

정보보호와시스템보안 AI-Based Malware Detection Project

팀원

- 20191608 서형빈
- 20191600 박찬혁
- 20182249 최용렬

Datasets + Optimization

- PEMINER, EMBER와 다르게 PESTUDIO는 학습데이터 기준으로 20,000개 중 560개의 데이터가 부족한 것을 확인하였습니다. 검증데이터와 테스트데이터 또한 비슷한 양상을 보입니다. 이 때 부족한 파일에 대해서 결측치라고 정의하였습니다.
- python에서 multiprocessing을 지원하는 모듈인 ray를 사용하여 Feature Extraction Method를 병렬화하였습니다. 이에 소요 시간 측면에서 성능이 약 3배 정도 향상되었습니다. (511초 -> 168초)
- json 파일을 읽는 시간을 단축시키기 위해 Python보다 속도가 훨씬 빠른 C언어 기반으로 json reading을 수행하는 ujson 모듈을 사용하였습니다.

Feature Engineering

- FeatureHashing은 0으로 초기화된 벡터에, 모듈러 연산 기반으로 해당하는 인덱스에 1씩 숫자를 더해나가는 함수를 구현하여 수행하였습니다. 해시함수로 md5를 사용하였습니다.
- PEMINER는 제시된 예시 코드에서 사용한 방법과 같이 모든 피처를 그대로 사용하여 벡터화 하였습니다.
- EMBER에 대해서는 다음과 같은 방식으로 피처를 추출했습니다.
 - COFF: timestamp를 제외한 characteristics, machine을 피처로 사용하여 64차원으로 FeatureHashing을 진행하였습니다.
 - OPTIONAL: dll_characteristics, subsystem, magic을 피처로 사용하여 64차원으로 FeatureHashing하였고, major_image_version, major_image_version, minor_image_version... 등 실수형 데이터로 저장되어 있는 피처는 그대로 사용하였습니다.

- IMPORTS: DLL name(key)은 512차원으로, 각 key에 들어있는 여러 function name(value)은 8192차원으로 FeatureHashing하였습니다. 이때 function name이 같으나 DLL name이 다를 수 있기 때문에 value를 FeatureHashing 할 때 key인 DLL name을 더해서 진행하였습니다. 또한, function name 자체에도 의미가 있을 수 있기에 value만 가지고 1024차원으로 FeatureHashing을 다시 수행하였습니다.
- EXPORTS: 모든 데이터를 512차원으로 FeatureHashing 하였습니다.
- DATADIRECTORY: 피처에서 등장할 수 있는 모든 name마다 존재여부를 판단하였습니다. 해당 name이 없는 경우 결측치로 판단하여 size와 virtual_address를 모두 0으로 설정해 피처로 사용하였고, 존재하면 size와 virtual_address 값을 그대로 피처로 사용하였습니다.
- GENERAL, STRING: 제시된 예시 코드의 방법과 동일합니다.
- BYTEENTROPY, HISTOGRAM: 전체 벡터에 합계를 나누는 정규화 과정을 거쳐 피처 벡터로써 추가하였습니다.
- SECTIONS: size가 0인 section의 개수, name이 존재하지 않는 section의 개수, mem_read, mem_execute 권한이 존재하는 section의 개수, mem_write 권한이 존재하는 section의 개수를 피처로써 벡터에 추가하였습니다. 교안에서는 size, entropy, vsize를 FeatureHashing 한다고 하였으나, 3개의 피처는 수의 크기가 의미 있는 정보인데 이를 FeatureHashing 하게 되면 의미가 사라질 것이라고 판단하였습니다. 따라서 FeatureHashing 적용 대신에 각 피처의 평균, 최댓값, 최솟값을 구하여 피처에 추가하였습니다. 마지막으로 section내에 존재하는 모든 props는 1024차원으로 FeatureHashing 하였고, entry section name 또한 64차원으로 FeatureHashing 하여 벡터에 추가하였습니다.
- PESTUDIO에 존재하는 다양한 중요 피처는 PEMINER나 EMBER에서 등장하였기 때문에 많은 피처를 다시 추가할 필요는 없다고 판단하였습니다. 그 대신, string 데이터에서 많은 양의 문자열 텍스트가 그대로 들어있는 것을 확인하였고 공격자의 패턴 등 다양한 정보를 얻을 수 있을 것이라고 판단하여 벡터에 추가하였습니다.
 - ascii 문자열이 등장한 횟수와 unicode 문자열이 등장한 횟수, 전체 문자열이 등장한 횟수를 각각 피처에 넣어주었습니다. 특정 문자열의 출현 빈도 또한 중요한 정보가 될 수 있을 것이라고 판단하였습니다.
 - ascii 문자열과 unicode 문자열의 텍스트를 모두 합쳐서 8192차원 피처해싱을 진행해 벡터에 추가하였습니다.
- Standard, MinMax 등의 정규화 기법은 적용하지 않았습니다. 위 과정을 거쳐 피처 추출을 진행하면 실제로 이상치가 등장하는데, 정규화는 이상치를 충분히 필터링 한 다음에 성능이 좋아진다는 특징이 있어 쉽게 사용하지는 못하였습니다. 그 대신, 정규화를

