BÁO CÁO BÀI TẬP 1

Môn học: Phương pháp học máy trong an toàn thông tin

Tên chủ đề: Feature Selection

GVHD: Phan Thế Duy

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: .....

STT	Họ và tên	MSSV	Email
1	Trần Hoàng Khang	19521671	19521671@gm.uit.edu.vn
2	Nguyễn Tú Ngọc	20521665	20521665@gm.uit.edu.vn
3	Lê Hồng Bằng	19520396	19520396@gm.uit.edu.vn+

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá
1	Phân tích chung về bộ dữ liệu	100%
1.1	Tổng số nhãn, tổng số thuộc tính	100%
1.2	Xem thông tin chung các giá trị của các thuộc tính, các giá trị **NA**	100%
1.3	Thông tin về giá trị means, median, min, max, … (Làm thêm cho vui)	100%
2	Tìm hiểu và thực hiện đầy đủ 03 nhóm phương pháp: **Wrapper, Filter, Intrinsic** trong *Feature Selection*	100%
2.1	Dùng **Filter**, sử dụng các phương pháp loại bỏ feature bằng phương pháp truyền thống	100%
a	Bỏ feature có nhiều giá trị ***NA (Not Available)***	100%
b	Bỏ feature bằng ***VarianceThreshold***	100%
c	Bỏ feature bằng các hệ số ước lượng bằng ***Pearson Coefficient***	100%
2.2	Dùng **Wrapper**, sử dụng kỹ thuật ***Sequential Feature Selector***	100%

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

a	Chỉ thực hiện làm mẫu với phương pháp ***Sequential Forward Selection***	100%
2.3	Dùng **Intrinsic**, sử dụng các thuật toán phổ biến đối với phương pháp này	100%
a	Dùng thuật toán ***Lasso***: tạo dataset, build model, train, xuất kết quả và tìm tham số tối ưu. Đồng thời so sánh mức độ hiệu quả với các thuật toán khác	100%
b	Dùng thuật toán ***Ridge***: tạo dataset, build model, train, xuất kết quả và tìm tham số tối ưu. Đồng thời so sánh mức độ hiệu quả với các thuật toán khác	100%
c	Dùng thuật toán ***ElasticNet***: tạo dataset, build model, train, xuất kết quả và tìm tham số tối ưu. Đồng thời so sánh mức độ hiệu quả với các thuật toán khác	100%
3	Thực nghiệm trên model thực tế (một model khác)	80%
3.1	Xây dựng model Machine Learning	80%
a	Dùng Gradient Boosting	80%
3.2	Xây dựng model Deep Learning	80%
b	Dùng	80%

BÁO CÁO CHI TIẾT

1. Phân tích chung về bộ dữ liệu (tổng số nhãn, tổng số thuộc tính, phân phối của các nhãn dữ liệu…). Yêu cầu lập trình code để in các thông tin này ra màn hình quan sát. Thông tin bộ dữ liệu gốc, từ paper: Identifying Useful Features for Malware Detection

TABLE I: Feature Groups Included in the Ember Dataset

Feature group	Number of features
General file information (general)	10
Header information (header)	62
Imported functions (imports)	1,280
Exported functions (exports)	128
Section information (section)	255
Byte histogram (histogram)	256
Byte-entropy histogram (byteentropy)	256
String information (strings)	104
All	2,351

a. (Dataset gốc)Thông tin số lượng các nhãn:

```
[ ]  import numpy as np

     # Show all possible value of 'label' and their frequency
     np.unique(label_list, return_counts=True)

     (array([-1,  0,  1]), array([ 2698, 13397,  3531]))
```

b. (Dataset khác) Thông tin chung:

Thừa nhận dataset của https://ieee-dataport.org/authors/quynh-trinh trong link sau: Maliciou_Benign_PE_file_dataset.

Note: *Việc trích xuất file Jsonl thành file CSV mình đã làm được nhưng dữ liệu đối với một số trường vẫn đang ở dạng Object, mình cần phải biến đổi thêm, tuy nhiên phần này mình chưa thực hiện nên để cho nhanh và tiện thì chúng ta công nhận một dataset khác*

Dataset này được trích xuất sẵn dựa vào dữ liệu 2017 (cũng thuộc bộ dữ liệu của EMBER) nên có ít thuộc tính hơn. Đồng thời, các features trong dataset chứa các dữ

liệu và các thư viện được ***imported*** và ***exported***, được gán giá trị 0 (không) và 1(có). Song, mình thấy số lượng feature vẫn khá đầy đủ để phục vụ cho việc thực nghiệm Feature Selection

