Assignment #2. Malware Classification

컴퓨터소프트웨어학부 2018008722 유수영

1. 사용한 프로그래밍 언어 : 파이썬(3.7.0 버전)

2. 컴파일 환경 및 방법 : Windows에서 코드 작성 및 실행

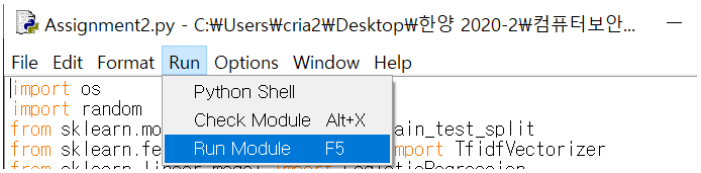
   1) Window CMD창에서 실행

```
C:₩Users₩cria2>cd C:₩Users₩cria2₩Desktop₩한양 2020-2₩컴퓨터보안

C:₩Users₩cria2₩Desktop₩한양 2020-2₩컴퓨터보안>python Assignment2.py
```

   ➔ cmd창에서 코드가 있는 path로 이동한 다음, python [코드파일 이름].py로 컴파일 및 실행

   2) python shell에서 실행



   ➔ idle 코드 파일에서 F5(Run module)을 통해 컴파일 및 바로 실행

3. 사용 module : numpy, os, glob, random, sklearn

```
C:₩Users₩cria2>cd C:₩Users₩cria2₩AppData₩Roaming₩Microsoft₩Windows₩Start Menu₩Programs₩Python 3.7

C:₩Users₩cria2₩AppData₩Roaming₩Microsoft₩Windows₩Start Menu₩Programs₩Python 3.7>pip install scikit-learn
Collecting scikit-learn
  Downloading scikit_learn-0.23.2-cp37-cp37m-win_amd64.whl (6.8 MB)
     |████████████████████████████████| 6.8 MB 6.8 MB/s
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-2.1.0-py3-none-any.whl (12 kB)
Requirement already satisfied: numpy>=1.13.3 in c:₩users₩cria2₩appdata₩local₩programs₩python₩python37₩lib₩site-packages (from sc
ikit-learn) (1.18.2)
Collecting scipy>=0.19.1
  Downloading scipy-1.5.4-cp37-cp37m-win_amd64.whl (31.2 MB)
     |████████████████████████████████| 31.2 MB 726 kB/s
Collecting joblib>=0.11
  Downloading joblib-0.17.0-py3-none-any.whl (301 kB)
     |████████████████████████████████| 301 kB 3.3 MB/s
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-0.17.0 scikit-learn-0.23.2 scipy-1.5.4 threadpoolctl-2.1.0
```

   ➔ sklearn 사용을 위해 scikit-learn을 cmd창에서 설치

   1) os : file과 경로의 존재 여부를 판단하기 위해 import

   2) random : 사용자가 입력한 수 만큼의 파일을 training에 이용하기 위해 경로에 있는 파일들 중 랜덤하게 사용자가 입력한 수 만큼의 파일을 선택하기 위해 import

   3) sklearn : data를 training할 것과 정확도 계산을 위해 test할 것으로 나누기 위한 train_test_split이나, tfidf 계산을 위한 것, classify를 위한 logistic regression을 위하여 import

```
import os
import random
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
```

4. 사용 분석 결과와 사용 기법

   1) 정적 분석 : 정적 분석 결과로 얻을 수 있는 정보 중 하나인 opcode의 sequence를 이용하였다.

   2) Tf – idf : opcode sequence에 나오는 단어들의 빈도와 문서군 전체에서 나오는 빈도 등을 따져서 benign code과 malware code를 분류하는 tf-idf 기법을 사용하였다.

   3) logistic regression : 해당 인공지능 분류 모델을 사용하여 opcode sequence들을 분류할 수 있도록 하였다.

