Assignment #2. Malway Classification Report

임문경 2018068622

서론

분석데이터 : API sequence

사용언어 : python3

실행환경 : jupyter notebook

사용모듈 : numpy / os

모델을 설명하기 앞서 이번 과제에서 분석한 데이터는 동적 분석 결과로 얻을 수 있는 정보 중 하나인 API sequence를 이용하여 악성코드 분석을 실시하였으며 구현을 위해서 사용한 언어는 python3이고 jupyter notebook을 통해서 실행하였습니다.

또한 사용된 모듈은 numpy와 os로 numpy는 데이터의 행렬 및 벡터 처리를 위해서, os는 폴더내의 API파일을 불러오기 위해서 사용되었습니다. 그리고 주어진 API sequence를 행렬로 변환한 후 별도의 파일로 저장하여서 해당 코드실행시 데이터 처리 및 변환과정을 생략할 수 있도록 하였습니다. 해당 데이터파일은 "dataset.npz"와 "opSet.npz"로 해당 보고서와 같은 장소에 저장되어 있습니다.

이 보고서에서 사용된 모델은 "장준영 김우재 옥정윤 임을규, 랜섬웨어 Native API 정보 분석을 통한 탐지 방법 연구" 논문을 참고하여 작성하였으며 해당 논문에서 사용된 방법 중 TF-IDF와 코사인 유사도를 이용한 빈도수 중심 분석을 이용하였습니다. 그리고 이때 사용된 TF,IDF값은 위 논문의 실험 결과 중 랜섬웨어가 아닌 경우의 탐지율이 가장 높게 나온 수식인 수식(4)와 수식(5) 그리고 검사하려는 파일의 IDF값을 1로 설정한 수식을 이용하였습니다.

또한 구현 코드에서 읽은 모든 file및 폴더는 같은 폴더내에 존재한다는 가정하에 작성되었습니다.

구현설명

1. (실행 X : 전처리) 전체 API scanning

현재경로에 있는 0과 1폴더(API sequence폴더)의 파일리스트를 가져와 파일에 접근하여 API목록을 "total_API"변수에 저장한다. 그리고 중복되는 API 목록을 제거하기 위해서 "total_API"변수를 set으로 변환한 뒤 다시 list변수로 변환하여 중복을 제거한 후 모든 API목록을 입력받은 후 정렬하여 저장합니다. 또한 전체 API의 갯수를 "num_of_total_API"변수에 저장하여 후에 TF-IDF값을 계산할 때 사용합니다.

2. (실행 X : 전처리) train data와 test data 분류

전체 데이터셋에서 train data와 test data를 분류하고, 각 sample에 API의 갯수를 저장합니다.

train data와 test data는 전체 데이터에서 70%, 30%의 비율로 나눴습니다.

각각 train data중 악성코드가 아닌 것, 악성코드인 것, test data중 악성코드가 아닌 것, 악성코드인 것 총 4가지로 분류하여 train_data0, train_data1, test_data0, test_data1 변수에 각각 매핑하였습니다.

3. 전체 API 목록 및 개수, train data와 test data 불러오기

같은 폴더내에 저장된 "dataset.npz", "opSet.npz"를 불러와 각 변수에 저장

4. train data에서 TF 및 TF_IDF값 구하기

train data에 대해서 TF값을 구하기 위해서 각 sample의 API목록을 scanning하면서 TF수식을 적용했습니다. 이 때 TF값은 log(1 + w/d),수식(4),를 사용하였고, IDF값은 상수값 1,수식(5),를 사용하였습니다.

sample들의 API마다 TF_IDF값을 구한뒤, 각각의 API들의 TF_IDF의 평균값을 구하여 "train0AVG_TF_IDF"와 "train1AVG_TF_IDF" 변수에 저장하였습니다.

해당 값과 구하고자하는 데이터의 코사인 유사도를 비교하여 악성코드 분류를 시행하게 될 것입니다.

5. train data에 대한 악성코드 분류

test data에 대해 분류를 하기 전 코사인 유사도를 통해 train data의 악성코드 분류를 시행하였습니다.

앞 단계에서 구한 "train0AVG_TF_IDF"와 "train1AVG_TF_IDF" 값을 통해 구하고자 하는 값의 코사인 유사도를 각각 구한 뒤 유사도가 더 높은 쪽으로 해당 데이터를 분류하였습니다. 그리고 유사도 공식에서 분모가 0이 되는 것을 막기 위해서 아주 작은 'alpha'값을 분모에 더하였습니다.

또한 training data가 이미 올바르게 분류된 상태이므로 정확도를 계산하는 방법은 악성코드가 아닌 train data(train0TF_IDF)의 경우

"train0AVG_TF_IDF"와 더 유사하다고 분류된 데이터의 개수 / 악성코드가 아닌 train data의 전체 개수"로 쉽게 구할 수 있습니다.

위 분류의 결과로 악성코드가 아닌 데이터를 올바르게 분류할 확률은 80.62%로, 악성코드인 데이터를 올바르게 분류할 확률은 53.69%로 산출되었습니다.

6. test data에서 TF_IDF값 구하기

4번 단계에서와 마찬가지로 test data에 대해서도 TF_IDF값을 구하여 각각 "test0TF_IDF"와 "test1TF_IDF" 변수에 저장하였습니다.

7. test data에 대한 악성코드 분류

5번 단계와 마찬가지로 코사인 유사도를 통하여 test data의 악성코드 분류를 시행하였습니다.

앞 단계와는 다르게 결과값을 각각 "test0_y", "test1_y"에 저장하였습니다.

그 후 분류의 결과를 출력하였으며, 악성코드가 아닌 데이터를 올바르게 분류할 확률은 83.30%, 악성코드인 데이터를 올바르게 분류할 확률은 54.17%로 산출되었음을 확인할 수 있었습니다.