컴퓨터보안 Assignment #2. Malware Classification

2018008177 김찬위

1. 컴파일환경

- linux: ubuntu 18.04.4 기준 정상작동 확인
- python 3.7.7 로 작성

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
import os
```

- TfidfVectorizer 등의 모듈을 import 할 수 있어야 함
- 실행파일은 main.py `$ python main.py` 명령어로 실행가능

2. 기본 알고리즘 및 입력 주의사항

> - 정적 분석 결과로 얻을 수 있는 중 하나인 opcode sequence를 활용하여 sequence가 주어졌을 때 해당 sequence가 악성코드인지 정상파일인지 분류하는 모델 구현

➔ 악성 코드와 정상 파일의 opcode sequence를 가지고 TF-IDF 기법을 활용해 빈도수를 측정

➔ 판별해야 할 대상 sequence와 데이터로 모은 sequence들의 유사도를 cosine similarity 기법을 이용해 확인하여 악성 코드와 정상 파일 중 유사도가 더 높은 쪽으로 판별

- TF-IDF 기법

: 단어의 빈도와 역 문서 빈도를 사용해 각 단어들마다 중요한 정도를 가중치로 주는 방법. TfidfVectorizer 모듈을 사용하면 직접 구현하지 않아도 함수로 바로 원하는 값을 얻을 수 있음

$$\mathbf{tf}(t,d) = \frac{f_d(t)}{\max_{w \in d} f_d(w)} \qquad \mathbf{idf}(t,D) = \ln\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$$

$$\mathbf{tfidf}(t,d,D) = \mathbf{tf}(t,d) \cdot \mathbf{idf}(t,D)$$

- cosine similarity

: 두 벡터 간의 코사인 각도를 이용해 구할 수 있는 두 벡터의 유사도. 1에 가까울수록 유사도가 높다고 판단할 수 있음

$$similarity = cos(\Theta) = \frac{A \cdot B}{||A||\,||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

[입력 주의사항]

- 각각의 폴더에는 opcode sequence가 나열되어 있는 파일이 존재하며 아래 설명에 맞는 폴더에 파일을 넣어 프로그램을 작동시킬 수 있음

- 현재 예시로 주어진 opcode.zip의 파일들이 폴더에 각각 100개씩 존재하며, seq 폴더에는 다른 폴더에 넣지 않은 새로운 예시 존재

폴더 명	역할
0	판별을 위한 데이터가 될 정상 파일
1	판별을 위한 데이터가 될 악성 코드 파일
seq	판별하고자 하는 대상 파일

3. 코드 구현

```
opcodes = ['mov', 'push', 'call', 'pop', 'cmp', 'jz', 'lea', 'test', 'jmp', 'add', 'jnz', 'retn',
 'xor', 'and', 'bt', 'fdivp', 'fild', 'fstcw', 'imul', 'int', 'nop', 'pushf', 'rdtsc', 'sbb', 'se
tb', 'setle', 'shld', 'std', '(bad)']
```

- 가장 공통적으로 가지고 있으며 구분할 기준이 될 opcode 총 29개를 선정해서 이것만 비교

```
def main():
    global opcodes

    normal = sample(0)
    malware = sample(1)

    dirname = os.getcwd() + '/seq'
    allfiles = os.listdir(dirname)
    filelist = [filename for filename in allfiles]

    cnt_norm = 0
    cnt_mal = 0
    n = len(filelist)
#    n = 10
    for i in range(0, n):

        tfidf = test(filelist[i])

        norm = max_similarity(normal, tfidf)
        mal = max_similarity(malware, tfidf)

        if norm > mal:
            cnt_norm += 1
        else:
            cnt_mal += 1

    print('normal code: ' + str(cnt_norm))
    print('malware: ' + str(cnt_mal))
```

[main 함수]

1. 판별 데이터가 될 파일들의 TF-IDF를 구함

2. 판별하고자 하는 파일이 있는 seq 폴더에 접근해 파일 하나하나를 모든 데이터와 유사도를 비교하여 판별하고 그 수를 카운트

[sample 함수]

```
def sample(num):
    global opcodes, norm, mal

    dirname = os.getcwd() + '/' + str(num)
    allfiles = os.listdir(dirname)
    filelist = [filename for filename in allfiles]

    n = len(filelist)
#   n = 10
    data = []
    for i in range(0, n):
        fname = dirname + '/' + filelist[i]
        f = open(fname, 'r')
        lines = f.readlines()
        seq = ''
        for op in lines:
            seq += op
        data.append(seq)
        f.close()

    data = np.array(data)
    t = TfidfVectorizer().fit(data)
    ops = t.get_feature_names()
    t = t.transform(data)
    t = t.toarray()
```