사용하지 않아도 올바르게 학습할 수 있는 결정 트리 타입의 머신러닝 모델을 사용하여 문제를 완화하였습니다.

Model Train & Test

- EDA 파트에서 언급했던 PESTUDIO 데이터 결측치 문제 때문에 모든 피처를 있는 그대로 사용할 수 없습니다. 이에 PEMINER+EMBER로 학습한 모델 A와, 3가지의 피처 추출기를 모두 사용한 모델 B를 각각 만들어 학습 및 검증을 수행하였습니다.
- 모델 A는 전체 학습 데이터 20,000개로 학습하여 결측치인 데이터 202개로 검증을 수행합니다.
- 모델 B는 결측치가 제거된 학습 데이터 19,440개로 학습하여 결측치가 제거된 검증 데이터 9,798개로 검증을 수행합니다.
- 모델은 모두 LightGBM을 사용했습니다. 최적의 HyperParameter를 구하기 위해 `param = {'n_iter': [128, 256, 384, 512], 'num_leaves': [7, 15, 31, 63, 127], 'learning_rate': [0.1, 0.07, 0.05, 0.03, 0.01]}` 으로 설정하여 총 100개의 경우의 수에 대해 그리드 서치를 진행하였습니다.
- 모든 모델을 저장하고 앙상블 기법을 적용하기 위해 생성된 100개의 모델 중 가장 검증 성능이 좋았던 상위 9개의 모델을 가져와서 간단한 hard voting을 적용하였습니다.
- modelA에서 가장 좋은 모델의 정확도가 96.53%이고, 앙상블을 적용 했을 때의 정확도가 96.03%입니다. 가장 좋은 모델의 검증 정확도가 높기 때문에 테스트 데이터 예측을 수행할 때 가장 좋은 단일 모델 하나만 사용하였습니다.
- modelB에서 가장 좋은 모델의 정확도가 96.30%이고, 앙상블을 적용 했을 때의 정확도가 96.33%입니다. 앙상블 모델의 검증 정확도가 높기 때문에 테스트 데이터 예측을 수행할 때 앙상블 모델을 사용하였습니다.
- modelA와 modelB의 검증 정확도를 종합해보면 96.34%로 계산됩니다.

```
modelA best model inference
Accuracy) 0.9653465346534653
Precision) 0.967032967032967
Recall) 0.9943502824858758
F1-Score) 0.9805013927576602
```

```
modelA ensemble inference
Accuracy) 0.9603960396039604
Precision) 0.9720670391061452
Recall) 0.9830508474576272
F1-Score) 0.9775280898876404
```

```
modelB best model inference
Accuracy) 0.963053684425393
Precision) 0.9821708743403224
Recall) 0.96672750245683
F1-Score) 0.974388000566011
```

```
modelB ensemble inference
Accuracy) 0.9633598693610941
Precision) 0.9823160296634341
Recall) 0.9670082830268145
F1-Score) 0.9746020516448531
```

