Identification of malwares from Windows PE files

Artificial Intelligence for Cybersecurity Project
AY 2024/2025

Matteo Giannini
Giacomo Lombardi

Identification of malwares from Windows PE files

Table of Contents

Goal Definition

Classification Problem

Developing a machine learning model capable of classifying Windows Portable executable (PE) files as malicious or bening based on static features extracted from the files themselves.

To achieve this objective, the dataset used consists of PE file characteristics extracted from a collection of Windows executable and DLL files. Each entry represents a unique file with various attributes extracted from its PE header and structure. The dataset includes both benign software samples and known malware samples.

Identification of malwares from Windows PE files

Table of Contents

Data Acquisition and Data Exploration

Dataset Features Description

```
df = pd.read_csv('./dataset/data_file.csv', sep=',')
df
```

	FileName	md5Hash	Machine	DebugSize	DebugRVA	MajorImageVersion	MajorOSVersion
0	0124e21d-018c-4ce0-92a3-b9e205a76bc0.dll	79755c51e413ed3c6be4635fd729a6e1	332	0	0	0	4
1	05c8318f98a5d301d80000009c316005.vertdll.dll	95e19f3657d34a432eada93221b0ea16	34404	84	121728	10	10
2	06054fba-5619-4a86-a861-ffb0464bef5d.dll	85c32641d77a54e19ba8ea4ab305c791	332	0	0	0	4
3	075822ac99a5d301660400009c316005.adhapi.dll	62e3b959d982ef534b66f819fe15f085	34404	84	19904	10	10
4	090607dd9ba5d301ca0900009c316005.SensorsNative...	ae38c5f7d313ad0ff3bfb8826476767f	34404	84	97728	10	10
...	...	...	...	...	...	...	...
62480	VirusShare_a43ceb5e5fffc793e0205d15a0606cb0	a43ceb5e5fffc793e0205d15a0606cb0	332	0	0	1	4
62481	VirusShare_0190dafc8304111a00fccf57340ea6a4	0190dafc8304111a00fccf57340ea6a4	332	0	0	7	10
62482	VirusShare_0f3ca55979aaf59158d6b01140696e44	0f3ca55979aaf59158d6b01140696e44	332	0	0	0	4
62483	VirusShare_fca5ce35f1690db6babca5aa5d559535	fca5ce35f1690db6babca5aa5d559535	332	0	0	0	4
62484	VirusShare_d7955a7e6d1e16800feafd4204cbac2b	d7955a7e6d1e16800feafd4204cbac2b	332	0	0	0	5

62485 rows × 18 columns

ExportRVA	ExportSize	IatVRA	MajorLinkerVersion	MinorLinkerVersion	NumberOfSections	SizeOfStackReserve	DllCharacteristics	ResourceSize	BitcoinAddresses	Benign
0	0	8192	8	0	3	1048576	34112	672	0	1
126576	4930	0	14	10	8	262144	16864	1024	0	1
0	0	8192	8	0	3	1048576	34112	672	0	1
21312	252	18160	14	10	6	262144	16736	1040	0	1
105792	1852	70592	14	10	7	262144	16736	1096	0	1
...	...	...	...	...	...	...	...	...	...	...
0	0	4096	6	0	3	1048576	0	23504	0	0
0	0	0	7	0	7	1048576	0	15704	0	0
0	0	404908	2	50	11	1048576	0	2364	0	0
14448	70	4096	8	0	4	1048576	0	130296	0	0
0	0	4096	9	0	6	1048576	0	6912	0	0

Data Acquisition and **Data Exploration**

Dataset Features Description

The dataset is composed of 17 features, plus the column named 'Benign' representing the class

	Column Name	Description
0	FileName	Name or identifier of the PE file
1	md5Hash	MD5 hash of the file for unique identification
2	Machine	Target machine architecture identifier
3	DebugSize	Size of debug information
4	DebugRVA	Relative Virtual Address of debug information
5	MajorImageVersion	Major version number of the image
6	MajorOSVersion	Major version number of required operating system
7	ExportRVA	Relative Virtual Address of export table
8	ExportSize	Size of export table
9	IatVRA	Relative Virtual Address of Import Address Table
10	MajorLinkerVersion	Major version number of linker
11	MinorLinkerVersion	Minor version number of linker
12	NumberOfSections	Number of sections in the PE file
13	SizeOfStackReserve	Size of stack to reserve
14	DllCharacteristics	DLL characteristics flags
15	ResourceSize	Size of resource section
16	BitcoinAddresses	Number of potential Bitcoin addresses found
17	Benign	Binary label (1 for benign, 0 for malicious)

Data Acquisition and **Data Exploration**

Dataset Features
Feature Types

```
df.info()
```

Almost all of the features except for ‘FileName’ and ‘md5Hash’ are numerical but from further analysis, based on domain knowledge, we can say that some of them are really numerical and others are categorical
After identifying numerical, categorical and binary variables we converted the categorical columns into the type ‘category’ and we left the numerical columns in the original type int64

```
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   FileName              62485 non-null  object
 1   md5Hash               62485 non-null  object
 2   Machine               62485 non-null  int64
 3   DebugSize             62485 non-null  int64
 4   DebugRVA              62485 non-null  int64
 5   MajorImageVersion     62485 non-null  int64
 6   MajorOSVersion        62485 non-null  int64
 7   ExportRVA             62485 non-null  int64
 8   ExportSize            62485 non-null  int64
 9   IatVRA                62485 non-null  int64
 10  MajorLinkerVersion    62485 non-null  int64
 11  MinorLinkerVersion    62485 non-null  int64
 12  NumberOfSections      62485 non-null  int64
 13  SizeOfStackReserve    62485 non-null  int64
 14  DllCharacteristics    62485 non-null  int64
 15  ResourceSize          62485 non-null  int64
 16  BitcoinAddresses      62485 non-null  int64
 17  Benign                62485 non-null  int64
```

Data Acquisition and **Data Exploration**

Dataset Features
Categorical and Numerical Features

Categorical Features

- Machine: CPU for which the file was designed (it has 6 unique values).
- MajorImageVersion: main version of the software.
- MajorOSVersion: main version number of the required operating system.
- MajorLinkerVersion: main version number of the linker that created the file.
- MinorLinkerVersion: secondary version number of the linker that created the file.
- DllCharacteristics: under which circumstances to initialize the DLL.
- NumberOfSections: number of sections the PE file should contain.

Numerical Features

- DebugSize: size of the debug information.
- DebugRVA: Relative Virtual Address (RVA) of the debug information.
- ExportRVA: RVA of the export table.
- ExportSize: size of the export table.
- IatVRA: RVA of the Import Address Table (IAT).
- SizeOfStackReserve: amount of memory to reserve for the initial thread stack.
- ResourceSize: size of the resources.

Data Acquisition and **Data Exploration**

Dataset Features
Null and Unique Values

```
df.isnull().sum()
```

```
FileName                0
md5Hash                 0
Machine                 0
DebugSize               0
DebugRVA                0
MajorImageVersion       0
MajorOSVersion          0
ExportRVA               0
ExportSize              0
IatVRA                  0
MajorLinkerVersion      0
MinorLinkerVersion      0
NumberOfSections        0
SizeOfStackReserve      0
DllCharacteristics      0
ResourceSize            0
BitcoinAddresses        0
Benign                  0
dtype: int64
```

No missing data

No need to manage them in preprocessing phase

```
df.nunique()
```

```
FileName    62485
md5Hash     62485
```

Unique values for each instance, so they do not provide useful information for the classification problem

```
df.drop(columns=['FileName', 'md5Hash'], inplace=True)
```

Data Acquisition and **Data Exploration**

Dataset Features

Distribution of Numerical Features

```
df[numeric_columns].describe()
```

	DebugRVA	DebugSize	ExportRVA	ExportSize	IatVRA	ResourceSize	SizeOfStackReserve
count	6.248500e+04	6.248500e+04	6.248500e+04	6.248500e+04	6.248500e+04	6.248500e+04	6.248500e+04
mean	1.541611e+05	2.587048e+04	8.953186e+05	4.094623e+05	1.466311e+05	1.844664e+05	8.759830e+05
std	1.903142e+06	6.461396e+06	3.779527e+07	2.851820e+07	1.124630e+06	1.732625e+07	6.288189e+05
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	4.096000e+03	1.080000e+03	2.621440e+05
50%	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	8.520000e+03	2.496000e+03	1.048576e+06
75%	1.283200e+04	2.800000e+01	2.875200e+04	1.040000e+02	6.553600e+04	2.350400e+04	1.048576e+06
max	2.852127e+08	1.615155e+09	2.147484e+09	2.415919e+09	6.615450e+07	4.294942e+09	1.677722e+07

