Assignment #2
opcode sequence analysis

using TF-IDF, cosine similarity

2018007956 김채아

[opcode analysis 방법] – opcode frequency 사용
TF-IDF, cosine similarity로 계산

악성코드 분류를 위한
opcode 중요도 순위

Rank	Opcode	Rank	Opcode
1	SMSW	21	ROL
2	BSR	22	AAS
3	CMPSB	23	IMUL
4	SHLD	24	POPAD
5	AND	25	LEAVE
6	SUB	26	OUTSD
7	NOT	27	REPNE
8	NEG	28	LOOPE
9	TEST	29	SETLE
10	MOVZX	30	MOV
11	DIV	31	LTR
12	MOVSB	32	XCHG
13	INC	33	JLE
14	SCASW	34	PUSH
15	LEA	35	SBB
16	CLC	36	CMP
17	JNE	37	DEC
18	AAD	38	CMPSD
19	CALL	39	SETNL
20	JO	40	OUTSW

악성코드 분류에 큰 영향력을 가지는 연산부호이다
상위 몇 개의 연산부호를 선택하고
cosine similarity를 계산하는
방법을 사용했다

malware의 opcode frequency

NO.	Opcode	Frequency
1	AAA	0.62033
2	AAD	0.126886
3	AAM	0.098689
4	AAS	0.028197
5	ADC	2.49542
6	ADD	31.806
7	AND	2.56591
8	ARPL	0.733117
9	BOUND	0.775412
10	BSF	0
11	BSR	0
12	BSWAP	0.014098
13	BT	0
14	BTC	0
15	BTR	0
16	BTS	0
17	CALL	0.394755
18	CBW	0
19	CDQ	0.098689
20	CLC	0.155082

연산부호 빈도수 :
프로그램에서 사용된
전체 연산부호 개수 대비
특정 연산부호의 개수

malware opcode의 tf-idf 가 나온 자료를 샘플로 사용한다
각 테스트 파일들의 빈도수 tf-idf가 위 샘플과 유의미한 차이를 보이면
정상파일로, 차이가 작으면(유사도가 높으면) 악성파일로 분류한다
(머신 러닝을 사용하지 않고, 미리 머신러닝이 되어 나와있는 값을 이용해서 분류한다
tf-idf, cosine similarity를 구해보는 것에 의의를 둠)

〈악성코드 분류를 위한 중요 연산부호 선택 및 그 유용성에 관한 연구〉 논문을 참고하였습니다

python code

```
path = "C:\\Users\\LG\\Desktop\\sicurity~20\\opcode\\1\\*.txt"
files = glob.glob(path)

corpus =[]
# 파일을 통째로 str로 저장
for file in files:
    with open(file,"r") as f:
        doc = []
        for word in f:
            doc.append(word.rstrip('\n'))
        corpus.append(str(' '.join(doc)))
# print(corpus)
vocab = list(set(w for x in corpus for w in x.split()))
vocab.sort()

N = len(corpus)# 총 문서의 수

def tf(t, d):
    return d.count(t)

def idf(t):
    df = 0
    for doc in corpus:
        df += t in doc
    return log(N/(df + 1))

def tfidf(t, d):
    return tf(t,d)* idf(t)


result = []
for i in range(N):
    result.append([])
    d = corpus[i]
    for j in range(len(vocab)):
        t = vocab[j]

        result[-1].append(tfidf(t,d))

result = np.array(result)
tfidf_ = pd.DataFrame(result, columns = vocab)
print(tfidf_)
```

txt파일을 하나씩 불러와서
하나의 string으로 만들어준다
corpus = ['txt1', 'txt2', ··· , 'txtN']
이런 형태를 가진다

corpus를 단어별로 잘라서 하나의 리스트로 표현한다
정렬은 굳이 필요X

tf-idf를 연산하는 함수

받은 파일의 tf-idf를 계산한다

```
analysis=[]
malware = [0, 2.56591, 0.155082, 0.126886, 0.39755]
for n in range(len(corpus)):
    opcode_list = np.split(result[n],len(result[n]))

    re_opcode_list =[]
    need_only = ['bsr', 'and', 'clc', 'aad', 'call']
    for opcode in need_only:
        if opcode in vocab:
            idx = vocab.index(opcode)
            re_opcode_list.append(float(opcode_list[idx]))
        else:
            re_opcode_list.append(0)

    # cosine similarity
    cosine_sim = 1 - spatial.distance.cosine(re_opcode_list, malware)

    # print(cosine_sim)
    analysis.append(cosine_sim)

# 정확도 분석
notmal = 0
malware = 0
for i in range(len(analysis)):
   if analysis[i] > 0:
      malware += 1
   else:
      notmal += 1

print('정상파일 개수:',notmal)
print('악성파일 개수:',malware)
```

아래 need_only 리스트에 있는 opcode순으로
측정된 tf-idf값을 저장한다

악성코드 분류에 영향력이 큰
opcode를 몇 개 추려낸다

코사인 유사도
= 1-코사인 거리

모든 파일을 코사인 유사도를 계산해서
analysis 리스트에 넣는다

몇 번의 테스트를 통해
정상 파일은 유사도가 음수로,
악성 파일은 유사도가 양수로 나온다는 것을 알았다

이 방법으로 파일이 악성 파일인지 정상 파일인지 판단한다
각각의 분류된 개수를 측정하여 정확도를 알아본다

[TF-IDF]

```
path = "C:\\Users\\LG\\Desktop\\sicurity~20\\opcode\\1\\*.txt"
files = glob.glob(path)

corpus =[]
# 파일을 통째로 str로 저장
for file in files:
    with open(file,"r") as f:
        doc = []
        for word in f:
            doc.append(word.rstrip('\n'))
        corpus.append(str(' '.join(doc)))
# print(corpus)
vocab = list(set(w for x in corpus for w in x.split()))
vocab.sort()

N = len(corpus)# 총 문서의 수

def tf(t, d):
    return d.count(t)

def idf(t):
    df = 0
    for doc in corpus:
        df += t in doc
    return log(N/(df + 1))

def tfidf(t, d):
    return tf(t,d)* idf(t)


result = []
for i in range(N):
    result.append([])
    d = corpus[i]
    for j in range(len(vocab)):
        t = vocab[j]

        result[-1].append(tfidf(t,d))

result = np.array(result)
tfidf_ = pd.DataFrame(result, columns = vocab)
print(tfidf_)
```