Tập dataset mới chỉ có 2 nhãn:

```
import numpy as np
np.unique(df_main['label'], return_counts=True) # Show unique label's values and their frequency
```

```
(array([0, 1]), array([382, 617]))
```

Sau đó ta loại bỏ các cột không cần thiết đi

```
df_main = df_main.drop(['sha256', 'appeared'], axis=1)
```

Lực hiện lấy tập dữ liệu để rtain:

```
df_main_train = df_main.drop("label", axis=1)
```

Thông tin về số lượng (mẫu, đặc trưng):

```
df_main.shape # -> (a, b): a samples, b properties
```

```
(999, 1001)
```

Số mẫu: 999

Số lượng feature: 1001

Note: *Mình thực hiện lấy 999 mẫu để demo chạy đỡ lâu. Nếu để > 10000 thì thời gian xử lý của các thuật toán ở các lần sau lên đến cả tiếng (tầm >5 tiếng)*

Xem tổng quát "hình dạng" của mẫu dữ liệu

```
df_main.head(5) # Overview in our data
```

	label	GetProcAddress	ExitProcess	WriteFile	GetLastError	CloseHandle	FreeLibrary	Sleep	GetStdHandle	MultiByteToWideChar	...	DuplicateToken	bind	RegEnumKeyExA	WinHttpOpen	_controlfp	WinExec	GetSe
0	1	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
3	1	1	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	
4	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

5 rows × 1001 columns

Thông tin một số thông số đặc biệt *(count, means, average, min, max, ...)*:

	label	GetProcAddress	ExitProcess	WriteFile	GetLastError	CloseHandle	FreeLibrary	Sleep	GetStdHandle	MultiByteToWideChar	...	DuplicateToken	bind	RegEnumKeyExA	WinHttpOpen	_contro
count	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	...	999.000000	999.000000	999.000000	999.000000	999.00
mean	0.617618	0.332332	0.216216	0.172172	0.237237	0.214214	0.148148	0.217217	0.107107	0.161161	...	0.011011	0.008008	0.010010	0.108108	0.027
std	0.486213	0.471285	0.411870	0.377719	0.425602	0.410482	0.355425	0.412558	0.309404	0.367864	...	0.104406	0.089173	0.099598	0.310672	0.162
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000
50%	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000
75%	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.000000	1.000000	1.000000	1.000

8 rows × 1001 columns

2. Feature Selection:

a. Sử dụng nhóm phương pháp Filter:

Định nghĩa cơ bản: Filter method là một trong 3 nhóm phương pháp chính cho Feature Selection và xếp hạng từng feature dựa trên một số chỉ số "đơn biến" (uni-variate) và sau đó chọn các feature có xếp hạng cao nhất. Cách tính toán chỉ số ranking này tùy thuộc vào thuật toán ta sử dụng

Tham khảo: https://medium.com/mlearning-ai/feature-selection-using-filter-method-python-implementation-from-scratch-375d86389003

i. Loại bỏ những feature có giá trị hằng (constant):

Giải thích ngắn gọn: Các feature không đổi là loại feature chỉ chứa một giá trị cho mọi đầu ra trong tập dữ liệu. Các feature cố định không cung cấp thông tin nào có thể giúp ích cho việc phân loại bản ghi hiện có. Do đó, nên xóa tất cả các feature này

Sau khi loại bỏ các feature nói trên:

```
df_main_train = df_main_train.drop(constant_columns,axis=1)
df_main_train.shape
```

```
(999, 956)
```

Số lượng feature hiện tại còn **956** (so với **1001**)

ii. Loại bỏ Quasi-Constant Features:

Giải thích ngắn gọn : Các feature *gần như không đổi (quasi),* như tên cho thấy, là các feature gần như không đổi. Các feature như vậy không hữu ích lắm để đưa ra dự đoán. *Không có quy tắc nào về ngưỡng cho phương sai của các quasi-feature.*

TIếp tục với số lượng feature trên, tạo một đối tượng VarianceThreshold với ngưỡng là 0.1

```
from sklearn.feature_selection import VarianceThreshold

nearly_constant_filter = VarianceThreshold(threshold = 0.01)
```

Tuy nhiên, theo nguyên tắc chung, hãy loại bỏ các đặc trưng gần như không đổi có giá trị tương tự hơn 99% cho các quan sát đầu ra.

