CSE4003

—

Cyber Security

Digital Assignment -3

Yerramalli Sai Sreekar / 20BCE1296

C1 / Slot

Dr. Subbulakshmi T/ Professor

Implementation & Results:

Machine Learning Models:

```
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.decomposition import PCA
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_predict
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import GridSearchCV
import pefile
import os
import numpy as np
import pandas as pd
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from fastapi import Request, FastAPI
from fastapi import File, UploadFile
from starlette.responses import FileResponse
from pydantic import BaseModel
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
import warnings
warnings.filterwarnings('ignore')
app = FastAPI()


data = pd.read_csv('C:/Users/saisr/Downloads/dataset_malwares.csv')
Data = data.dropna(how="any", axis=0)
Data.head()
X = Data.drop(['Name', 'Malware', 'e_magic', 'e_cblp', 'e_cp', 'e_crlc',
'e_cparhdr', 'e_minalloc', 'e_maxalloc', 'e_ss', 'e_sp', 'e_csum', 'e_ip',
'e_cs', 'e_lfarlc',
              'e_ovno', 'e_oemid', 'e_lfanew', 'e_oeminfo',
'SuspiciousImportFunctions', 'SuspiciousNameSection', 'DirectoryEntryImport',
'DirectoryEntryImportSize', 'DirectoryEntryExport'], axis=1)


Y = Data['Malware']
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=1)
X_train.head()


print("\n----------------------------------------------------------------\n")
print("Feature Importance")
mod = ExtraTreesClassifier().fit(X,Y)
```

```
model = SelectFromModel(mod,prefit=True)

X_select = model.transform(X)

nb_features = 25

print(nb_features)


indices = np.argsort(mod.feature_importances_)[::-1][:nb_features]

for i in range(nb_features):

    print("%d.feature %s(%f)" %
(i+1,data.columns[2+indices[i]],mod.feature_importances_[indices[i]]))


print("\n----------------------------------------------------------------\n")


#Scaking

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X_train)


#Naive Bayes

gnb = GaussianNB()

gnb.fit(X_scaled, Y_train)

Y_pred = gnb.predict(X_test)

print(classification_report(Y_pred, Y_test))

print("Gaussian Naive Bayes model accuracy(in %) is ",

      metrics.accuracy_score(Y_test, Y_pred)*100)

print("\n----------------------------------------------------------------\n")


X_new = pd.DataFrame(X_scaled, columns=X.columns)

X_new.head()


#Random Forest

model = RFC(n_estimators=100, random_state=0,
```

```
            oob_score=True,
            max_depth=16,
            max_features='sqrt')
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)
print(classification_report(Y_pred, Y_test))
print("Random Forest model accuracy(in %):",
      metrics.accuracy_score(Y_test, Y_pred)*100)


print("\n----------------------------------------------------------------\n")



# using Principal Component Analysis to increase interpretability
skpca = PCA(n_components=55)
X_pca = skpca.fit_transform(X_new)
print('Variance sum : ', skpca.explained_variance_ratio_.cumsum()[-1])


print("\n----------------------------------------------------------------\n")


# Random Forest Using PCA
model = RFC(n_estimators=100, random_state=0,
            oob_score=True,
            max_depth=16,
            max_features='sqrt')
model.fit(X_pca, Y_train)
X_test_scaled = scaler.transform(X_test)
X_test_new = pd.DataFrame(X_test_scaled, columns=X.columns)
X_test_pca = skpca.transform(X_test_new)
Y_pred = model.predict(X_test_pca)
print(classification_report(Y_pred, Y_test))
```

```
print("Random Forest model accuracy(in %) using PCA:",
      metrics.accuracy_score(Y_test, Y_pred)*100)



print("\n----------------------------------------------------------------\n")



#Ensemble Model


tree = DecisionTreeClassifier(max_depth=5, random_state=42)
rf = RFC(n_estimators=50, max_depth=10, random_state=42)
svm = SVC(kernel='rbf', C=10, gamma=0.1, random_state=42)

# create the ensemble model using majority voting
ensemble = VotingClassifier(estimators=[('tree', tree), ('rf', rf), ('svm',
svm)], voting='hard')

# train the ensemble model
ensemble.fit(X_pca, Y_train)

# make predictions on the testing set
y_pred = ensemble.predict(X_test_pca)

# evaluate the performance of the ensemble model
acc = accuracy_score(y_pred,Y_test)
print(classification_report(Y_pred, Y_test))
print('Ensemble Voting Classifier Accuracy: {:.2f}%'.format(acc*100))

print("\n----------------------------------------------------------------\n")
```

```
#USing Ada Boosting Classifier instead of Voting Classifier
ensemble = AdaBoostClassifier(base_estimator=RFC(n_estimators=100,
random_state=0,
            oob_score=True,
            max_depth=16,
            max_features='sqrt'), n_estimators=50, learning_rate=1.0,
algorithm='SAMME.R', random_state=None)

# fit the ensemble model on the data
ensemble.fit(X_train, Y_train)

# predict the labels of the test data using the ensemble model
y_pred = ensemble.predict(X_test)

# evaluate the performance of the ensemble model
accuracy = accuracy_score(y_pred,Y_test)
print(classification_report(Y_pred, Y_test))
print("Ada Boosting Accuracy: %.2f%%" % (accuracy * 100.0))

print("\n----------------------------------------------------------------\n")

adb = AdaBoostClassifier()
model = adb.fit(X_train, Y_train)
y_pred = model.predict(X_test)
score = metrics.accuracy_score(Y_test,y_pred)
```

Backend:

```
app.mount("/static", StaticFiles(directory="static"), name="static")
templates = Jinja2Templates(directory="templates")


@app.get("/")
async def read_items(request: Request):


    return templates.TemplateResponse('index.html', {"request": request})



@app.post("/predict")
async def create_upload_file(request: Request, file: UploadFile = File(...)):
    try:
        print(file.read())
        pe = pefile.PE(data=file.file.read())
        dict_df = pd.DataFrame.from_dict([extract(pe)])
        # print(pdf)
        # print(extract(pe))
        # print(model.predict(dict_df))
        # # y_pred = model.predict(X_test)
        # # y_pred q
        # # print(confusion_matrix(Y_test, y_pred))
        # # print(classification_report(Y_test, y_pred))
        # # print(metrics.accuracy_score(Y_test, y_pred))
        Y_pred = model.predict(dict_df)
        print("Ypred is " , Y_pred)
        if (Y_pred == [0]):
```

```
            return templates.TemplateResponse('pg2.html', {"request": request})
        else:
            return templates.TemplateResponse('yes.html', {"request": request})
    except Exception as e:
        print(e)
```

TABLE I

Classification Report of Gaussian Naïve Bayes

	Precision	Recall	F1 score	Support
0	1.00	0.27	0.42	3635
1	0.10	0.99	0.18	288
Accuracy			0.32	3923
Macro Average	0.55	0.63	0.30	3923
Weighted Average	0.93	0.32	0.41	3923

TABLE II
Classification Report of Random Forest Model

	Precision	Recall	F1 score	Support
0	0.96	0.99	0.97	961
1	1.00	0.99	0.99	2962
Accuracy			0.99	3923
Macro Average	0.98	0.99	0.98	3923
Weighted Average	0.99	0.99	0.99	3923

Website:

TABLE III

Classification Report of Random Forest Model USING PCA

	Precision	**Recall**	**F1 score**	**Support**
0	0.99	0.99	0.99	970
1	1.00	0.99	1.00	2953
Accuracy			0.99	3923
Macro Average	0.99	0.99	0.99	3923
Weighted Average	0.99	0.99	0.99	3923