```
    B = []
    for i in range(0, len(opcodes)):
        if opcodes[i] in ops:
            idx = ops.index(opcodes[i])
            tmp = []
            for j in range(0, n):
                tmp.append(t[j][idx])
            B.append(tmp)
        else:
            tmp = []
            for j in range(0, n):
                tmp.append(0)
            B.append(tmp)

    t = np.array(B)

    return t
```

1. 판별 데이터가 될 파일들의 opcode를 읽어 TfidfVectorizer 모듈을 활용하여 TF-IDF를 구함

2. 혹시 파일에 opcode가 읽히지 않아 없다면 0으로 채워 넣음

3. 선정한 opcode만 남을 수 있도록 선별하여 TF-IDF를 배열 형태로 만들어 반환

[test 함수]

```
def test(filename):
    fname = os.getcwd() + '/seq/' + filename

    if os.stat(fname).st_size ==  0:
        return np.array([0]*29).reshape(-1,)

    f = open(fname, 'r')
    lines = f.readlines()
    seq = ''
    for op in lines:
        seq += op
    f.close()

    seq = np.array([seq])
    tfidf = TfidfVectorizer().fit(seq.ravel())
    ops = tfidf.get_feature_names()
    tfidf = tfidf.transform(seq)
    tfidf = tfidf.toarray()

    A = []
    for i in range(0, len(opcodes)):
        if opcodes[i] in ops:
            idx = ops.index(opcodes[i])
            A.append(tfidf[0][idx])
        else:
            A.append(0)

    tfidf = np.array(A).reshape(-1,)
    return tfidf
```

1. 판별하고자 하는 파일이 있는 seq 폴더에 접근

2. 만일 opcode가 읽히지 않아 비어 있다면 0으로 채워 넣음

3. TfidfVectorizer 모듈을 활용해 TF-IDF를 구하고 선정한 opcode만 남도록 선별한 뒤 배열 형태로 반환

[cos_similarity 함수]

1. 판별하고자 하는 sequence와 데이터 sequence의 유사도를 검사

```
def cos_similarity(v1, v2):

    dot_product = np.dot(v1, v2)
    l2_norm = (np.sqrt(sum(np.square(v1))) * np.sqrt(sum(np.square(v2))))
    similarity = dot_product / l2_norm

    return similarity
```

[max_similarity 함수]

```
def max_similarity(num, v1):

    maximum = 0
    for i in range(0, num.shape[1]):
        v2 = np.array(num[:,i]).reshape(-1,)
        res = cos_similarity(v1, v2)

        if maximum < res:
            maximum = res

    return maximum
```

1. 인자로 받은 악성코드나 정상 파일의 TF-IDF 배열과 판별하고자 하는 sequence의 TF-IDF 배열의 유사도 검사

2. 가장 큰 유사도를 반환

4. 실행 화면

➔ 다소 효율적이지 못한 코드 탓에 각 폴더에 파일을 100개씩 넣은 상황에서 조금 느

```
main.py:55: RuntimeWarning: invalid value encountered in double_scalars
  similarity = dot_product / l2_norm
```

리게 동작

➔ cosine similarity 모듈을 이용하지 않고 직접 수식을 구현했기 때문에 runtime warning이 발생하고 다소 느리지만 정상 작동

- 100개 이상의 데이터를 학습하고 판별할 경우 시간이 오래 걸리고 컴퓨터가 멈추는 경우가 발생하여 최대 100개 까지만 넣어서 확인

0 (정상 파일)	1 (악성 코드)	Seq (대상 파일)	실행 결과
100	100	정상 파일 100	normal code: 70 malware: 30
100 위와 다른 파일로 선정	100	정상 파일 100	normal code: 75 malware: 25
100	100	악성 코드 100	normal code: 9 malware: 91
100	100 위와 다른 파일로 선정	악성 코드 100	normal code: 16 malware: 84

➔ 데이터로 넣어준 파일에 따라 같은 대상 sequence에서도 다른 결과가 도출. 데이터의 개수가 많아진다면 더 높은 정확도를 보일 것으로 예상. 유사도가 동일하게 나올 경

우에 악성 코드로 판별하도록 구현하여 악성 코드로 인식되는 경우가 더 많이 발생

➔ 100을 기준으로 정상 파일은 약 70~75% 정도의 정확도로 찾을 수 있고, 악성 코드의 경우 약 85~90%로 비교적 높은 정확도로 찾을 수 있음