Sau khi loại bỏ các feature nói trên:

```
df_main_train = df_main_train.drop(nearly_constant_columns, axis=1)
df_main_train.shape
```

```
(999, 612)
```

Số lượng feature hiện tại còn **612** (so với **956**)

iii. Loại bỏ Columns bị trùng nhau:

Giải thích ngắn gọn : Các feature trùng lặp là các feature có giá trị tương tự nhau. Các feature trùng lặp không bổ sung bất kỳ giá trị nào cho quá trình training thuật toán, thay vào đó chúng thêm chi phí hoạt động và độ trễ không cần thiết vào thời gian đào tạo. Do đó, luôn luôn nên xóa các feature trùng lặp khỏi tập dữ liệu trước khi đào tạo.

Tiếp tục với số lượng feature trên, ta loại bỏ các cột có giá trị trung nhau bằng cách xử lý dataframe như với dòng, rồi xóa bỏ các dòng có giá trị bị duplicated.

Hoán vị dataframe của chúng ta:

```
df_main_train_T = df_main_train.T
df_main_train_T.shape
```

```
(612, 999)
```

Tìm những hàng bị trùng nhau và loại bỏ chúng

```
df_main_train_T.duplicated().sum()
```

```
31
```

```
df_main_train = df_main_train_T.drop_duplicates().T
df_main_train.shape
```

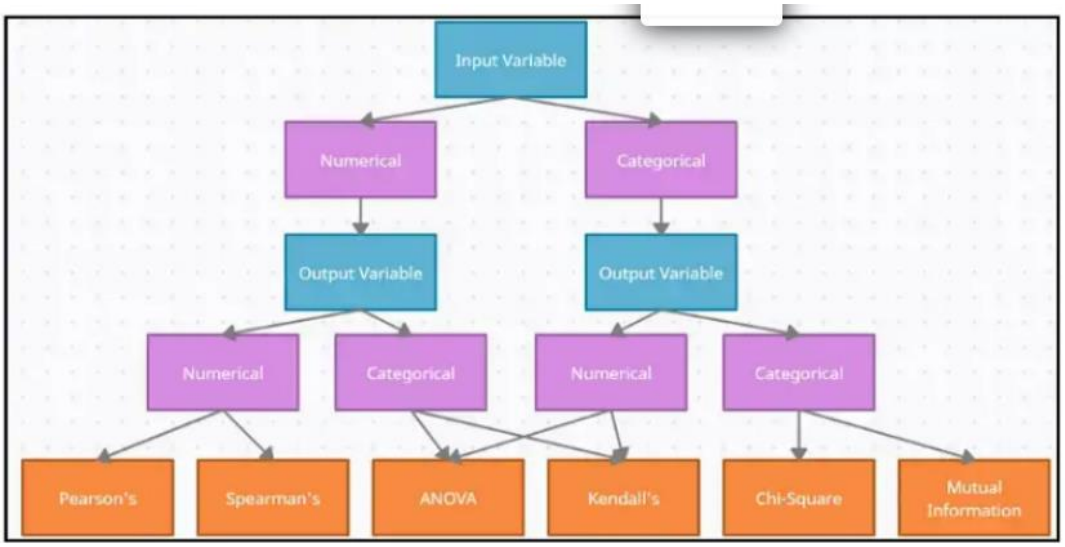
```
(999, 581)
```

Số lượng feature hiện tại còn **581** (so với **612**)

iv. Tính hệ số tương quan (Correlation) với cột đích (label) và loại bỏ các hệ số thấp:

Giải thích ngắn gọn: Tính hệ số tương quan hay còn gọi mức độ tương quan (Correlation) với cột đích (label) bằng phương pháp Pearson và chọn các cột có hệ số tương quan cao sao cho "phù hợp", sự "phù hợp" này tùy theo số lượng feature cuối cùng chúng ta chọn

Cách chọn thuật toán phù hợp nhất để xác định mối tương quan của tính năng



Trong bài tập này, mình sẽ thử phương pháp được đề xuất đầu tiên: *Hệ số Pearson*

Nguồn: https://medium.com/analytics-vidhya/feature-selection-in-machine-learning-ec1f5d053007

TIếp tục sử dụng các feature đã được trích xuất trên, ta tính toán các hệ số Pearson

```
# Use Pearson's coefficient for feature selection
imp = full_data.drop("label", axis=1).apply(lambda x: x.corr(full_data.label, method='pearson'))
imp
```

```
GetProcAddress     -0.017706
ExitProcess         0.002975
WriteFile          -0.154024
GetLastError       -0.214873
CloseHandle        -0.151470
                       ...
SetFileSecurityA    0.083025
DuplicateToken      0.043547
WinHttpOpen         0.273944
_controlfp         -0.097497
GetTimeFormatW     -0.073229
Length: 581, dtype: float64
```

