#라이브러리 불러오기

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from plotly.offline import init\_notebook\_mode, iplot

import plotly.graph\_objs as go

from plotly import tools

init\_notebook\_mode(connected=True)

pd.set\_option('display.max\_columns', 500)

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from xgboost import XGBClassifier

from lightgbm import LGBMClassifier

from sklearn.model\_selection import train\_test\_split,KFold,cross\_val\_score, RandomizedSearchCV

from sklearn.metrics import confusion\_matrix, f1\_score

from sklearn.metrics import classification\_report

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score

import seaborn as sns

#데이터타입 입력

dtypes = {

'MachineIdentifier': 'category',

'EngineVersion': 'category',

'AppVersion': 'category',

'AvSigVersion': 'category',

'AVProductStatesIdentifier': 'category',

'AVProductsInstalled': 'category',

'CountryIdentifier': 'category',

'CityIdentifier': 'category',

'OrganizationIdentifier': 'float16',

'GeoNameIdentifier': 'category',

'LocaleEnglishNameIdentifier': 'category',

'Processor': 'category',

'OsBuild': 'category',

'OsSuite': 'category',

'OsPlatformSubRelease': 'category',

'OsBuildLab': 'category',

'SkuEdition': 'category',

'IsProtected': 'float16',

'SMode': 'float16',

'IeVerIdentifier': 'category',

'SmartScreen': 'category',

'Census\_MDC2FormFactor': 'category',

'Census\_OEMNameIdentifier': 'category',

'Census\_OEMModelIdentifier': 'float32',

'Census\_ProcessorCoreCount': 'float16',

'Census\_ProcessorManufacturerIdentifier': 'float16',

'Census\_ProcessorModelIdentifier': 'category',

'Census\_PrimaryDiskTotalCapacity': 'float32',

'Census\_PrimaryDiskTypeName': 'category',

'Census\_SystemVolumeTotalCapacity': 'float32',

'Census\_HasOpticalDiskDrive': 'int8',

'Census\_TotalPhysicalRAM': 'float32',

'Census\_ChassisTypeName': 'category',

'Census\_InternalPrimaryDiagonalDisplaySizeInInches': 'float16',

'Census\_InternalPrimaryDisplayResolutionHorizontal': 'float16',

'Census\_InternalPrimaryDisplayResolutionVertical': 'float16',

'Census\_PowerPlatformRoleName': 'category',

'Census\_InternalBatteryNumberOfCharges': 'category',

'Census\_OSVersion': 'category',

'Census\_OSArchitecture': 'category',

'Census\_OSBranch': 'category',

'Census\_OSBuildNumber': 'category',

'Census\_OSBuildRevision': 'category',

'Census\_OSEdition': 'category',

'Census\_OSSkuName': 'category',

'Census\_OSInstallTypeName': 'category',

'Census\_OSInstallLanguageIdentifier': 'category',

'Census\_OSUILocaleIdentifier': 'category',

'Census\_OSWUAutoUpdateOptionsName': 'category',

'Census\_GenuineStateName': 'category',

'Census\_ActivationChannel': 'category',

'Census\_IsFlightingInternal': 'float16',

'Census\_FlightRing': 'category',

'Census\_FirmwareManufacturerIdentifier': 'category',

'Census\_FirmwareVersionIdentifier': 'category',

'Census\_IsSecureBootEnabled': 'int8',

'Census\_IsTouchEnabled': 'int8',

'Census\_IsAlwaysOnAlwaysConnectedCapable': 'float16',

'Wdft\_IsGamer': 'float16',

'Wdft\_RegionIdentifier': 'category',

'HasDetections': 'category'

}

#데이터 로딩

malware = pd.read\_csv('Malware.csv', dtype=dtypes)

#대소문자로 구분되어있는 동일한 값 합치는 함수

def rename\_edition(x):

x = x.lower()

if 'core' in x:

return 'Core'

elif 'pro' in x:

return 'pro'

elif 'enterprise' in x:

return 'Enterprise'

elif 'server' in x:

return 'Server'

elif 'home' in x:

return 'Home'

elif 'education' in x:

return 'Education'

elif 'cloud' in x:

return 'Cloud'

else:

return x

#OSEdition 변수에 함수 적용

malware['Census\_OSEdition'] = malware['Census\_OSEdition'].astype(str)

malware['Census\_OSEdition'] = malware['Census\_OSEdition'].apply(rename\_edition)

malware['Census\_OSEdition'] = malware['Census\_OSEdition'].astype('category')