Different ranges → Normalization necessary for Logistic Regression classifier

Data Acquisition and **Data Exploration**

Dataset Features

Importance of zeroes in numerical features

	DebugRVA	DebugSize	ExportRVA	ExportSize
count	6.248500e+04	6.248500e+04	6.248500e+04	6.248500e+04
mean	1.541611e+05	2.587048e+04	8.953186e+05	4.094623e+05
std	1.903142e+06	6.461396e+06	3.779527e+07	2.851820e+07
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
75%	1.283200e+04	2.800000e+01	2.875200e+04	1.040000e+02
max	2.852127e+08	1.615155e+09	2.147484e+09	2.415919e+09

Lots of 0s

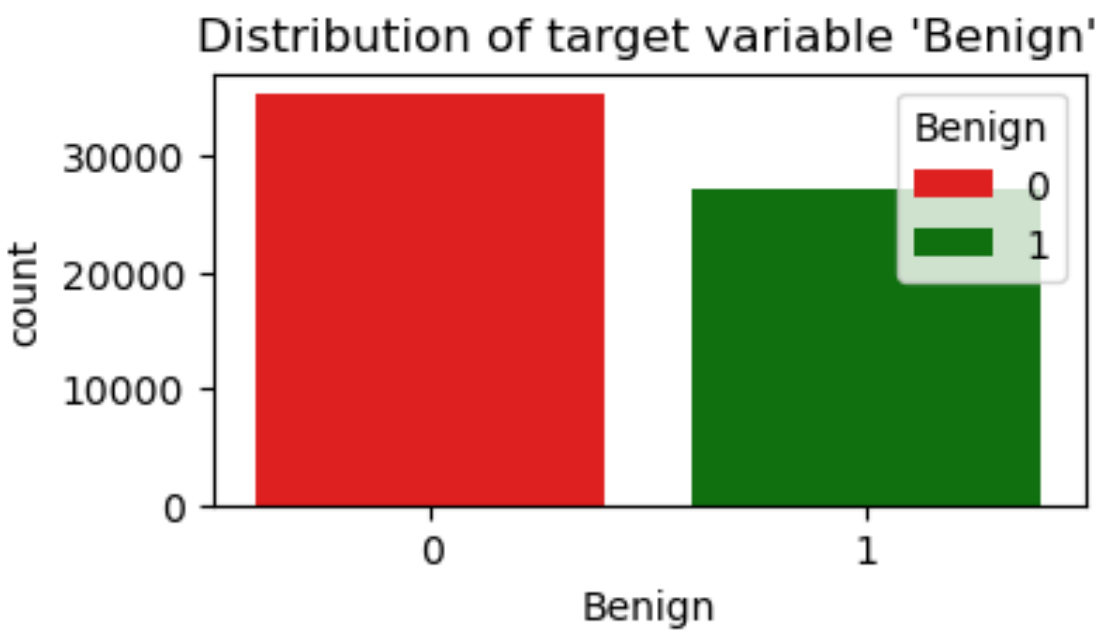
Significant for the classification

```
DebugRVA = 0 and DebugSize = 0
Number of malicious: 30828
Number of benign: 5676
DebugRVA != 0 and DebugSize = 0
Number of malicious: 17
Number of benign: 0
DebugRVA = 0 and DebugSize != 0
Number of malicious: 45
Number of benign: 0
DebugRVA != 0 and DebugSize != 0
Number of malicious: 4477
Number of benign: 21442
```

```
ExportRVA = 0 and ExportRVA = 0
Number of malicious: 29593
Number of benign: 12319
ExportRVA != 0 and ExportRVA = 0
Number of malicious: 122
Number of benign: 0
ExportRVA = 0 and ExportRVA != 0
Number of malicious: 96
Number of benign: 0
ExportRVA != 0 and ExportRVA != 0
Number of malicious: 5556
Number of benign: 14799
```

Variables Plot

Distribution of target variable



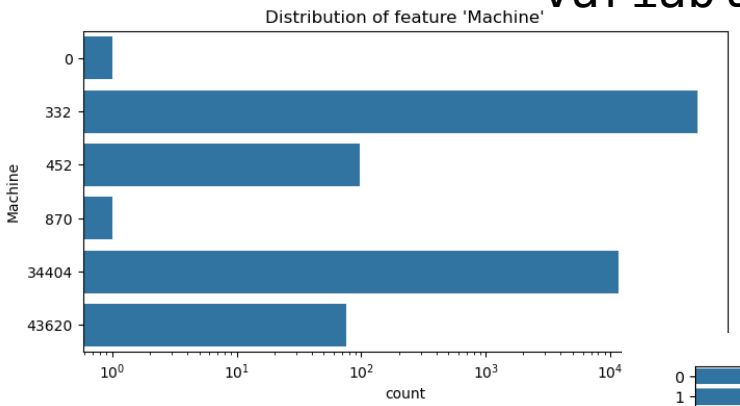
Balanced Dataset

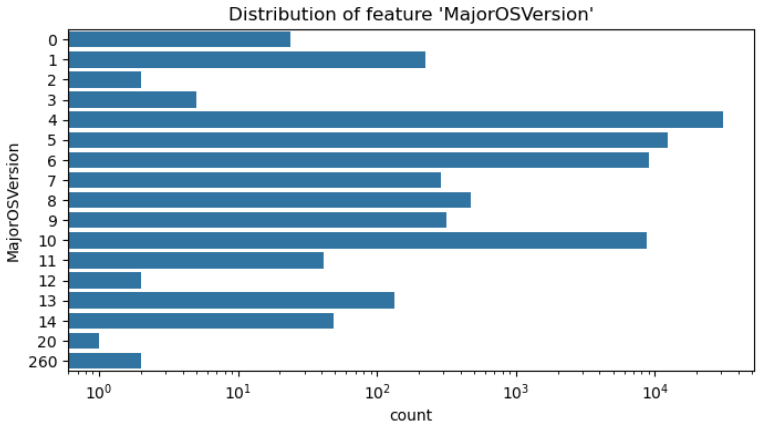
Unusual for this type of dataset

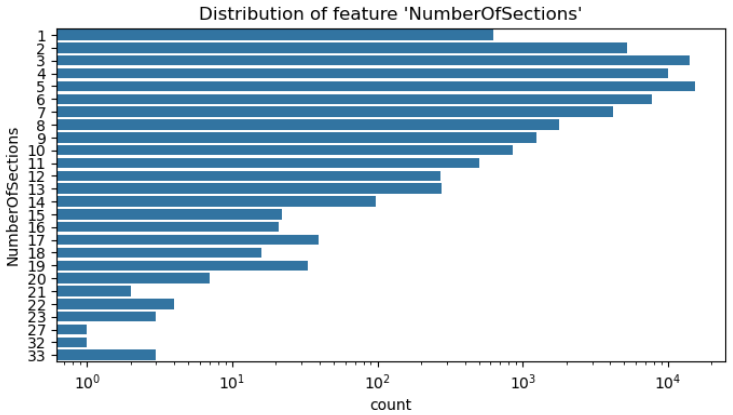
Data Acquisition and **Data Exploration**