5. 전체적인 코드 설명 : 사용자로부터 benign code들이 있는 폴더의 경로와 malware code들이 있는 폴더의 경로를 입력 받고 해당 경로로 가서 그 폴더에 있는 파일들에 담긴 code들을 이용하여 benign과 malware를 구분하는 모델을 만들게 된다. 이 때, 사용자로부터 폴더 안에 있는 몇 개의 파일을 사용할 것인지(폴더 안의 몇 개의 코드를 모델을 만드는 데에 사용할 것인지) 개수를 입력 받고, 그 개수만큼 랜덤하게 파일 리스트를 뽑아내어 해당 리스트에 있는 파일의 내용만을 이용하여 모델을 만든다. (예를 들어 사용자가 5를 입력했다면 사용자가 입력한 경로에 있는 파일들 중 5개만을 랜덤하게 뽑아 모델을 만들 때 사용한다) 그리고 랜덤하게 뽑힌 파일들을 다시 training할 것들과 정확도를 계산하기 위한 test할 것들로 나눈다. training을 하기 위한 benign 파일들에 등장하는 opcode들과 malware 파일들에 등장하는 opcode들의 빈도수와 역문서 빈도를 파이썬 내의 모듈과 함수를 이용하여 계산하여 각 파일마다 tf-idf의 값을 만들고, 이를 이용하여 logistic regression으로 test 파일들의 opcode들이 benign인지 malware인지 분류하여 실제 test파일들의 malware 여부와 일치하는지를 확인하여 정확도를 계산할 수 있도록 하는 모델이 구현되어 있다. 즉, training을 위해 뽑힌 파일들의 내용을 가지고 tf-idf 값을 계산하고, 각 benign과 malware opcode들의 tf-idf 값의 특징을 이용하여 logistic regression으로 test 파일들을 분류하고 분류한 결과가 맞는지 정도로 정확도를 계산한다. 그리고, 사용자로부터 분류하고자 하는 opcode sequence나 그 sequence들이 적혀 있는 파일의 이름과 경로를 입력 받고, 그 opcode sequence의 tf-idf값을 계산하여 아까 training시켰던 값들로 fit 시킨 logistic regression으로 malware 여부를 판단하여 결과를 출력해준다.

6. 예외 처리

   1) 사용자가 입력된 경로가 잘못된 경우(os.path.exists 사용)

   2) 사용자가 입력한 파일의 이름이 잘못되어서 해당 경로에 없거나 파일이 아닌 디렉토리인 경우

   3) 사용자로부터 입력받은 모델을 만들기 위해 사용할 파일의 수가 실제로 존재하는 파일의 수보다 많을 경우

➢ 모두 경고 문구를 출력한 뒤 프로그램을 바로 종료한다.

7. 프로그램 실행 흐름

   1) 사용자에게 benign opcode들이 있는 경로와 malware opcode들이 있는 경로를 입력받는다.

   2) 해당 경로에 있는 파일들의 이름을 모두 담은 리스트를 만들고, 그 중 몇 개의 파일을 모델 형성에 사용할 것인지 사용자에게 값을 받아 그 값만큼 랜덤하게 파일 이름을 추출하여 새로운 리스트를 만든다.

   3) 리스트에 있는 각 파일들을 training할 것과 test할 것으로 나눈다.