#OSSkuName 변수에 함수 적용

malware['Census\_OSSkuName'] = malware['Census\_OSSkuName'].astype(str)

malware['Census\_OSSkuName'] = malware['Census\_OSSkuName'].apply(rename\_edition)

malware['Census\_OSSkuName'] = malware['Census\_OSSkuName'].astype('category')

#데이터타입 별로 변수 묶기

category = str('category')

numerical\_columns = [c for c,v in dtypes.items() if v not in category]

binary\_variables = [c for c in malware.columns if malware[c].nunique() == 2]

categorical\_columns = [c for c in malware.columns

if (c not in numerical\_columns) & (c not in binary\_variables)]

#데이터타입 비율 보기

variables = {

'categorical\_columns': len(categorical\_columns),

'binary\_variables': len(binary\_variables),

'numerical\_columns': len(numerical\_columns)

}

#Pie chart 그리기

pie\_trace = go.Pie(labels=list(variables.keys()), values=list(variables.values()))

layout = dict(title= "Variable types", height=400, width=800)

fig = dict(data=[pie\_trace], layout=layout)

iplot(fig)

#OsBuildLab에 category 추가(feature engineering위해서)

malware['OsBuildLab'] = malware['OsBuildLab'].cat.add\_categories(['0.0.0.0.0-0'])

#feature Engineering 위한 함수

def fe(df):

df['EngineVersion\_2'] = df['EngineVersion'].apply(lambda x: x.split('.')[2]).astype('category')

df['EngineVersion\_3'] = df['EngineVersion'].apply(lambda x: x.split('.')[3]).astype('category')

df['AppVersion\_1'] = df['AppVersion'].apply(lambda x: x.split('.')[1]).astype('category')

df['AppVersion\_2'] = df['AppVersion'].apply(lambda x: x.split('.')[2]).astype('category')

df['AppVersion\_3'] = df['AppVersion'].apply(lambda x: x.split('.')[3]).astype('category')

df['AvSigVersion\_0'] = df['AvSigVersion'].apply(lambda x: x.split('.')[0]).astype('category')

df['AvSigVersion\_1'] = df['AvSigVersion'].apply(lambda x: x.split('.')[1]).astype('category')

df['AvSigVersion\_2'] = df['AvSigVersion'].apply(lambda x: x.split('.')[2]).astype('category')

df['OsBuildLab\_0'] = df['OsBuildLab'].apply(lambda x: x.split('.')[0]).astype('category')

df['OsBuildLab\_1'] = df['OsBuildLab'].apply(lambda x: x.split('.')[1]).astype('category')

df['OsBuildLab\_2'] = df['OsBuildLab'].apply(lambda x: x.split('.')[2]).astype('category')

df['OsBuildLab\_3'] = df['OsBuildLab'].apply(lambda x: x.split('.')[3]).astype('category')

# df['OsBuildLab\_40'] = df['OsBuildLab'].apply(lambda x: x.split('.')[-1].split('-')[0]).astype('category')

# df['OsBuildLab\_41'] = df['OsBuildLab'].apply(lambda x: x.split('.')[-1].split('-')[1]).astype('category')

df['Census\_OSVersion\_0'] = df['Census\_OSVersion'].apply(lambda x: x.split('.')[0]).astype('category')

df['Census\_OSVersion\_1'] = df['Census\_OSVersion'].apply(lambda x: x.split('.')[1]).astype('category')

df['Census\_OSVersion\_2'] = df['Census\_OSVersion'].apply(lambda x: x.split('.')[2]).astype('category')

df['Census\_OSVersion\_3'] = df['Census\_OSVersion'].apply(lambda x: x.split('.')[3]).astype('category')

return df

#malware data에 feature engineering 적용

malware = fe(malware)

