AI 프로젝트 보고서

20181565 강동우

20202084 박지수

20202100 이수민

20191661 전하은

AI 프로젝트에서는 pe파일이 주어졌을 때, 주어진 파일이 악성파일인지 정상 파일인지 판별하는 이진 분류기 모델을 만드는 것을 수행하였습니다. 본 보고서 에서는 프로젝트를 수행하는 목표를 특징 추출 및 전처리, 학습과 성능 평가로 나누어 설명하고자 합니다.

1. 특징 추출 및 전처리

특징 추출 및 전처리 단계에서는 제공받은 pestudio, ember, peminer의 json 파일에서 특징을 추출하여 특징 벡터를 생성하였습니다. EmberParser 클래스에 범주형 데이터를 가공하여 특징을 추가하였으며 추가한 특징은 다음과 같습니다.

def get\_byteentropy\_info(self):

byteentropy = np.array(self.report["byteentropy"])

total = byteentropy.sum()

vector = byteentropy / total

return vector.tolist()

get\_byteentropy\_info()에서는 ember의 정보 엔트로피 정보를 담고 있는 byteentropy 특징을 가공하여 추가하였습니다.

def get\_exports\_info(self):

if len(self.report["exports"]) > 0:

exports = self.report["exports"]

vector = FeatureHasher(128, input\_type = "string").transform(exports).toarray()[0]

return vector.tolist()

return [0]\*128

get\_exports\_info() 에서는 ember의 exports 라는 특징을 가공하여 추가하였는데, 악성 파일과 정상 파일 사이에는 export에 차이가 있을 것이라 생각하여 가공하여 추가해 주었습니다. 이때, exports가 없는 파일을 해결해 주기 위해 exports의 값이 존재하는 것은 그 값으로 FH를 해주었고, 값이 존재하지 않는 것은 0으로 채워진 같은 크기의 리스트를 추가해주었습니다.

def get\_header\_info(self):

header = self.report["header"]

if len(header["optional"]["dll\_characteristics"]) > 0:

dll\_characteristics =FeatureHasher(10, input\_type = "string").transform(header["optional"]["dll\_characteristics"]).toarray()[0].tolist()

else: dll\_characteristics = [0]\*10

if len(header["coff"]["characteristics"]) > 0:

characteristics =FeatureHasher(10, input\_type = "string").transform(header["coff"]["characteristics"]).toarray()[0].tolist()

else: characteristics = [0]\*10

if len(header["optional"]["magic"]) > 0:

magic =FeatureHasher(10, input\_type = "string").transform(header["optional"]["magic"]).toarray()[0].tolist()

else: magic = [0]\*10

if len(header["optional"]["subsystem"]) > 0:

subsystem =FeatureHasher(10, input\_type = "string").transform(header["optional"]["subsystem"]).toarray()[0].tolist()

else: subsystem = [0]\*10

vector = []

vector += characteristics

vector += dll\_characteristics

vector += magic

vector += subsystem

vector += [

header["optional"]["sizeof\_code"],

header["optional"]["sizeof\_headers"],

header["optional"]["sizeof\_heap\_commit"]

]

return vector

get\_header\_info() 에서는 헤더 정보에서 coff 헤더의 정보와 optional 헤더의 정보를 가져와 추가해주었습니다. Coff의 characteristics과, optional의 dll\_characteristics, magic, subsystem, sizeof\_code, sizeof\_headers, sizeof\_heap\_commit 정보를 가져와 벡터에 추가해 주었습니다.

def get\_imports\_info(self):

imports = self.report["imports"]

dll = list(set([i.lower() for i in imports.keys()]))

hdll = FeatureHasher(256, input\_type = "string").transform([dll]).toarray()[0]

funct = [i.lower() + ':' + k for i, j in imports.items() for k in j]

hfunct = FeatureHasher(1024, input\_type="string").transform([funct]).toarray()[0]

vector = []

vector += hdll.tolist()

vector += hfunct.tolist()

return vector

get\_imports\_info()에서는 dll과 dll:함수 이름으로 묶어 FH 해서 벡터에 추가해 주었습니다.

def process\_report(self):

vector = []

vector += self.get\_general\_file\_info()

vector += self.get\_histogram\_info()

vector += self.get\_string\_info()

vector += self.get\_byteentropy\_info()

vector += self.get\_exports\_info()

vector += self.get\_header\_info()

vector += self.get\_imports\_info()

return vector

process\_report에서 위와 같이 벡터에 추가하여 주었습니다.