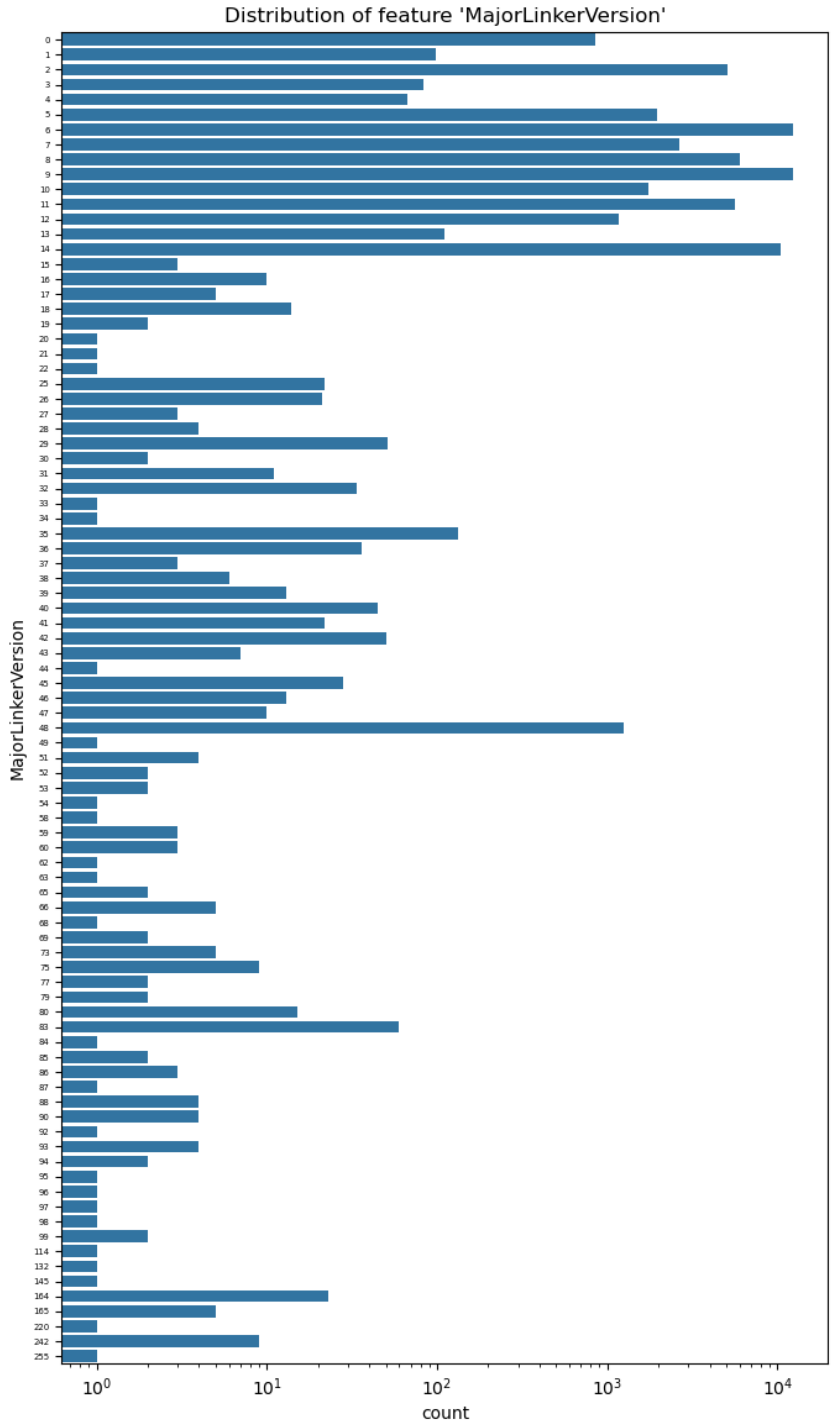
Variables Plot

Distribution of categorical variables





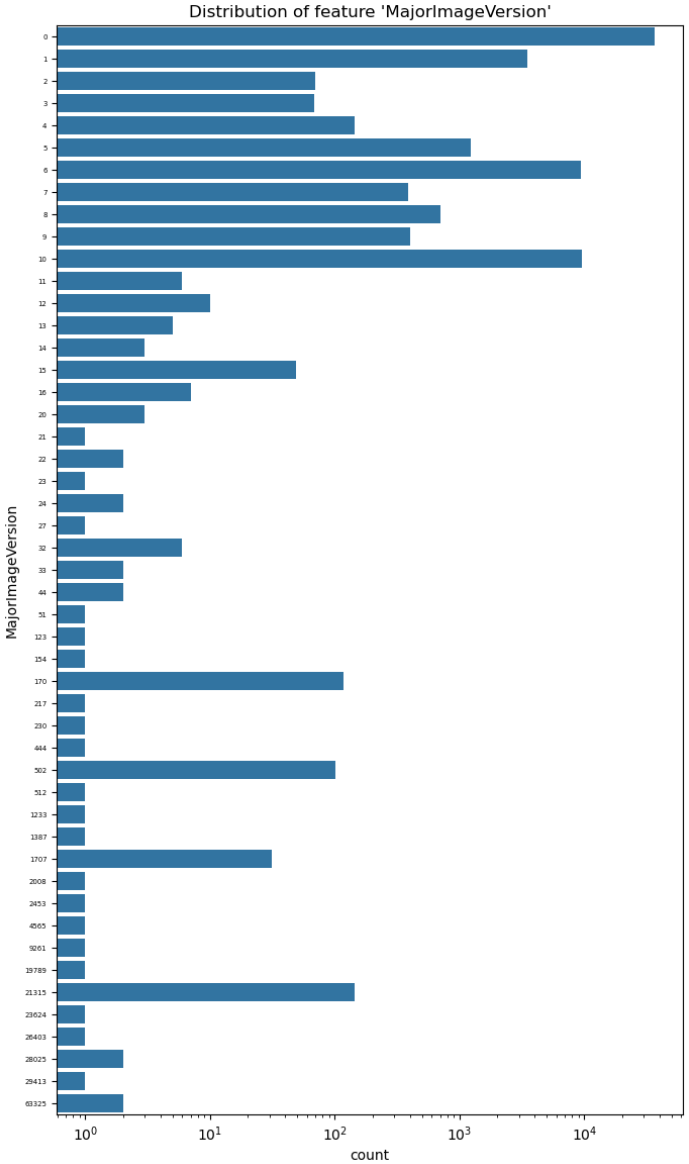


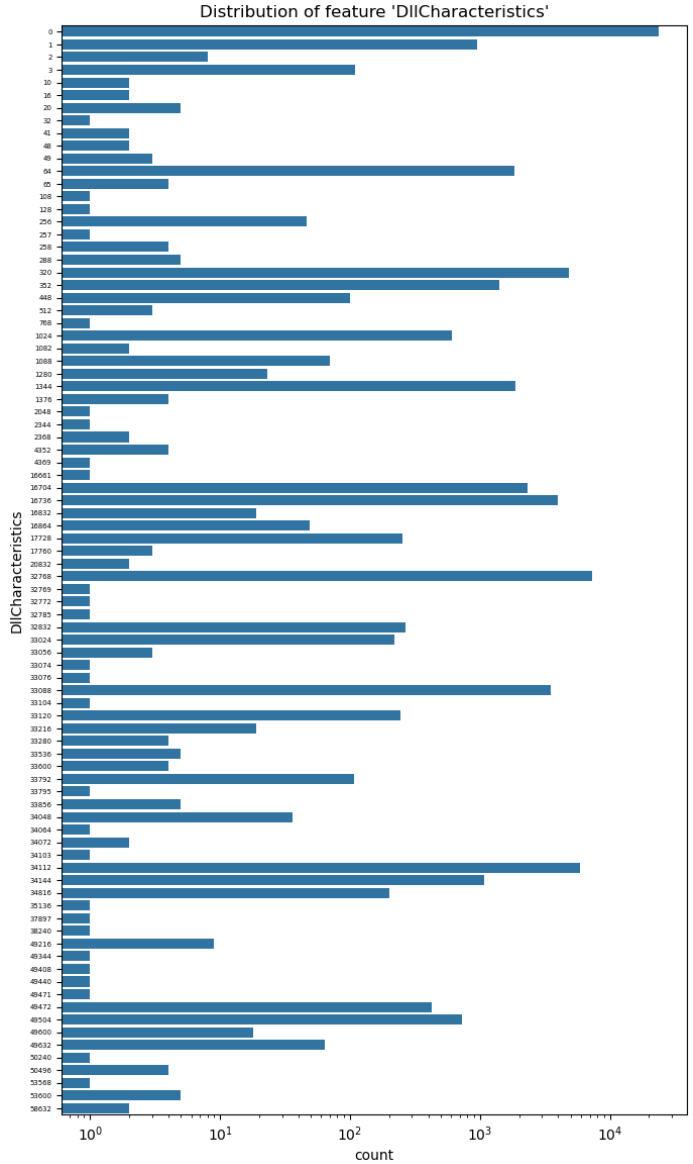


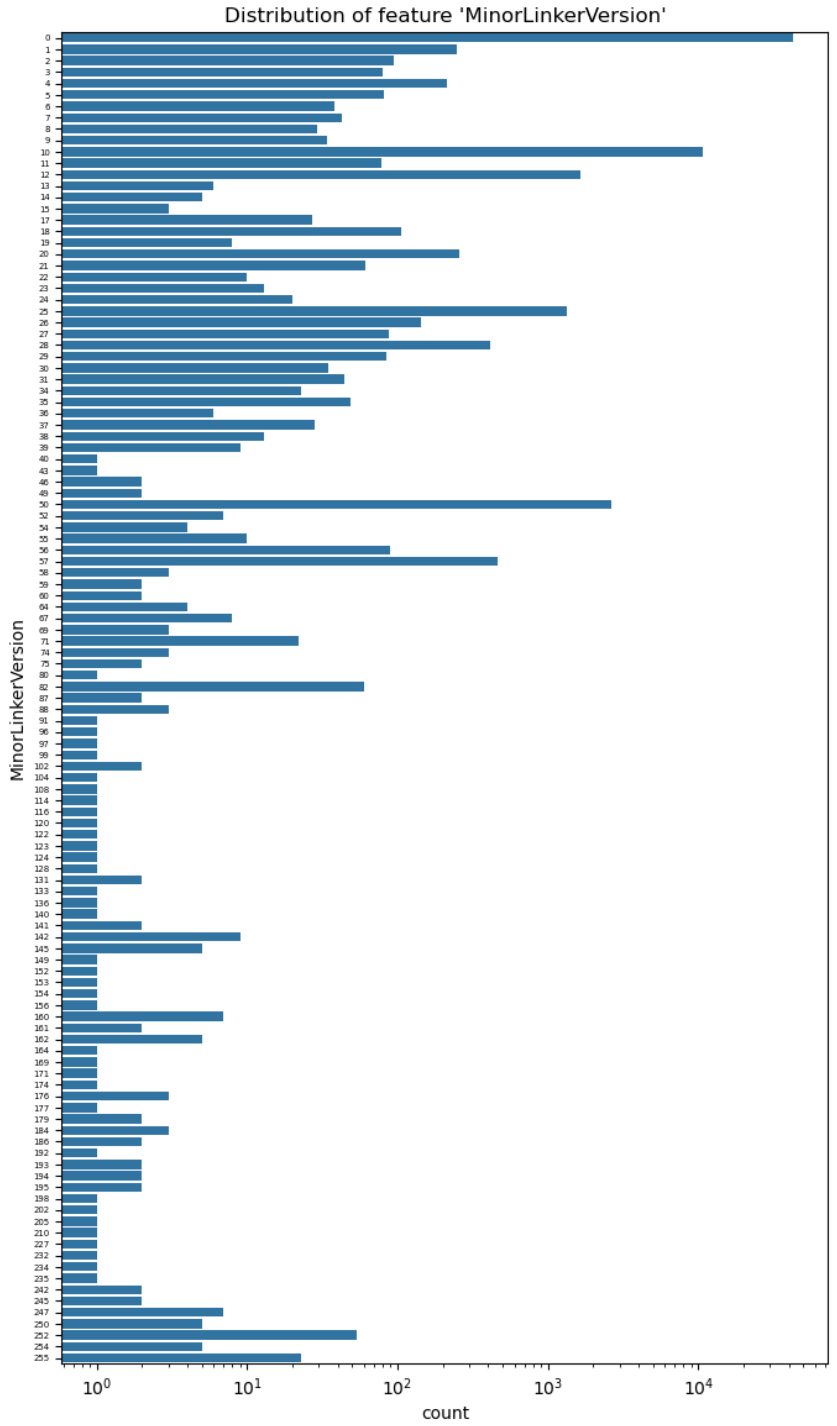
Variables Plot