Sắp xếp lại cho dễ nhìn bằng numpy.argsort(array) và in ra màn hình:

```
import numpy as np

indices = np.argsort(imp)
indices['WriteFile']
```

```
363
```

```
imp[indices]    # Sorted in ascending order
```

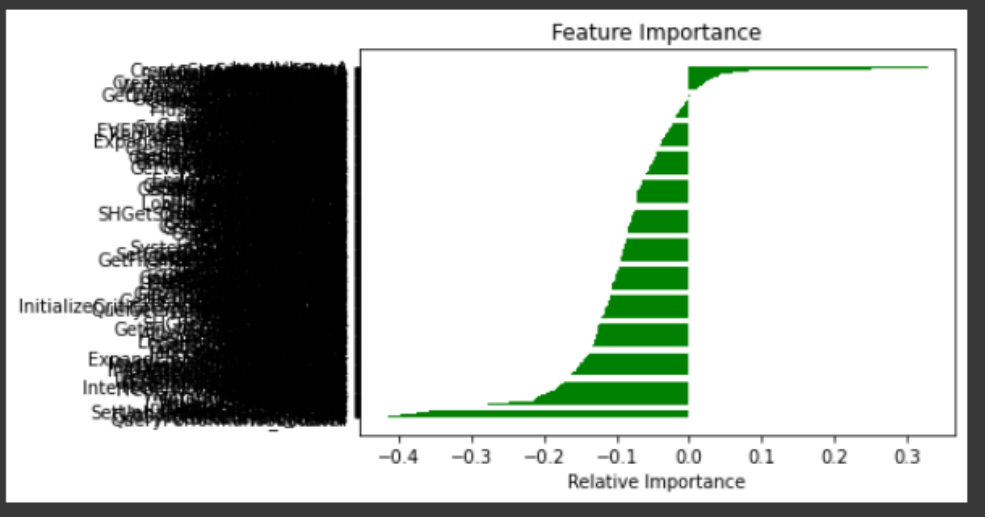
```
QueryPerformanceCounter    -0.417110
GetCurrentThreadId         -0.416901
GetCurrentThread           -0.411448
_initterm                  -0.400998
GetSystemTime              -0.398665
                               ...
wsprintfA                   0.187561
CryptHashData               0.252538
CreateStreamOnHGlobal       0.261702
WinHttpOpen                 0.273944
LoadLibraryA                0.329393
Length: 581, dtype: float64
```

Vẽ biểu đồ để xem thông tin vùng giá trị:

```
import matplotlib.pyplot as plt

names = X.columns
plt.title('Feature Importance')

# Plotting horizontal bar graph
plt.barh(range(len(indices)), imp[indices], color='g', align='center')
plt.yticks(range(len(indices)), [names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Sau khi thử một số threshold (ngưỡng), mình chọn 0,1 để thu hẹp số lượng feature

```
feat_imp_list = []
for i in range(0, len(indices)):
    if np.abs(imp[i])>0.1:
        feat_imp_list.append(names[i])

len(feat_imp_list)

245

X_new = X[feat_imp_list] # Producing new list of features for real-model training
X_new.shape
```

Số lượng feature hiện tại còn **245** (so với **581**)

Và đó là một chuỗi các chiến lược khi dùng phương pháp Filter, từ **1001** thành **245** feature

b. Sử dụng nhóm phương pháp Wrapper:

Định nghĩa cơ bản Wrapper method đánh giá nhiều mô hình bằng cách sử dụng các quy trình thêm và/hoặc loại bỏ các yếu tố dự đoán để tìm ra sự kết hợp tối ưu giúp tối đa hóa hiệu suất của mô hình. Các quy trình này thường được xây dựng sau khi khái niệm về thuật toán *"Tìm kiếm tham lam."* ra đời. Thuật toán tham lam là bất kỳ thuật toán nào tuân theo kinh nghiệm giải quyết vấn đề để đưa ra lựa chọn tối ưu cục bộ ở mỗi giai đoạn. Và những thuật toán này sẽ cố overfitting dữ liệu của chúng ta.

Sequential Feature Selector là một chiến lược để chọn một tập hợp con các tính năng có hiệu suất tốt nhất. Điều này bao gồm 4 chiến lược chính:

- Sequential Forward Selection (SFS)
- Sequential Backward Selection (SBS)
- Sequential Backward Floating Selection (SBFS)
- Sequential Forward Floating Selection (SFFS) Chúng tôi sử dụng sklearn để biến các thử nghiệm thành hành động

Tham khảo (Giải thích đầy đủ):

http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/

Các thông số đánh giá: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SequentialFeatureSelector.html

Ta lấy lại dataset ban đầu với số chiều là (999, 1001). Ở đây, mình sẽ sử dụng chiến lược đầu tiên: **Sequential Forward Selection**

Tạo một model với thuật toán **Kneighbor**:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=4)
```