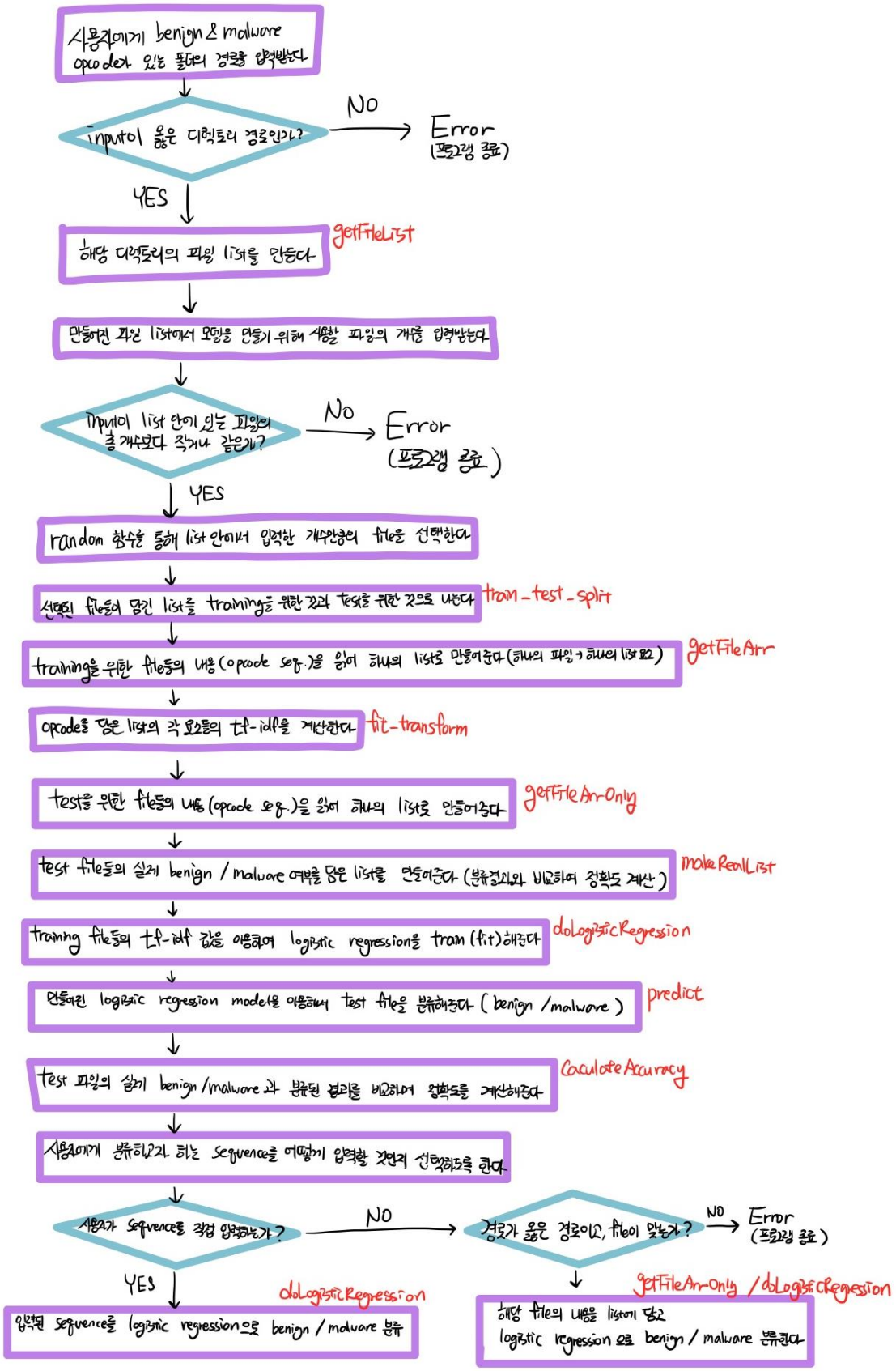
   4) Training을 위한 파일들의 내용을 읽고, 각 파일별로 tf-idf 값을 계산한다.

   5) 계산한 tf-idf을 이용하여 logistic regression 모델을 만들고, test 파일들의 내용의 tf-idf 값을 바탕으로 해당 모델을 이용하여 benign인지 malware인지 분류한다.

   6) test파일들의 분류값이 실제와 맞는지를 확인하고, 그 정확도를 계산한다.

   7) 사용자에게 분류하고 싶은 sequence나 그 sequence가 담긴 파일의 이름과 경로를 입력 받는다.

   8) 앞에서 계산했던 training data들의 tf-idf값을 이용하여 5번에서 진행했던 것과 동일하게 logistic regression을 이용하여 분류하여 7번에서 입력받은 sequence가 benign인지 malware인지를 알려준다.