Distribution of categorical variables

Data Acquisition and **Data Exploration**



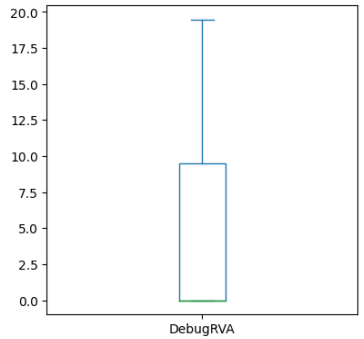


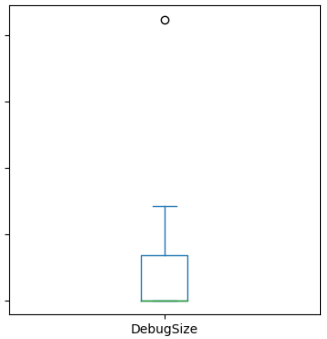


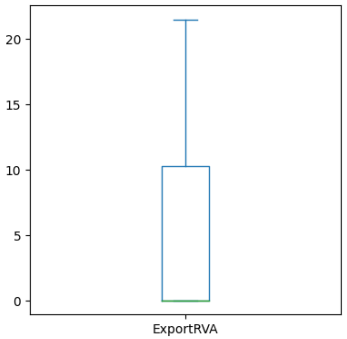
Data Acquisition and **Data Exploration**