#3가지 변수 제외한 categorical 변수만 리스트로 저장

cat\_cols = [col for col in malware.columns if col not in ['MachineIdentifier', 'Census\_SystemVolumeTotalCapacity', 'HasDetections'] and str(malware[col].dtype) == 'category']

len(cat\_cols)

#라벨 인코더 생성

le = LabelEncoder()

#categorical 변수에 라벨 인코더 적용

mal\_encode = malware[cat\_cols].apply(le.fit\_transform)

#mal\_encode는 categorical 변수만 포함되어 있는 DataFrame

mal\_encode.head()

#혹시 몰라서 데이터셋 따로 저장

malware\_drop\_cat = malware

#malware data에서 categorical 변수만 제거

malware\_drop\_cat = malware\_drop\_cat.drop(cat\_cols, axis=1)

#categorical 변수가 있는 mal\_encode에 나머지 numerical 변수가 남아있는 malware\_drop\_cat 연결(같은 index끼리 붙여줌)

d = pd.concat([mal\_encode, malware\_drop\_cat], axis=1)

d.head()

#데이터 분할 (train 80% / test 20%)

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test = train\_test\_split(d, test\_size=0.2)

x\_train.shape #64000개, 77개 변수

x\_test.shape #16000개, 77개 변수

#각각 저장

x\_train.to\_csv('malware\_train.csv', index=False)

x\_test.to\_csv('malware\_test.csv', index=False)

#train data불러오기

d = pd.read\_csv('malware\_train.csv')

#X, y 로 분할

y\_train = d['HasDetections']

X\_train = d[d.columns.difference(['HasDetections','MachineIdentifier'])] # HasDetection 열과 id 제외

#모델 생성

dt = DecisionTreeClassifier()

rf = RandomForestClassifier()

knn = KNeighborsClassifier(n\_neighbors=35)

lg = LogisticRegression()

xgb = XGBClassifier()

lgb = LGBMClassifier()

#교차검증모델

from sklearn.model\_selection import StratifiedShuffleSplit , cross\_val\_score

ss = StratifiedShuffleSplit(n\_splits=2,test\_size=0.2,random\_state=0)

#kfold 설정(k=10 -> 10번의 교차검증)

from sklearn.model\_selection import KFold

kfold = KFold(n\_splits=10, shuffle=True, random\_state=0)

#모델별 교차검증 정확도 점수 저장

dt\_cv = cross\_val\_score(dt,X\_train,y\_train, scoring=‘accuracy’ ,cv=kfold)

rf\_cv = cross\_val\_score(rf,X\_train,y\_train,scoring=‘accuracy’ ,cv=kfold)

knn\_cv = cross\_val\_score(knn,X\_train,y\_train,scoring=‘accuracy’ , cv=kfold)

lg\_cv = cross\_val\_score(lg,X\_train,y\_train,scoring=‘accuracy’ ,cv=kfold)

xgb\_cv = cross\_val\_score(xgb,X\_train,y\_train,scoring=‘accuracy’ ,cv=kfold)

lgb\_cv = cross\_val\_score(lgb,X\_train,y\_train, scoring=‘accuracy’ ,cv=kfold)

#10번 점수 print

print('DT n\_splits={}, cross validation score: {}'.format(10, dt\_cv))

print('RF n\_splits={}, cross validation score: {}'.format(10, rf\_cv))

print('Knn n\_splits={}, cross validation score: {}'.format(10, knn\_cv))

print('Lg n\_splits={}, cross validation score: {}'.format(10, lg\_cv))

print('XGB n\_splits={}, cross validation score: {}'.format(10, xgb\_cv))

print('LGB n\_splits={}, cross validation score: {}'.format(10, lgb\_cv))

#10번 평균 print

print('DecisionTree.mean \n{:.3f}'.format(dt\_cv.mean()))

print('RandomForest.mean \n{:.3f}'.format(rf\_cv.mean()))

print('Knn.mean \n{:.3f}'.format(knn\_cv.mean()))

print('LogisticRegression.mean \n{:.3f}'.format(lg\_cv.mean()))

print('XGBoost.mean \n{:.3f}'.format(xgb\_cv.mean()))