사용자에게 benign & malware opcode가 있는 폴더의 경로를 입력받는다
input이 옳은 디렉토리 경로인가?
No
Error (프로그램 종료)
YES
해당 디렉토리의 파일 list를 만든다
getFileList
만들어진 파일 list에서 모델을 만들기 위해 사용할 파일의 개수를 입력받는다
input이 list 안에 있는 파일의 총 개수보다 작거나 같은가?
No
Error (프로그램 종료)
YES
random 함수를 통해 list 안에서 입력한 개수만큼의 file을 선택한다
선택된 file들이 담긴 list를 training을 위한 것과 test를 위한 것으로 나눈다
train_test_split
training을 위한 file들의 내용(opcode 종류)을 읽어 하나의 list로 만들어준다(하나의 파일 → 하나의 list 요소)
getFileArr
opcode를 담은 list의 각 요소들의 tf-idf를 계산한다
fit_transform
test를 위한 file들의 내용(opcode 종류)을 읽어 하나의 list로 만들어준다
getFileArr-Only
test file들의 실제 benign / malware 여부를 담은 list를 만들어준다 (분류결과와 비교하여 정확도 계산)
makeRealList
training file들의 tf-idf 값을 이용하여 logistic regression을 train (fit)해준다
doLogisticRegression
만들어진 logistic regression model을 이용해서 test file을 분류해준다 (benign / malware)
predict
test 파일의 실제 benign / malware과 분류된 결과를 비교하여 정확도를 계산해준다
CaculateAccuracy
사용자에게 분류하고자 하는 sequence를 어떻게 입력할 것인지 선택하도록 한다
사용자가 sequence를 직접 입력하는가?
NO
경로가 옳은 경로이고, file이 있는가?
NO
Error (프로그램 종료)
YES
doLogisticRegression
입력된 sequence를 logistic regression으로 benign / malware 분류
getFileArr-Only / doLogisticRegression
해당 file의 내용을 list에 담고
logistic regression으로 benign / malware 분류한다

8. 함수 설명

1) getFileLIst(path)

```
def getFileList(path) :
    if os.path.exists(path) :
3       fileList = os.listdir(path)
        return fileList
    else :
2       return False
```

1- 사용자로부터 입력 받은 benign이나 malware opcode들이 있는 폴더의 경로로 가서 해당 폴더에 있는 모든 파일들의 이름을 list의 형태로 만들어 반환해주는 함수이다.

2- 인자로 받는 path가 사용자가 입력한 경로이며, path가 존재하지 않는 path라면 false를 반환하여 main함수에서 프로그램을 끝낼 수 있도록 한다.

3- 존재하는 path라면 fileList라는 변수에 해당 path에 있는 모든 파일의 list를 담고 return한다.

2) getFielArr(benignFileList, malwareFileList, benignPath, malwarePath)

```
def getFileArr(benignFileList, malwareFileList, benignPath, malwarePath) :  2
    fileArray = []
    for file in benignFileList :                                              3
        benignFile = benignPath + "/" + file
        f = open(benignFile, mode = 'rt')
        temp = ''
        line = f.readline()
        while line != '' :
            line = line.rstrip('₩n')
            temp = temp + line + ' '
            line= f.readline()
        fileArray.append(temp)
    f.close()

    for file in malwareFileList :                                             4
        malwareFile = malwarePath + "/" + file
        f = open(malwareFile, mode = 'rt')
        temp = ''
        line = f.readline()
        while line != '' :
            line = line.rstrip('₩n')
            temp = temp + line + ' '
            line= f.readline()
        fileArray.append(temp)

    f.close()
    return fileArray                                                          5
```

1- benign 파일들의 내용과 malware 파일들의 내용을 합쳐서 하나의 list로 return해주는 함수이다. 하나의 파일이 list의 하나의 요소가 되고, 각 요소들은 파일 안의 opcode 내용들을 이어 붙여 하나의 string으로 만든 것이다.

2- 인자로 받는 benignFileList에는 benign opcode들 파일의 이름의 list가,

malwareFileList에는 malware opcode들 파일의 이름의 list가, benignPath에는 benign 코드 파일이 있는 경로가, malwarePath에는 malware 코드 파일이 있는 경로가 담겨져 있다.