	param	acc	pre	re	f1
14	{iter: 128, leaves: 31, lr: 0.01}	0.965347	0.967033	0.994350	0.980501
99	{iter: 512, leaves: 127, lr: 0.01}	0.960396	0.972067	0.983051	0.977528
44	{iter: 256, leaves: 63, lr: 0.01}	0.960396	0.972067	0.983051	0.977528
24	{iter: 128, leaves: 127, lr: 0.01}	0.960396	0.972067	0.983051	0.977528
74	{iter: 384, leaves: 127, lr: 0.01}	0.960396	0.972067	0.983051	0.977528
39	{iter: 256, leaves: 31, lr: 0.01}	0.960396	0.972067	0.983051	0.977528
34	{iter: 256, leaves: 15, lr: 0.01}	0.960396	0.961749	0.994350	0.977778
49	{iter: 256, leaves: 127, lr: 0.01}	0.960396	0.972067	0.983051	0.977528
19	{iter: 128, leaves: 63, lr: 0.01}	0.960396	0.972067	0.983051	0.977528


ModelA의 성능 상위 9개 검증 score 결과



	param	acc	pre	re	f1
87	{iter: 512, leaves: 31, lr: 0.07}	0.963462	0.982869	0.966587	0.974660
86	{iter: 512, leaves: 31, lr: 0.1}	0.963360	0.982591	0.966728	0.974595
61	{iter: 384, leaves: 31, lr: 0.1}	0.963258	0.982038	0.967149	0.974537
92	{iter: 512, leaves: 63, lr: 0.07}	0.963156	0.982449	0.966587	0.974453
62	{iter: 384, leaves: 31, lr: 0.07}	0.963054	0.982171	0.966728	0.974388
88	{iter: 512, leaves: 31, lr: 0.05}	0.962747	0.982439	0.966026	0.974163
67	{iter: 384, leaves: 63, lr: 0.07}	0.962645	0.981336	0.967008	0.974120
35	{iter: 256, leaves: 31, lr: 0.1}	0.962543	0.981746	0.966447	0.974036
91	{iter: 512, leaves: 63, lr: 0.1}	0.962135	0.982010	0.965604	0.973738

ModelB의 성능 상위 9개 검증 score 결과



Method Explain

PEParser.py



<u>Aa</u> method	 description
<u>read_json</u>	json 파일을 읽어오는 함수
<u>class FHargs</u>	FeatureHashing에 사용할 차원을 상수로 정의한 클래스
<u>FeatureHashing</u>	피쳐 해시 함수. 차원의 기본 값을 32로 한다.
<u>get_coff_info</u>	COFF 헤더 정보를 추출하는 함수
<u>get_optinal_info</u>	OPTIONAL 헤더 정보를 추출하는 함수
<u>get_imports_info</u>	IMPORTS 헤더 정보를 추출하는 함수
<u>get_exports_info</u>	EXPORTS 헤더 정보를 추출하는 함수

 method	 description
<u>get_general_file_info</u>	GENERAL 헤더 정보를 추출하는 함수
<u>get_string_info</u>	STRING 헤더 정보를 추출하는 함수
<u>get_histogram_info</u>	HISTOGRAM 헤더 정보를 추출하는 함수
<u>get_byreentropy_info</u>	BYTEENTROPY 헤더 정보를 추출하는 함수
<u>get_sections_info</u>	SECTIONS 헤더 정보를 추출하는 함수
<u>get_directory_info</u>	DIRECTORY 헤더 정보를 추출하는 함수
<u>PeminerFeatureExtract</u>	PEMINER 파일의 피처를 추출하는 함수
<u>EmberFeatureExtract</u>	EMBER 파일의 피처를 추출하는 함수
<u>PestudioFeatureExtract</u>	PESTUDIO 파일의 피처를 추출하는 함수

train.ipynb

 method	 description
<u>read_label_csv</u>	csv 파일의 내용을 읽어오는 함수
<u>train</u>	model을 입력받아 해당 모델로 학습시키는 함수
<u>getFeaturesA</u>	모델 A에 사용할 PEMINER와 EMBER의 피처를 추출하는 함수
<u>getFeaturesB</u>	모델 B에 사용할 모든 파일의 피처를 추출하는 함수

make_submission.ipynb + inference.ipynb

 method	 description
<u>scoring</u>	단일 모델을 입력받아 라벨을 예측하는 함수
<u>hard_voting</u>	앙상블 모델을 입력받아 hard voting으로 라벨을 예측하는 함수
<u>evaluate</u>	해당 모델의 검증 score를 출력하는 함수