Tạo bộ chọn Wrapper với chiến lược nói trên và đưa dữ liệu vào tập train.

Note: *Việc train bằng phương pháp Wrapper hay các thuật toán tham lam nói chung đều rất lâu, do vậy bước này sẽ khá take time* 💔

```
from sklearn.feature_selection import SequentialFeatureSelector

# Using K-nearest neighbors algorithm
sfs = SequentialFeatureSelector(knn,
                                n_features_to_select=20,
                                direction='forward',
                                scoring='accuracy') # The number of


sfs.fit(X, y)
```

Lấy ra những index của các feature quan trọng sau khi train:

```
import numpy as np
index_array = np.where(sfs.get_support() == True)
index_array
```

```
(array([  2,   4,   9,  85, 134, 160, 265, 287, 328, 391, 404, 517, 531,
        568, 631, 669, 814, 863, 882, 991]),)
```

Với index tìm được, xem thông tin tên feature.

```
extracted_feature = X.columns[index_array]
extracted_feature
```

```
Index(['WriteFile', 'CloseHandle', 'GetCurrentThreadId', 'GetTickCount',
       'IsChild', 'EnumWindows', 'CreatePalette', 'InterlockedPushEntrySList',
       'IsDialogMessageA', 'UnrealizeObject', 'ImageList_GetDragImage',
       'CreateRectRgn', 'VerifyVersionInfoW', 'GetTextExtentPoint32A',
       'VirtualQueryEx', 'Chord', 'GetShortPathNameW', 'connect', '_initterm',
       'bind'],
      dtype='object')
```

```
len(extracted_feature)
```

```
20
```

Vậy số lượng feature cuối cùng là **20** (so với **1001**)

c. Sử dụng nhóm phương pháp Intrinsic(Embedded) Method:

Các phương thức **Embedded** kết hợp các phẩm chất của **Filter** và **Wrapper**. Nó được triển khai bởi các thuật toán có các phương thức lựa chọn tính năng tích hợp sẵn của riêng chúng. Bản chất, **Embedded** là phương pháp bỏ dữ liệu của chúng ta vào một model thực và liên tục đánh giá và tối ưu nó

Một số ví dụ phổ biến nhất của các phương pháp này là hồi quy LASSO và RIDGE có các chức năng xử phạt sẵn có để giảm tình trạng thừa. Ngoài ra, còn có thuật toán ElasticNet được đề xuất là sự kết hợp của cả hai.

Note: *Trong file .ipynb, mình triển khai cả ba thuật toán nói trên để so sánh mức độ hiệu quả*

Ta lấy lại dataset ban đầu với số chiều là (999, 1001).

i. Lasso algorithm:

Tạo model cho thuật toán và fit() dữ liệu của chúng ta vào model

```
from sklearn.linear_model import LassoCV

lasso_reg_cv = LassoCV(max_iter=100)
lasso_reg_cv.fit(X_train, y_train)

print("Best alpha using built-in LassoCV: %f" % lasso_reg_cv.alpha_)
print("Best score using built-in LassoCV: %f" % lasso_reg_cv.score(X_train,y_train))
print('Best testing score using built-in LassoCV: %f' % lasso_reg_cv.score(X_test,y_test))
```

Note *Khác với Lasso, thư viện LassoCV của sklearn cho phép tìm các tham số tối ưu nhất khi chạy model*

```
Best alpha using built-in LassoCV: 0.002235
Best score using built-in LassoCV: 0.550805
Best testing score using built-in LassoCV: 0.476052
```

Đo lường các thông tin sai số:

<Bổ sung>

```
from sklearn import metrics

y_pred = lasso_reg_cv.predict(X_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 0.2834651083665724
Mean Squared Error: 0.1272014980163987
Root Mean Squared Error: 0.35665319011106394
```