3- 빈 list fileArray 변수를 만들고, 우선 benign파일의 내용부터 filaArray에 담게 된다. 인자로 받은 beanignPath와 각 파일의 이름을 통해 파일을 open하고, 한 줄씩 읽고 모든 내용을 하나의 string으로 만들어 이를 fileArray에 추가해준다. 즉, 하나의 파일을 읽고 나면 해당 파일에 써져 있는 opcode들이 하나의 string으로 합쳐져서 list에 하나의 string이 추가되는 것이다.(하나의 파일 당 하나의 string 추가) 이를 위해 각 파일이 끝날 때까지 while문을 돌며 temp라는 임의의 변수에 읽은 단어들을 합쳐주고, readline을 통해 다음 단어를 읽는다.(예를 들어 temp가 'push add sub'인 상태에서 'pop'을 읽었다면 temp에 추가하여 temp는 'push add sub pop'이 되고, pop이 해당 파일의 마지막 단어였다면 fileArray에 'push add sub pop'이 추가되는 것이다)

4- 3번과 같은 방식으로 모든 benign 파일들의 내용을 fileArray라는 list에 담아주고, 뒤이어 malware 파일의 내용들도 동일하게 담아준다

5- 파일을 사용했다면 파일을 닫고, (close함수 이용) 모든 파일의 내용이 담아져 있는 list인 fileArray 를 return한다.

3) getFileArrOnly(fileList, path)

```
def getFileArrOnly(fileList, path) :
    fileArray = []
    for file in fileList :
        filePath = path + "/" + file
        f = open(filePath, mode = 'rt')
        temp = ''
        line = f.readline()
        while line != '' :
            line = line.rstrip('₩n')
            temp = temp + line + ' '
            line= f.readline()
        fileArray.append(temp)
    return fileArray
```

2

3

1- getFileArr함수와 거의 동일한 함수이다. 그러나 getFileArr함수가 두 개의 경로와 두 개의 파일 리스트들을 인자로 받아 benign과 malware 파일의 내용을 하나의 list에 담아준 것이라면, 이 함수는 하나의 경로와 하나의 파일 리스트만을 인자로 받아 해당 경로의, 파일 리스트에 들어있는 파일들에 대하여 각 파일들의 내용을 읽어 list에 넣어준 다음, list를 return해주는 함수이다.

2- getFileArr함수와 동작은 동일하다. 빈 list인 fileArray를 만들어주고, fileList에 있는 모든 파일들에 대해서 파일의 끝이 올 때까지 while문을 돌며 file의 내용을 하나

의 string으로 만들어준다. 하나의 파일을 다 읽고 나면 만들어진 string(변수 temp, 읽은 파일의 모든 opcode들이 써져있다)을 list에 추가해준다.

3- fileList에 있는 모든 파일을 읽고 나면 fileArray를 return해준다.

4) doTrainTestSplit(fileList)

```
def doTrainTestSplit(fileList) :
    fileTrain, fileTest = train_test_split(fileList)
    return fileTrain, fileTest
```

2

1- file list에 들어있는 파일들 중 무엇을 training을 위해 사용할 것이고 무엇을 test에 사용할 것인지를 랜덤하게 나누어 주는 함수이다. 최종적으로 training을 위한 파일들과 test를 위한 file들을 list의 형태로 각각 return해준다.(return되는 값이 두 개)

2- 파이썬에 정의되어 있는 train_test_split함수를 사용해주고, 모든 설정은 default 그대로 사용한다.