1. 학습, 성능 평가

train\_path\_l = "데이터/학습데이터\_정답.csv" #정답\_데이터가 저장되어있는 위치의 경로 변수

train\_peminer = "데이터/PEMINER/학습데이터"#peminer가 저장되어있는 위치의 경로 변수

train\_ember = "데이터/EMBER/학습데이터"#ember가 저장되어있는 위치의 경로 변수

train\_pestudio = "데이터/PESTUDIO/학습데이터"#pestudio가 저장되어있는 위치의 경로 변수

vali\_path\_l = "데이터/검증데이터\_정답.csv" #정답\_데이터가 저장되어있는 위치의 경로 변수

vali\_peminer = "데이터/PEMINER/검증데이터"#peminer가 저장되어있는 위치의 경로 변수

vali\_ember = "데이터/EMBER/검증데이터"#ember가 저장되어있는 위치의 경로 변수

vali\_pestudio = "데이터/PESTUDIO/검증데이터"#pestudio가 저장되어있는 위치의 경로 변수

train\_V, train\_w = process1(train\_path\_l, train\_peminer, train\_ember,train\_pestudio)

vali\_V, vali\_w = process1(vali\_path\_l, vali\_peminer, vali\_ember, vali\_pestudio)

학습을 위해 다음과 같이 process1 함수를 이용하여 검증데이터와 학습데이터의 V, w를 각각 만들어주었습니다.

models = []

for model in ["rf", "lgb"]:

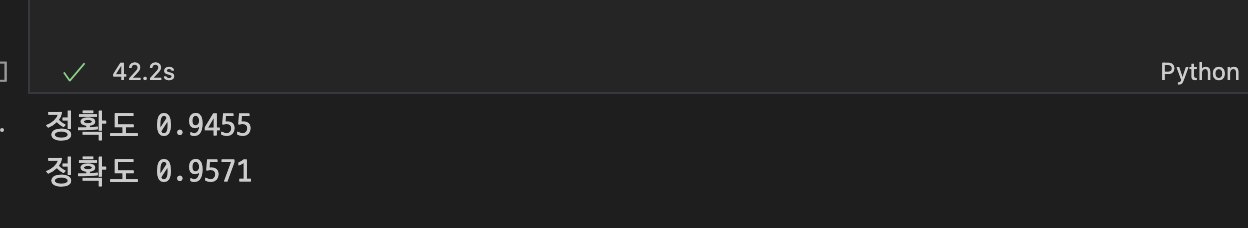
clf = train(train\_V, train\_w, model)

models.append(clf)

for model in models:

evaluate(vali\_V, vali\_w, model)

랜덤포레스트와 lgbm을 사용하여 학습시키고, 학습을 바탕으로 검증 데이터를 사용하여 정확도를 출력하였습니다. 이 때, lgbm의 n\_estimateors 값을 300으로 설정하여 주었습니다. 정확도는 다음과 같이 출력됩니다.



1. 개선점

위의 정확도를 개선하기 위하여 코드의 앙상블 예제와 특징 추출 알고리즘을 사용하여 다음과 같이 코드를 개선하였습니다.

models = []

for model in ["rf", "lgb"]:

selected\_X = select\_feature(train\_V, train\_w, model)

new\_model = train(selected\_X, train\_w, model)

models.append(new\_model)

ensemble\_result(vali\_V, vali\_w, models)

그러나 예상보다 실행에 오랜 시간이 걸려 결과까지는 확인해보지 못하였습니다. 아마 실행이 되었다면 정확도가 조금이나마 더 개선되었을 것이라 추측하고 있습니다.