 c6 8a 9a 5e 16 52  8c 4b 55 10 2b cd 45 61  |O3...^.R.KU.+.Ea|
01b01110  a5 00 00 00 00 00 00 00  00                      |.........|
01b01119
```

B3. Dùng ssdeep để so sánh 2 tập tin

```
import ssdeep
hash1 = ssdeep.hash_from_file("python-3.10.0-amd64.exe")
hash2 = ssdeep.hash_from_file("python-3.10.0-amd64-fake.exe")
ssdeep.compare(hash1, hash2)
```

® **Bài tập (yêu cầu làm)**

6. *Sinh viên cho biết quả của đoạn code trên*

i) Trích xuất N-grams

Trong phân tích văn bản, N-grams là chuỗi N tokens (từ hoặc ký tự). Ví dụ đoạn văn bản sau: *The quick brown fox jumped over the lazy dog,* nếu token là từ thì 1-grams là *the, quick, brown, fox, jumped, over, the, lazy*, 2-grams là *the quick, quick brown, brown fox,…* 3-grams là *the quickbrown, quick brown fox, brown fox jumped,…* Giống như một thống kê cho phép xây dựng chuỗi Markov để thống kê dự đoán và tạo văn bản từ kho dữ liệu. N-gram cho phép lập mô hình các thuộc tính thống kê của kho ngữ liệu, từ đó đự đonas mẫu là độc hại hay lành tính. Cài đặt thư viện sau:

```
pip3 install nltk
```

Tải tập tin mẫu cần thử nghiệm tại:
https://www.python.org/ftp/python/3.10.0/python-3.10.0-amd64.exe.

B1. Import thư viện `collections` để đếm và `ngrams` từ nltk để trích xuất N-grams

```
import collections
from nltk import ngrams
```

B2. Chọn tập tin phân tích

```
file_to_analyze = “python-3.10.0-amd64.exe”
```

B3. Định nghĩa hàm đọc tập tin từ bytes

```
def read_file(file_path):
    """Reads in the binary sequence of a binary file."""
    with open(file_path, "rb") as binary_file:
        data = binary_file.read()
    return data
```

B4. Viết một hàm lấy một chuỗi bytes thành N-grams

```
def byte_sequence_to_Ngrams(byte_sequence, N):
    """Creates a list of N-grams from a byte sequence."""
    Ngrams = ngrams(byte_sequence, N)
    return list(Ngrams
```

B5. Viết một hàm đọc một tập tin và lấy được N-grams của nó

```
def binary_file_to_Ngram_counts(file, N):
    """Takes a binary file and outputs the N-grams counts of its binary
sequence."""
    filebyte_sequence = read_file(file)
    file_Ngrams = byte_sequence_to_Ngrams(filebyte_sequence, N)
    return collections.Counter(file_Ngrams
```

B6. Định nghĩa với N-4 thu được số lượng 4-grams

```
extracted_Ngrams = binary_file_to_Ngram_counts(file_to_analyze, 4)
```

B7. Hiển thị 10 kết quả đầu 4-grams phổ biến trong tập tin

```
print(extracted_Ngrams.most_common(10))
```

j) Chọn N-grams tốt nhất

Số lượng thuộc tính là rất lớn cho nên phải chọn N phù hợp. Chẳng hạn N=3, thì có 256x256x256=16,777,216 N-grams. Sử dụng bộ dữ liệu Benign PE Samples và Malicious PE Samples. Để tìm số N sao cho có nhiều thông tin nhất.

B1. Chọn thư viện, xác đinh N và định nghĩa đường dẫn.

```
from os import listdir
from os.path import isfile, join
directories = ["Benign PE Samples", "Malicious PE Samples"]
N = 2
```

B2. Đếm tất cả N-grams trong tập tin

```
Ngram_counts_all_files = collections.Counter([])
for dataset_path in directories:
    all_samples = [f for f in listdir(dataset_path) if
isfile(join(dataset_path, f))]
    for sample in all_samples:
```

```
        file_path = join(dataset_path, sample)
        Ngram_counts_all_files +=
binary_file_to_Ngram_counts(file_path, N)
```

B3. Ta sẽ thêm với K=1000 với N-grams thường gặp

```
K1 = 1000
K1_most_frequent_Ngrams = Ngram_counts_all_files.most_common(K1)
K1_most_frequent_Ngrams_list = [x[0] for x in K1_most_frequent_Ngrams]
```

B4. Phương thức `featurize_sample` sẽ sử dụng mẫu và xuất số lần xuất hiện của của N-grams phổ biến trong chuỗi bytes của nó.

```
def featurize_sample(sample, K1_most_frequent_Ngrams_list):
    """Takes a sample and produces a feature vector.
    The features are the counts of the K1 N-grams we've selected.
    """
    K1 = len(K1_most_frequent_Ngrams_list)
    feature_vector = K1 * [0]
    file_Ngrams = binary_file_to_Ngram_counts(sample, N)
    for i in range(K1):
        feature_vector[i] =
file_Ngrams[K1_most_frequent_Ngrams_list[i]]
    return feature_vector
```

B5. Dùng hàm `featurize_sample` duyệt qua tất cả tập tin và gán nhãn cho chúng.