5) doLogisticRegression(Tfidf, pred, tfidf)

```
def doLogisticRegression(Tfidf, pred, tfidf) :
    label = []
    for i in range (0,int(Tfidf.shape[0] / 2)) :
        label.append("benign")
    for i in range (0,int(Tfidf.shape[0] / 2)) :
        label.append("malware")
    model = LogisticRegression()
    model.fit(Tfidf, label)
    predictTfidf = tfidf.transform(pred)
    return model.predict(predictTfidf)
```

2

3

1- logistic regression을 통해서 입력 받은 sequence가 malware인지 아닌지를 분류하는 함수이다. 인자로 받는 Tfidf는 앞서 만들어진 training을 위한 benign 파일과 malware 파일들의 tf-idf 값을 계산하여 나온 matrix형태의 값으로, logistic regression을 fit하기 위해 사용된다. Pred는 우리가 판단해야할 sequence이며, tfidf는 pred의 tfidf 계산을 위해 필요한 tf-idf vector이다.

2- 우선, Tfidf 값에 맞는 label list를 만들어주어야 한다.(Tfidf의 어느 값이 benign opcode의 tf-idf값이고 malware opcode의 tf-idf 값인지를 지정해주어야 하므로) Logistic regression을 fit하기 위해서 필요하며, Tfidf는 절반은 benign opcode들의 tf-idf값으로, 나머지 뒤 쪽 절반은 malware opcode들의 tf-idf 값으로 구성 되어있기 때문에 label은 Tfidf의 크기와 동일한 크기로 만들어지며, 앞쪽 절반은 "benign"으로, 뒤 쪽 절반은 "malware"의 값을 가지게 된다.

3- 만들어진 label과 Tfidf를 이용하여 logistic regression model을 만들어주고, pred의

tf-idf값을 tfidf 벡터를 이용하여 계산해준다. 그리고 계산한 값을 보고 logistic regression을 통해 benign에 가까운 sequence인지, malware에 가까운 sequence인지를 return해준다.

➔ Benign opcode들의 tf-idf 값과 malware opcode들의 tf-idf 값을 가지고 benign opcode와 malware opcode의 tf-idf의 값의 각 특성을 보고 logistic regression을 이용하여 새로 받은 opcode sequence의 tf-idf는 benign opcode의 tf-idf 값에 가까운지, malware opcode의 tf-idf 값에 가까운지를 판단하여 opcode sequence가 benign인지 malware인지를 판단하게 되는 것이다.

6) makeRealList(TBFL, TMFL)

```
def makeRealList(TBFL, TMFL) :
    real = []
3   for i in range (0, len(TBFL)) :
        real.append("benign")
    for i in range (0, len(TMFL)) :
        real.append("malware")
    return real
```

1- test를 위한 파일들이 각각 실제로 benign인지 malware인지를 나타내 주는 list를 만들어주는 함수이다. 정확도 계산을 위해 분류된 값과 실제 값을 비교해야 할 때 사용된다.

2- TBFL과 TMFL은 각각 test benign file list와 test malware file list로 각각 test를 위한 benign 파일들의 내용을 담고 있는 list와 malware 파일들의 내용을 담고 있는 list이다. (getFielArrOnly 함수를 거쳐 나온 결과물) 각 list의 한 요소는 하나의 파일에 써져 있는 모든 opcode들의 string이다.

3- 나중에 TBFL과 TMFL을 합쳐서 합쳐진 list를 logistic regression을 통해 잘 분류되는지 확인하기 때문에 각 파일들이 실제로 benign인지 malware인지를 알려주는 list는 먼저 TBFL의 크기만큼 "benign"을 list에 추가해주고, 그 뒤로 TMFL의 크기만큼 "malware"를 list에 추가해주면 된다.

7) calculateAccuracy(real, predict)

```
def calculateAccuracy(real, predict) :
    matchNum = 0
    benignMalware = 0
    malwareBenign = 0
    benign = 0
    malware = 0
3   for i in range (0, len(real)) :
        if real[i] == "benign" :
            benign = benign + 1
        elif real[i] == "malware" :
            malware = malware + 1

        if real[i] == predict[i] :
            matchNum = matchNum + 1
        elif real[i] == "benign" and predict[i] == "malware" :
            benignMalware = benignMalware + 1
        elif real[i] == "malware" and predict[i] == "benign" :
            malwareBenign = malwareBenign + 1
2   return matchNum / len(real), benign, malware, benignMalware, malwareBenign
```

1- 만들어진 model이 얼마나 잘 training되었는지를 test하고 난 뒤, 그 정확도를 계산하는 함수이다. Real은 실제 파일들의 benign / malware 여부가 들어있고, predict에는 모델을 통해 분류된 값들이 들어있어 이들을 차례로 비교해보면 된다.