#feature importance 추출

#decision Tree

f = pd.DataFrame(data={'columns': test\_X.columns,'feature\_importance':dt.feature\_importances\_})

f2 = f.sort\_values(by='feature\_importance',ascending=False).reset\_index().drop('index',axis=1)

dt\_xtrain = X\_train[[f2]]

#Randomforest

rf\_f = pd.DataFrame(data={'columns': test\_X.columns,'feature\_importance':rf.feature\_importances\_})

rf\_f2 = f.sort\_values(by='feature\_importance',ascending=False).reset\_index().drop('index',axis=1)

rf\_xtrain = X\_train[[f2]]

#xgboost

xgb\_f = pd.DataFrame(data={'columns': test\_X.columns,'feature\_importance':xgb.feature\_importances\_})

xgb\_f2 = f.sort\_values(by='feature\_importance',ascending=False).reset\_index().drop('index',axis=1)

xgb\_xtrain = X\_train[[f2]]

#lgbm

lgb\_f = pd.DataFrame(data={'columns': test\_X.columns,'feature\_importance':lgb.feature\_importances\_})

lgb\_f2 = f.sort\_values(by='feature\_importance',ascending=False).reset\_index().drop('index',axis=1)

lgb\_xtrain = X\_train[[f2]]

#변수 20개로 cross validation실행

dt\_cv = cross\_val\_score(dt,dt\_xtrain,y\_train, scoring=‘accuracy’ ,cv=kfold)

rf\_cv = cross\_val\_score(rf,rf\_xtrain,y\_train,scoring=‘accuracy’ ,cv=kfold)

knn\_cv = cross\_val\_score(knn,dt\_xtrain,y\_train,scoring=‘accuracy’ , cv=kfold)

lg\_cv = cross\_val\_score(lg,dt\_xtrain,y\_train,scoring=‘accuracy’ ,cv=kfold)

xgb\_cv = cross\_val\_score(xgb,xgb\_xtrain,y\_train,scoring=‘accuracy’ ,cv=kfold)

lgb\_cv = cross\_val\_score(lgb,lgb\_xtrain,y\_train, scoring=‘accuracy’ ,cv=kfold)

#10번 점수 print

print('DT n\_splits={}, cross validation score: {}'.format(10, dt\_cv))

print('RF n\_splits={}, cross validation score: {}'.format(10, rf\_cv))

print('Knn n\_splits={}, cross validation score: {}'.format(10, knn\_cv))

print('Lg n\_splits={}, cross validation score: {}'.format(10, lg\_cv))

print('XGB n\_splits={}, cross validation score: {}'.format(10, xgb\_cv))

print('LGB n\_splits={}, cross validation score: {}'.format(10, lgb\_cv))

#10번 평균 print

print('DecisionTree.mean \n{:.3f}'.format(dt\_cv.mean()))

print('RandomForest.mean \n{:.3f}'.format(rf\_cv.mean()))

print('Knn.mean \n{:.3f}'.format(knn\_cv.mean()))

print('LogisticRegression.mean \n{:.3f}'.format(lg\_cv.mean()))

print('XGBoost.mean \n{:.3f}'.format(xgb\_cv.mean()))

print('LightGBM.mean \n{:.3f}'.format(xgb\_cv.mean()))

#변수 20개 data 저장

train20 = pd.read\_csv('train\_20.csv')

#test data 로드

test = pd.read\_csv('malware\_test.csv')

# label 분리

test\_X = test.drop(['HasDetections','MachineIdentifier'], axis=1)

test\_y = test['HasDetections']

#최종 모델에 학습

lgb\_clf.fit(train20, y\_train)

#test data 예측

final\_pred = lgb\_clf.predict(test\_X)

#정확도 계산

accuracy\_lgb = accuracy\_score(test\_y,final\_pred)

print('final accuracy\_score on lgb : ', accuracy\_lgb)

#feature importance 그래프

plt.figure(figsize=(7,5))

sns.barplot(x="feature\_importance",

y="columns",

data=f3.sort\_values(by="feature\_importance",

ascending=False))

plt.title('LightGBM Features')

plt.tight\_layout()

plt.savefig('lgbm\_importances.png')