Phần trên là giai đoạn training model, quá trình diễn ra khá nhanh, tương tự như **Wrapper** method. Tiếp theo đến phần trích xuất ra các hệ số đánh giá theo tư duy của **Filter** method

```
coef = pd.Series(lasso_reg_cv.coef_, index = X_train.columns)
coef
```

```
GetProcAddress      -0.051668
ExitProcess         -0.001574
WriteFile           -0.000000
GetLastError         0.000000
CloseHandle          0.060437
                        ...
SetFileSecurityA     0.382338
DuplicateToken       0.000000
WinHttpOpen          0.279973
_controlfp           0.000000
GetTimeFormatW      -0.000000
Length: 581, dtype: float64
```

Chọn các đặc trưng theo một "ngưỡng" (threhold) xác định. Ở đây mình chọn **0**

```
extracted_feat = coef.where(coef > 0).dropna()
extracted_feat
```

```
CloseHandle                 0.060437
VirtualAlloc                0.216212
ReadFile                    0.125749
RaiseException              0.006771
GetModuleHandleW            0.008588
GetCurrentProcessId         0.085219
CreateFileW                 0.109653
SetLastError                0.035124
GetCommandLineW             0.008029
GetModuleHandleA            0.079842
IsWindow                    0.025152
LoadLibraryA                0.396409
RegOpenKeyExW               0.086525
CreateCompatibleBitmap      0.087823
IsProcessorFeaturePresent   0.053241
GetWindowThreadProcessId    0.060508
PostMessageW                0.174395
CreateStreamOnHGlobal       0.094002
UnregisterClassW            0.001322
lstrlenA                    0.003163
InterlockedExchange         0.003819
DefWindowProcA              0.092492
Rectangle                   0.057368
CreateProcessA              0.056322
ShellExecuteA               0.184437
SetFileAttributesA          0.016507
ExpandEnvironmentStringsA   0.008977
wsprintfW                   0.005397
HeapSetInformation          0.054691
DuplicateHandle             0.028837
__p__fmode                  0.023996
SetFileSecurityA            0.382338
WinHttpOpen                 0.279973
dtype: float64
```

Số feature được trích xuất là **33** (so với **1001** ban đầu)

```
X_new = pd.DataFrame(df_main_train, columns=list(extracted_feat.index)) # Producing new list of features for real-model training
X_new.shape
```

```
(999, 33)
```

Thực hiện tương tự với 2 thuật toán **Ridge** và **ElasticNet**. Ta có bảng so sánh sau

3. Áp dụng trên mô hình thực tế với ML và DL:

a. Machine Learning:

- Tách bộ train_test_plit từ sklearn.model_selection
- Do dữ liệu vào model để huấn luyện với các tham số mặc định

```
from sklearn.ensemble import GradientBoostingClassifier

# Default:
# learning_rate = 0.1
# n_estimators = 100

clf = GradientBoostingClassifier()
clf.fit(X_train, y_train)
```

Kết quả các thông số cơ bản: **Accuracy, Recall, Precision và F1-score**

```
Accuracy:  0.82
False positive rate(FPR):  0.08433735
False negative rate(FN):  0.24786325
Recall(TPR):  0.75213675
Precision:  0.92631579
F1 score:  0.83018868
```

b. Deep Learning:

Thực hiện với model Deep learning với thông tin được sử dụng sau:

Tên loại	Giá trị
Loại model	Sequential()
Số lượng hidden Layer	3
Layer 1 (lớp input): số lượng node	12
Activation function	Relu
Layer 2: số lượng node	8
Activation function	Relu
Layer 3: số lượng node	1
Activation function	Sigmoid
Hàm loss	binary_crossentropy
Hàm optimizer	Adam
Metric để đánh giá	Accuracy

Epoch	150
Batch size	10

```
input_shape_train = X_new.shape[1] # Define the exact input_shape parameter

# define the keras model
model = Sequential()
model.add(Dense(12, input_shape=(9,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Kết quả khi train xong:

```
from tensorflow.keras.metrics import Accuracy
m = Accuracy()
m.update_state(y_test, y_pred)
m.result().numpy()
```

```
0.015
```

```
from tensorflow.keras.metrics import Recall
m = Recall()
m.update_state(y_test, y_pred)
m.result().numpy()
```

```
0.7606838
```

```
from tensorflow.keras.metrics import Precision
m = Precision()
m.update_state(y_test, y_pred)
m.result().numpy()
```

```
0.9468085
```