2- 실제 값들과 일치하는 정도와, benign의 개수, malware의 개수, benign을 malware로 잘못 분류하는 개수, malware를 benign으로 잘못 분류하는 개수를 return해준다.

3- Real과 predict를 하나씩 보면서 둘의 값이 일치하는지, real의 값이 benign인데 predict의 값이 malware이고, real의 값이 malware인데 predict의 값이 benign인지 등을 for 문을 돌며 확인한다.

8) Main()

1- 가장 먼저, tf-idf 값을 계산할 때 사용하기 위한 벡터를 만들어주고 tfidf 변수에 넣어준다.

2- 사용자로부터 benign opcode와 malware opcode가 있는 경로를 입력 받기 위해 안내 문구를 출력해주고 benignPath와 malwarePath에 각각 경로를 저장한다.

저장된 path들을 이용하여 path에 있는 모든 파일들의 list를 만들기 위해 getFileList 함수를 사용하고 각 list를 benignFileList와 malwareFileList 변수에 저장한다.

만약 각 benignFileList와 malwareFileList의 값이 false라면 경로가 존재하지 않는 것이므로 그대로 프로그램을 종료한다.

3- 사용자에게 모델을 training하기 위해 사용하기 위한 파일의 개수를 입력 받는다. 입력 받은 개수가 list안에 있는 file의 총 개수보다 많으면 프로그램을 종료한다.

4- 사용자가 입력한 수 만큼 benignFileList와 malwareFileList에서 랜덤하게 선택하여 list를 update해준다.

5- doTrainTestSplit함수를 이용하여 benignFileList와 malwareFileList를 train할 파일들과 test할 파일들로 나누어 주고 각각 benignFileTrain, benignFileTest, malwareFileTrain, malwareFileTest에 넣어준다.

6- Train을 위한 file들과 파일이 있는 경로를 인자로 넘겨주어 getFileArr함수를 이용하여 FL 변수에 각 파일들의 내용이 하나의 요소(string)으로 담겨있는 list를 넣어준다.

7- FL을 이용하여 각 요소들의 tf-idf값을 계산해서 Tfidf 변수에 넣어준다.

8- Training이 끝나고 나면 test를 위해서 benignFileTest와 malwareFileTest를 이용해서 test를 위한 파일들의 내용이 하나의 요소로 담겨있는 list를 만들고 둘을 합쳐준다(getFileArrOnly 함수 사용)

9- 정확도 계산을 위해 real이라는 변수에 makeRealList함수를 이용하여 각 파일들의 실체 정체(benign / malware)를 담아주고, predicAc에는 doLogisticRegression함수를 이용하여 각 test 파일들의 예측값을 담아준다.

그 후 calcualteAccuracy 함수를 이용하여 정확도를 계산하고 화면에 출력해준다.

10- 그 다음으로 사용자에게 분류를 하고 싶은 sequence나 그 sequence가 있는 파일의 경로를 입력 받을 수 있도록 한다(사용자가 둘 중 하나를 선택할 수 있음)

11- Sequence를 직접 입력 받으면 바로 그 sequence를 doLogisticRegression을 통해서 분류하고 나온 결과값을 화면에 출력해준다.

12- 사용자가 경로 입력을 선택한다면 sequence가 있는 파일의 이름과 그 경로를 입력받고, getFileArrOnly 함수를 통해 해당 파일의 내용을 하나의 string으로 만들고 그 값을 이용하여 doLogisticRegression 함수를 통해 분류할 수 있도록 한다. 그리고 그 값을 화면에 출력해준다. 이 때, 사용자가 입력한 경로가 파일이 아니거나 해당 경로가 존재하지 않는다면 경고 문구 출력 후 프로그램을 바로 종료한다.