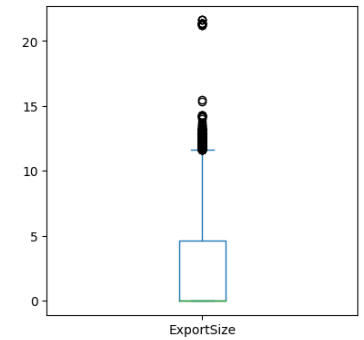
Variables Plot

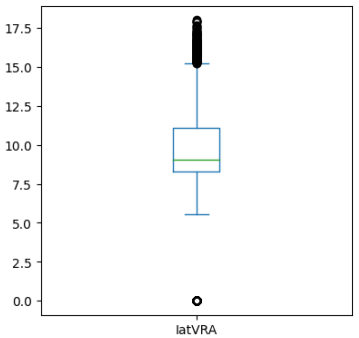
Numerical Variables Box Plots

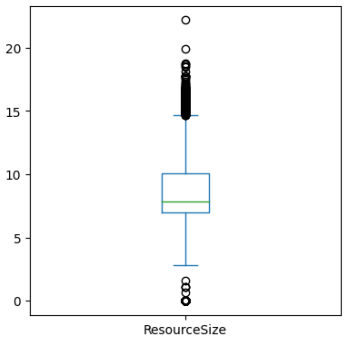


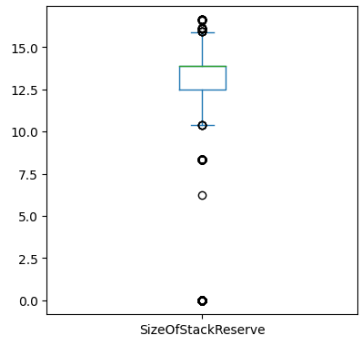


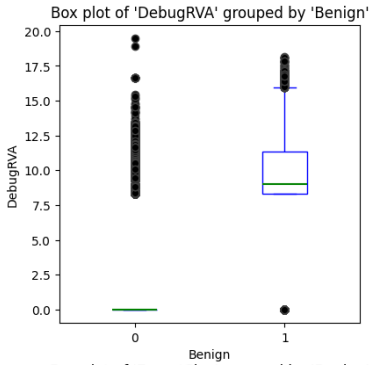


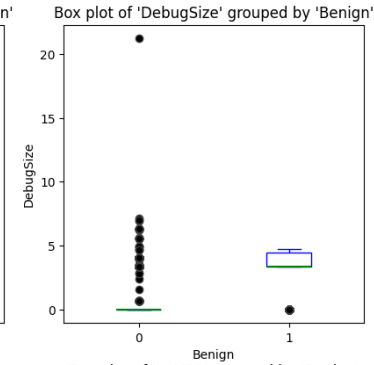


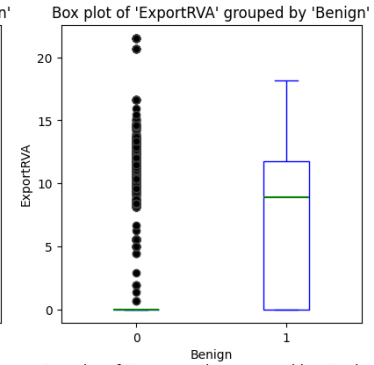


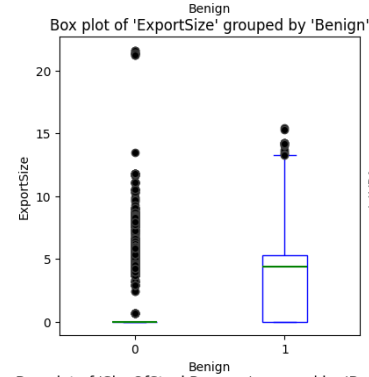


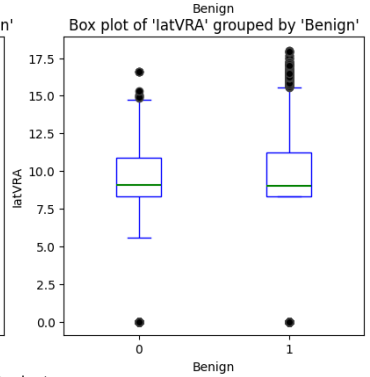


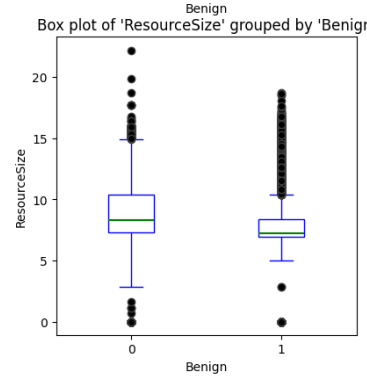


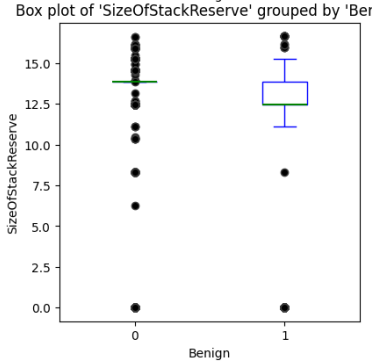






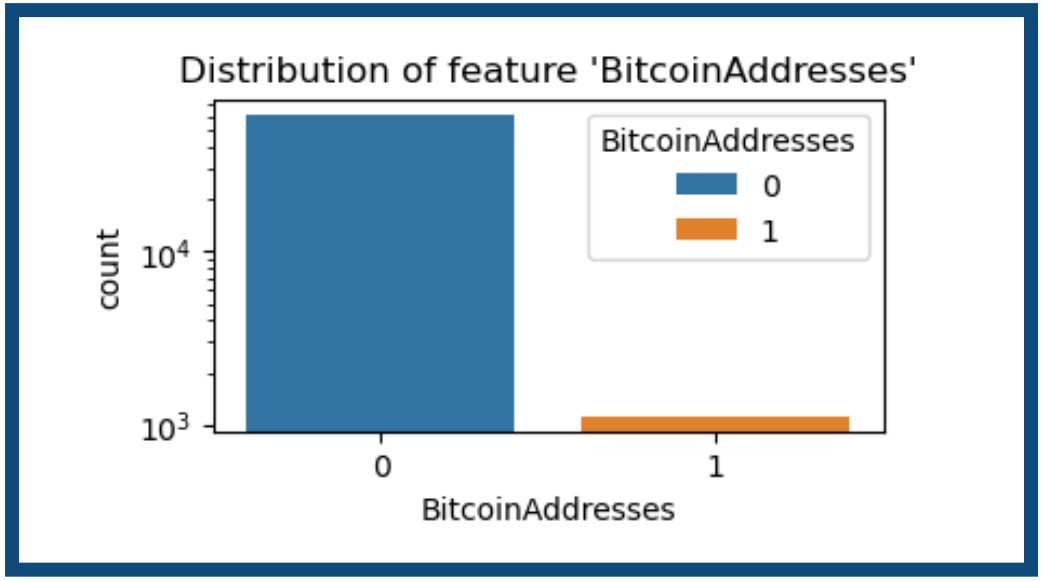






Data Acquisition and **Data Exploration**

Variables Plot
Distribution of BitcoinAddress




```
df_filtered = df.loc[(df['BitcoinAddresses'] == 1)]
count_malicious = (df_filtered['Benign'] == 0).sum()
count_benign = (df_filtered['Benign'] == 1).sum()
print(f"Number of malicious: {count_malicious}")
print(f"Number of benign: {count_benign}")
```

```
Number of malicious: 411
Number of benign: 714
```

Lots of 0s

+

No relevance of 1s

=

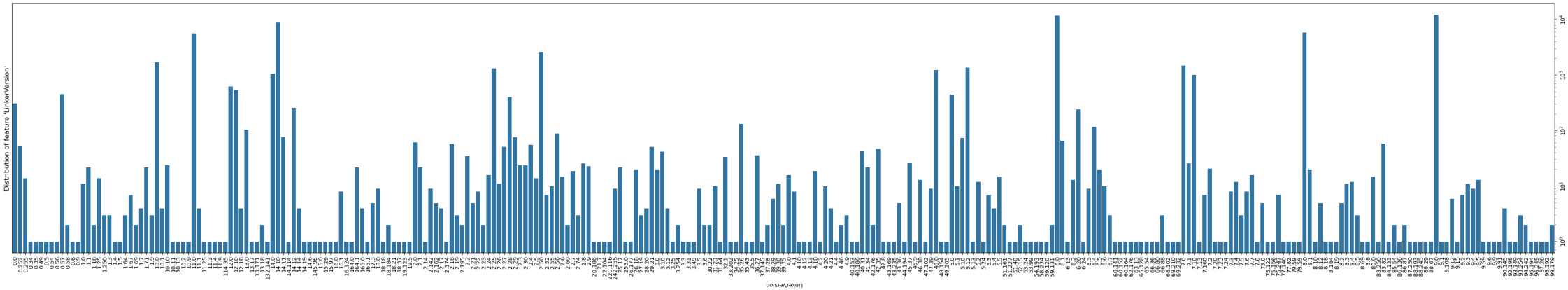
Feature removal

Dataset Feaures
Merging ‘MajorLinkerVersion’ and ‘MinorLinkerVersion’

Data Acquisition and **Data Exploration**

Considering the meaning of 'MajorLinkerVersion' and 'MinorLinkerVersion' we can think of merging them in a single variable called 'LinkerVersion'.

```
df['LinkerVersion'] = df['MajorLinkerVersion'].astype(str) + '.' + df['MinorLinkerVersion'].astype(str)
df['LinkerVersion'] = df['LinkerVersion'].astype('category')
df.drop(columns=['MajorLinkerVersion', 'MinorLinkerVersion'], inplace=True)

categorical_columns = [col for col in categorical_columns if col not in ['MajorLinkerVersion', 'MinorLinkerVersion']]
categorical_columns.append('LinkerVersion')
```



Data Acquisition and Data Exploration

Variables Correlation

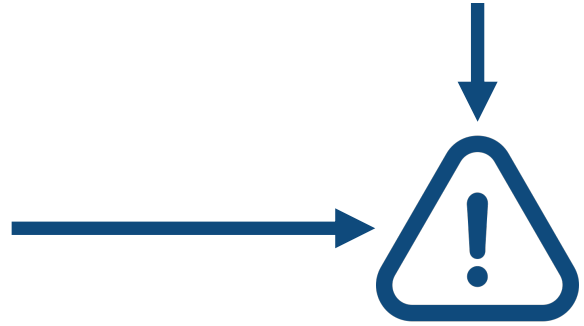
Categorical Variables Correlation with Chi2 Test

	Column1	Column2	Chi2	p-value
0	Machine	MajorImageVersion	16403.854910	0.0
1	Machine	MajorOSVersion	22934.330214	0.0
2	Machine	DllCharacteristics	44236.191923	0.0
3	Machine	NumberOfSections	12751.872957	0.0
4	Machine	LinkerVersion	81682.809445	0.0
5	MajorImageVersion	MajorOSVersion	94575.916371	0.0
6	MajorImageVersion	DllCharacteristics	178329.858421	0.0
7	MajorImageVersion	NumberOfSections	34414.385304	0.0
8	MajorImageVersion	LinkerVersion	303350.199794	0.0
9	MajorOSVersion	DllCharacteristics	104018.228449	0.0
10	MajorOSVersion	NumberOfSections	41212.790118	0.0
11	MajorOSVersion	LinkerVersion	481109.786324	0.0
12	DllCharacteristics	NumberOfSections	78955.413686	0.0
13	DllCharacteristics	LinkerVersion	388538.747385	0.0
14	NumberOfSections	LinkerVersion	168209.568483	0.0

```
df[categorical_columns].nunique()
```

✓ 0.0s

Machine	6
MajorImageVersion	49
MajorOSVersion	17
DllCharacteristics	86
NumberOfSections	26
LinkerVersion	293



No statistical relevance: All p-values are equal to 0 due to the high number of categories.