```
directories_with_labels = [("Benign PE Samples", 0), ("Malicious PE
Samples", 1)]
X = []
y = []
for dataset_path, label in directories_with_labels:
    all_samples = [f for f in listdir(dataset_path) if
isfile(join(dataset_path, f))]
    for sample in all_samples:
        file_path = join(dataset_path, sample)
        X.append(featurize_sample(file_path,
K1_most_frequent_Ngrams_list))
        y.append(label)
```

B6. Import thư viện để chọn thuộc tính và số lượng thuộc tính muốn lấy

```
from sklearn.feature_selection import SelectKBest, mutual_info_classif,
chi2

K2 = 10
```

B7. Chọn 3 kiểu để lựa chọn thuộc tính N-grams

- **Frequency:** Chọn N-grams phổ biến

```
X_top_K2_freq = X[:, :K2]
```

- **Mutual information:** Chọn N-grams có xếp hạng cao theo thuật toán mutual information

```
mi_selector = SelectKBest(mutual_info_classif, k=K2)
X_top_K2_mi = mi_selector.fit_transform(X, y)
```

- **Chi-squared:** Chọn N-grams có xếp hạng cao theo thuật toán chi squared

```
chi2_selector = SelectKBest(chi2, k=K2) X_top_K2_ch2 =
chi2_selector.fit_transform(X, y)
```

® **Bài tập (yêu cầu làm)**

7. *Sinh viên cho biết quả của đoạn code trên*

4. Xây dựng trình phát hiện phần mềm độc hại bằng phân tích tĩnh

Trình phân tích này sử dụng cả 2 bộ thuộc tính trích xuất từ PE header và N-grams. Sử dụng tập dữ liệu Benign PE Samples và Malicious PE Sample.

B1. Tạo list các mẫu và gán nhãn cho chúng.

B2. Chia dữ liệu train-test

B3. Các hàm lấy thuộc tính

B4. Chọn 100 thuộc tính phổ biến với 2-grams

B5. Trích xuất số lượng N-grams count, section names, imports và số lượng sections của mỗi mẫu trong train-test.

B6. Sử dụng hàm băm `tfidf` để chuyển imports, section names từ văn bản thành dạng số

B7. Kết hợp các vector thuộc tính thành 1 mảng.

B8. Ta huấn luyên bằng phân loại Random Forest cho tập train

B9. Thu thập các thuộc tính của tập test, giống như tập huấn luyện

B10. Ta chuyển đổi vector từ thuộc tính test, và kiểm tra kết quả của trình phân loại.

® Bài tập (yêu cầu làm)

8. *Sinh viên hoàn thành các bước trên*

C. YÊU CẦU & ĐÁNH GIÁ

1. Yêu cầu

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.. Đăng ký nhóm cố định từ buổi 1.
- Sinh viên báo cáo kết quả thực hiện và nộp bài bằng **1 trong 2 hình thức**:

k) Hình thức 1 - Báo cáo chi tiết:

Báo cáo cụ thể quá trình thực hành (có ảnh minh họa các bước) và giải thích các vấn đề kèm theo. Trình bày trong file PDF theo mẫu có sẵn tại website môn học.

l) Hình thức 2 - Video trình bày chi tiết:

Quay lại quá trình thực hiện Lab của sinh viên kèm thuyết minh trực tiếp mô tả và giải thích quá trình thực hành. Upload lên **Youtube** và chèn link vào đầu báo cáo theo mẫu. **Lưu ý:** *Không chia sẻ ở chế độ Public trên Youtube.*

Đặt tên file báo cáo theo định dạng như mẫu:

[Mã lớp]-LabX_MSSV1-Tên SV1_MSSV2 –Tên SV2

Ví dụ: *[NT101.I11.1]-Lab1_14520000-Viet_14520999-Nam.*

- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Nộp báo cáo trên theo thời gian đã thống nhất tại website môn học.

2. Đánh giá:

- Sinh viên hiểu và tự thực hiện được bài thực hành, đóng góp tích cực tại lớp.
- Báo cáo trình bày chi tiết, giải thích các bước thực hiện và chứng minh được do nhóm sinh viên thực hiện.
- Hoàn tất nội dung cơ bản và có thực hiện nội dung *mở rộng – cộng điểm* (với lớp **ANTN**).

Kết quả thực hành cũng được đánh giá bằng kiểm tra kết quả trực tiếp tại lớp vào cuối buổi thực hành hoặc vào buổi thực hành thứ 2.

Lưu ý: Bài sao chép, nộp trễ, "*gánh team*", ... sẽ được xử lý tùy mức độ.

HẾT

Chúc các bạn hoàn thành tốt!