9. 실행 화면

1) 사용자가 분류하고자 하는 sequence가 적혀 있는 파일의 경로를 입력하였고, 그 opcode sequence가 실제로 benign일 경우

```
========== RESTART: C:₩Users₩cria2₩Desktop₩한양 2020-2₩컴퓨터보안₩A2.py =======
====
Please enter the path where benign opcode is located and the path where malware
opcode is located to training
Benign opcode's path : C:₩Users₩cria2₩Desktop₩한양 2020-2₩컴퓨터보안₩opcode₩0
Malware opcode's path : C:₩Users₩cria2₩Desktop₩한양 2020-2₩컴퓨터보안₩opcode₩1
Plese enter the number of files to use within the each folder
15
......TRAINING......

......TESTING......

A total of 8 test codes were tested
This model's accuracy is 62.5 %
1 out of 4 benign codes were wrongly classified as malware
2 out of 4 malware codes were wrongly classified as benign

You can choose one of these,
1. Enter the sequence to check directly
2. Enter the path of the file that contains the sequence
Please enter the number
2
Please enter the file's name and the path that contains the sequence to classify
file's name : 0aa01e13c6db0c6e95eecaeb46bc85b6.vir.txt
file's path : C:₩Users₩cria2₩Desktop₩한양 2020-2₩컴퓨터보안₩opcode₩1
This sequence is benign code
>>>
```

➔ Benign 파일과 malware 파일을 각각 15개씩 사용하여 총 30개를 사용한 model을 만들었을 때, 30개 중 8개가 test 파일로 선정이 되었고, 62.5%의 정확도를 가지게 된다.(정확도는 매 실행마다 달라지는데, 사용자가 입력한 수에 따라 랜덤으로 파일이 선정되고 train_test_split 함수에서도 랜덤으로 train파일과 test파일을 나누기 때문이다) 실제 4개의 benign 파일 중 하나가 malware로 잘못 분류되었고, 4개의 malware 파일 중 2개가 benign으로 잘못 분류되었다.

➔ 사용자가 입력한 경로의 파일(실제로 benign인 파일)을 읽어 tf-idf 값을 계산하여 logistic regression으로 분류한 결과 benign 파일로 분류되었다.

2) 사용자가 분류하고자 하는 sequence를 직접 입력하였고, 해당 sequence가 malware의 opcode 중 일부분일 경우

```
...
========== RESTART: C:₩Users₩cria2₩Desktop₩한양 2020-2₩컴퓨터보안₩A2.py =======
====
Please enter the path where benign opcode is located and the path where malware
opcode is located to training
Benign opcode's path : C:₩Users₩cria2₩Desktop₩한양 2020-2₩컴퓨터보안₩opcode₩0
Malware opcode's path : C:₩Users₩cria2₩Desktop₩한양 2020-2₩컴퓨터보안₩opcode₩1
Plese enter the number of files to use within the each folder
10
......TRAINING......

......TESTING......

A total of 6 test codes were tested
This model's accuracy is 66.66666666666666 %
1 out of 3 benign codes were wrongly classified as malware
1 out of 3 malware codes were wrongly classified as benign

You can choose one of these,
1. Enter the sequence to check directly
2. Enter the path of the file that contains the sequence
Please enter the number
1
Please enter the sequence that you want to classify
jmp jmp jmp jmp jmp jmp jpm retn jmp pop push add sub cmp cmp
This sequence is malware code
>>>
```

➔ Benign 파일과 malware 파일을 각각 10개씩 사용하여 총 20개를 사용한 model 을 만들었을 때, 20개 중 6개가 test 파일로 선정이 되었고, 66.66%의 정확도를 가지게 된다. 실제 3개의 benign 파일 중 하나가 malware로 잘못 분류되었고, 3개의 malware 파일 중 1개가 benign으로 잘못 분류되었다.

➔ 사용자가 입력한 opcode sequenced를 읽고 tf-idf 값을 계산하여 logistic regression으로 분류한 결과 malware 파일로 분류되었다.

3) cmd창에서 실행