Data Acquisition and **Data Exploration**

Variables Correlation

Categorical Variables Correlation with Cramer's V Index

Variable 1	Variable 2	Cramér's V
Machine	MajorImageVersion	0.22914
Machine	MajorOSVersion	0.27094
Machine	DllCharacteristics	0.37628
Machine	NumberOfSections	0.20203
Machine	LinkerVersion	0.51132
MajorImageVersion	MajorOSVersion	0.30757
MajorImageVersion	DllCharacteristics	0.24384
MajorImageVersion	NumberOfSections	0.14843
MajorImageVersion	LinkerVersion	0.31803
MajorOSVersion	DllCharacteristics	0.32256
MajorOSVersion	NumberOfSections	0.20303
MajorOSVersion	LinkerVersion	0.69370
DllCharacteristics	NumberOfSections	0.22482
DllCharacteristics	LinkerVersion	0.27047
NumberOfSections	LinkerVersion	0.32815

Cramér's V is a statistical measure used to assess the strength of association between two categorical variables. It is based on the chi-squared statistic and provides a value between 0 and 1, where 0 indicates no association and 1 indicates a strong association. Unlike the chi-squared test alone, Cramér's V is normalized, making it easier to compare relationships across variables with different sizes or category counts.

While there's no universal scale, common interpretations of Cramér's V values are:

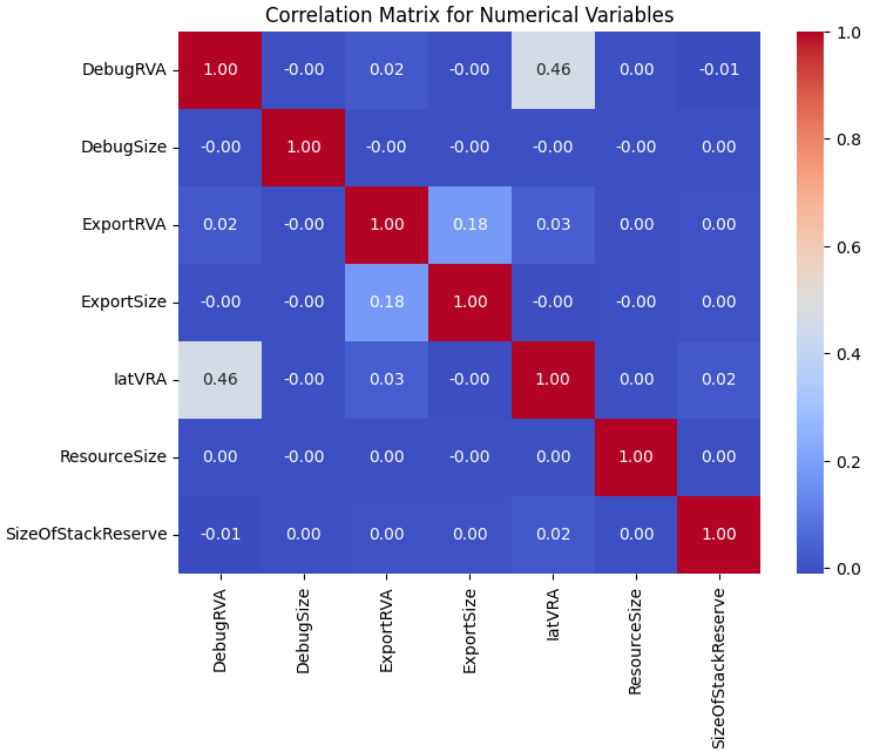
[0,0.10] → Very weak association
[0.10,0.30] → Weak association
[0.30,0.50] → Moderate association
>0.50 → Strong association