TF : Term Frequency 단어 빈도수
tf(t, d) : d에 t가 나오는 빈도수

DF : 문서의 빈도 값으로, 특정 단어가 여러 데이터에 자주 등장하는지를 알려주는 것이다.

IDF : Inversed Document Frequency 역문서 빈도. DF의 역수
특정 단어가 다른 데이터에 등장하지 않을수록 값이 커진다

$$idf(t,D) = \log \frac{|D|}{|\{d \in D : t \in d\}| + 1}$$

log ((t가 들어있는 문서의 개수)/(전체 문서의 개수+1))

TF-IDF : TF*IDF로, 어떤 단어가 해당문서에 자주 등장하지만 다른 문서에는 많이 없는 단어일수록 높은 값을 가지게 된다.
정보 검색과 텍스트 마이닝에서 이용하는 가중치로, 여러 문서로 이루어진 문서군이 있을 때 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타내는 통계적 수치이다
문서의 핵심어를 추출하거나, 검색 엔진에서 검색 결과의 순위를 결정하거나,
문서의 사이의 비슷한 정도를 구하는 등의 용도로 사용할 수 있다

입력된 파일의 TF-IDF를 계산하고 일부 malware opcode frequency의 TF-IDF와 cosine similarity값을 계산한다
유사도 값에 따라 malware/not_malware로 classification한다

[cosine similarity]

```
analysis=[]
malware = [0, 2.56591, 0.155082, 0.126886, 0.39755]
for n in range(len(corpus)):
    opcode_list = np.split(result[n],len(result[n]))

    re_opcode_list =[]
    need_only = ['bsr', 'and', 'clc', 'aad', 'call']
    for opcode in need_only:
        if opcode in vocab:
            idx = vocab.index(opcode)
            re_opcode_list.append(float(opcode_list[idx]))
        else:
            re_opcode_list.append(0)

    # cosine similarity
    cosine_sim = 1 - spatial.distance.cosine(re_opcode_list, malware)

    analysis.append(cosine_sim)
```

cosine similarity : 두 벡터의 사이각을 코사인으로 구하는 것
두 벡터의 유사도를 구한다

cosine similarity = 1 − cosine distance

각 문서에서 추출된
특정 opcode(악성 코드 분석에 유리한 opcode 지정)의 비율과
opcode malware sample과의 cosine similarity를 구해서
그 값을 보고 malware인지 아닌지 판단한다

[분석 결과] 정적 분석 과정에서 정상적으로 opcode가 추출되지 않은 빈 파일들은 제거 후 분석했다
코드 동작 시간이 너무 길어서 임의의 악성파일/정상파일 100개씩을 테스트해보았다

– 악성파일 100개 분석

```
정상파일 개수: 2
악성파일 개수: 98
```

분류 정확도 : 98%

– 정상파일 100개 분석

```
정상파일 개수: 93
악성파일 개수: 7
```

분류 정확도 : 93%

머신 러닝을 하지 않았지만 머신 러닝으로 잘 분석된 값을 이용하니 높은 정확도를 보였다

	COORD	DEVMODEA	DEVMODEW	DLGTEMPLATE	...	xlat	xor	xorpd	xorps
0	0.0	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000
1	0.0	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000
2	0.0	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000
3	0.0	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000
4	0.0	0.000000	0.000000	0.000000	...	1.768148	0.0	0.000000	0.000000
..	...	...	...	...	...	...	...	...	...
95	0.0	3.852348	2.430561	3.852348	...	0.000000	0.0	0.000000	0.000000
96	0.0	0.000000	0.000000	0.000000	...	2.231155	0.0	6.676243	5.069758
97	0.0	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000
98	0.0	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000
99	0.0	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000

(각 악성 파일에 나오는 opcode의 TF–IDF 결과)

	BLENDFUNCTION	NUMBERFMTW	_OSVERSIONINFOA	...	xor	xorpd	xorps
0	0.0	0.0	0.0	...	-0.054451	0.00000	0.000000
1	0.0	0.0	0.0	...	-0.057081	0.00000	0.000000
2	0.0	0.0	0.0	...	-0.062228	0.00000	0.000000
3	0.0	0.0	0.0	...	-0.073504	0.00000	0.000000
4	0.0	0.0	0.0	...	-0.033843	0.00000	0.000000
..	...	...	...	...	...	...	...
95	0.0	0.0	0.0	...	-0.081062	0.00000	0.000000
96	0.0	0.0	0.0	...	-0.065684	0.00000	0.000000
97	0.0	0.0	0.0	...	-0.058841	0.00000	0.000000
98	0.0	0.0	0.0	...	-0.047554	0.00000	0.000000
99	0.0	0.0	0.0	...	-0.081868	4.64376	6.389938

(각 정상 파일에 나오는 opcode의 TF–IDF 결과)