In our case we can see that most of the variables are weakly correlated

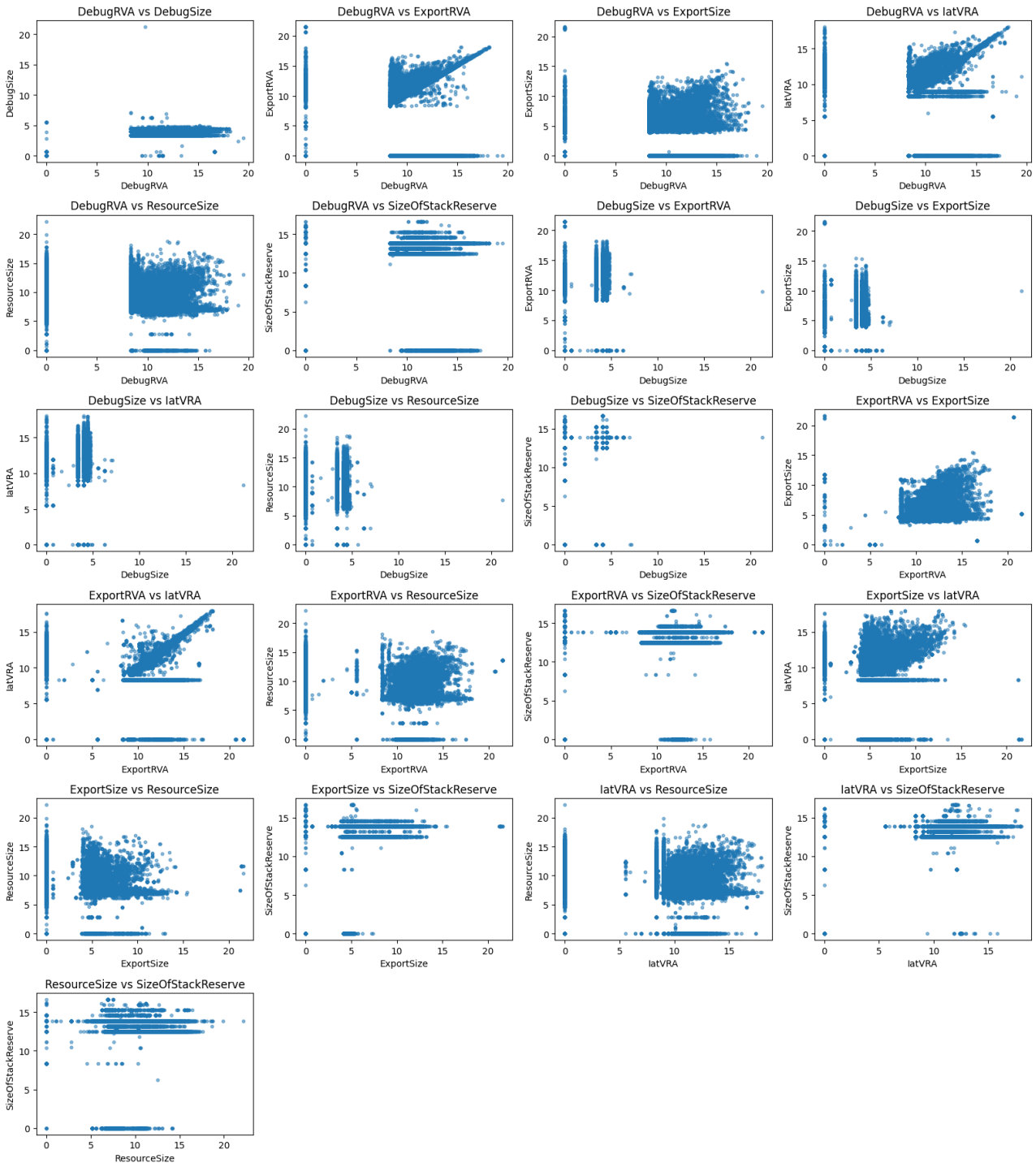
Variables Correlation

Numerical Variables Correlation with Scatter Plots and Correlation Matrix

Data Acquisition and **Data Exploration**

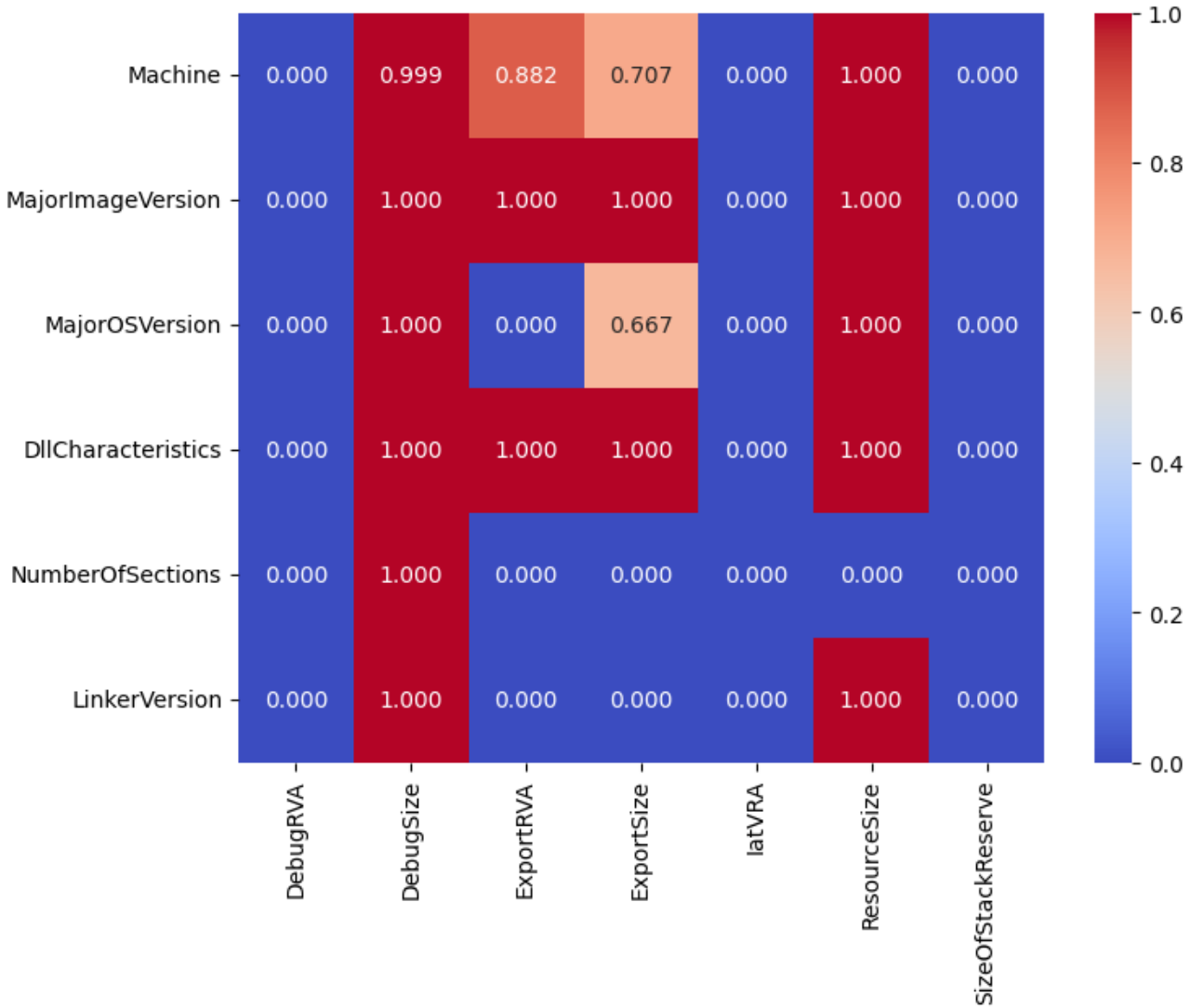


Given the correlation matrix and the scatter plots, we can see that the numerical variables are mostly uncorrelated with the exception of IatVRA and DebugRVA



Variables Correlation

Numerical and Categorical Variables Correlation with ANOVA Test



Given the results of the correlation matrix drawn with the ANOVA test we can see that most of the categorical variables are correlated with numerical variables. For this reason, in the following processing phase we will try to classify malwares using numerical and categorical variables separately

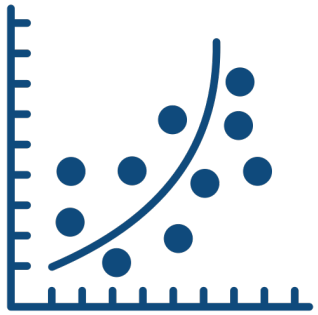
Identification of malwares from Windows PE files

Table of Contents

Data Preprocessing

Needed Preprocessing

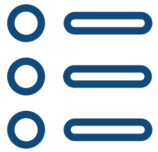
Due to the correlation seen between numerical and categorical variables from the ANOVA test we decided to conduct three types of experiments with two different classifiers with data implying three different types of preprocessing:



Use of Logistic Regression

Use of Numerical Features

StandardScaler



Use of Categorical Features

One-Hot Encoding

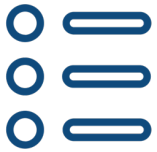
Use of Whole Dataset

One-Hot Encoding
StandardScaler
SelectFromModel

Use of Random Forest

Use of Numerical Features

No Preprocessing Needed



Use of Categorical Features

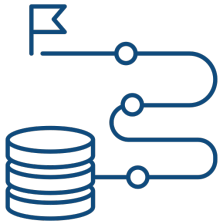
One-Hot Encoding

Use of Whole Dataset

One-Hot Encoding of Categorical Variables

Data Preprocessing

Use of the Pipeline



We adopted the use of a Pipeline to prevent data leakage, ensuring that all preprocessing steps are fitted only on the training set and then applied to the test set. In fact, test data remains completely unseen until prediction.

Our first step is to analyze the various preprocessing techniques used across the different pipelines.

Data Preprocessing

StandardScaler

From our exploration of numeric features, we have observed that they have different ranges, which could pose a problem for the Logistic Regression classifier.

We normalize numeric features using Standard Scaler in the experiments involving Logistic Regression that utilize numeric features.

OneHotEncoder

Since our categorical features do not have a meaningful order, we encode them with One Hot Encoding in all experiments where they're used.
Even though decision trees usually handle directly categorical variables, the scikit variant requires the encoding with some type of encoder such as One-Hot, hence the decision of using this type of encoding even when Random Forest is used.

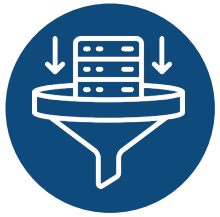
Feature Selection: SelectFromModel

We used SelectFromModel in experiments with Logistic Regression when using the entire dataset, as it automatically selects the most important features based on the model's feature importance.

It works by fitting a model that assigns importance scores to each feature—such as coefficients in linear models or feature importances in tree-based models—and then selects features whose importance exceeds a given threshold.

Identification of malwares from Windows PE files

Table of Contents



Data Processing

Choice of CrossValidation

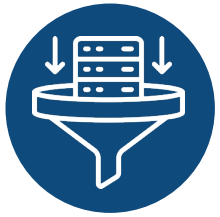
For each of our data processing we used cross validation. This method is used for three main reasons:

- **Prevents overfitting:** It helps detect if the model is too closely fitting the training data and not generalizing well.

- **More reliable evaluation:** Instead of testing the model on a single train-test split this methods permits to average the results over several splits for a more stable estimate.

- **Better use of data:** All data gets used for both training and testing at different points.

In our case we use k-fold cross-validation, splitting data into k (in our case 10) folds, and training the model on k-1 folds and testing it on the remaining one, repeating this process k times.

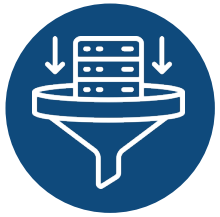


Data Processing

Performance evaluation

To evaluate performances and compare the models we used different metrics:

- **Accuracy:** we use accuracy as a general performance measure, giving us a general idea of how well the model is performing across both benign and malware classes. Since the dataset is balanced, accuracy gives interesting results.
- **Confusion Matrix:** to gain deeper insights into classification performance, we also examined the confusion matrices for each fold. This analysis allows us to observe both correct and incorrect classifications for each class.
- **Precision & Recall:** to summarize the results and compare models effectively, we focused on precision and recall. Precision is crucial because a false positive (where malware is incorrectly classified as benign) can lead to serious risks such as undetected malicious software. Recall is important to make sure that the model correctly identifies all instances of benign software, preventing unnecessary false alarms.
- **F-measure:** to compare the overall performance of different models, we also computed the F-measure, which provides a single score that balances precision and recall. This metric is particularly useful when both false positives and false negatives carry significant consequences. However, since our dataset is balanced, accuracy could be a good metric to compare models.

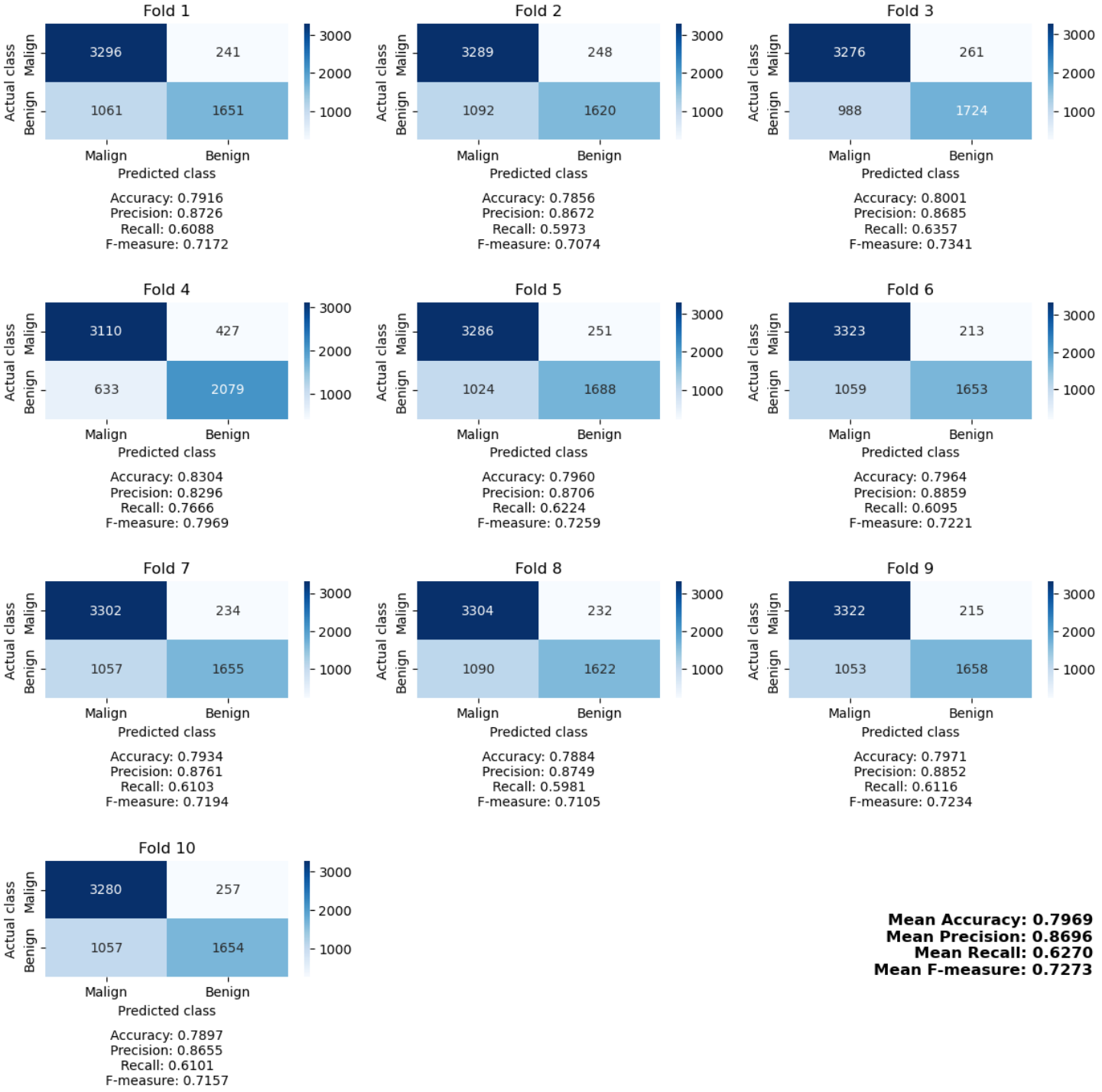


1st Experiment: Logistic Regression with Numeric Features

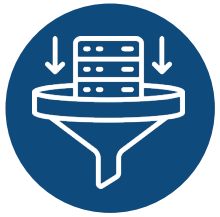
Data Processing

```
X = df[numeric_columns]
y = df['Benign']

preprocessor = ColumnTransformer(
    transformers=[
        ('num_scaler', StandardScaler(), numeric_columns)
    ]
)

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])
```

As already said, since the numerical features have different scales, we apply a pre-processing phase in which we normalize them using Standard Scaler.



Mean Accuracy: 0.7969
Mean Precision: 0.8696
Mean Recall: 0.6270
Mean F-measure: 0.7273

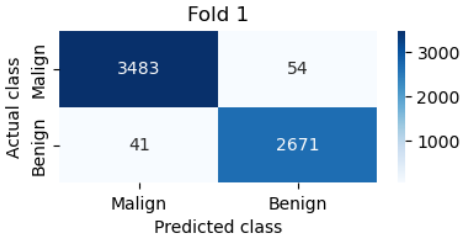


2nd Experiment: Logistic Regression with Categorical Features

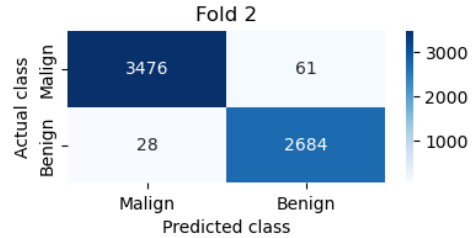
Data Processing

```
X = df[categorical_columns]
y = df['Benign']

preprocessor = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'), categorical_columns)
    ]
)

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000))
])
```

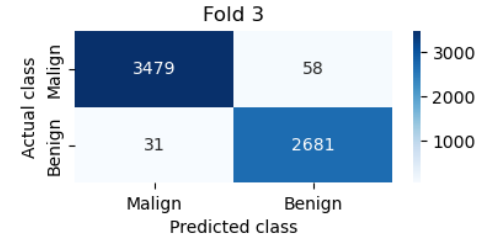
Before training the model, it is necessary to apply preprocessing to encode the categorical features. To do so, we use One Hot Encoder.



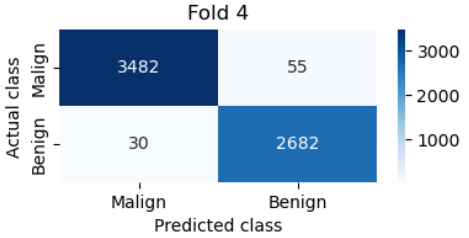
Accuracy: 0.9848
Precision: 0.9802
Recall: 0.9849
F-measure: 0.9825



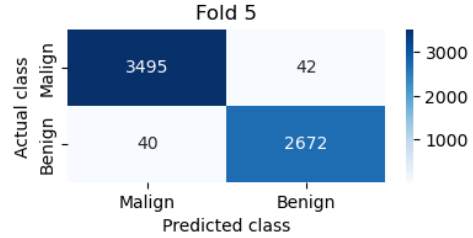
Accuracy: 0.9858
Precision: 0.9778
Recall: 0.9897
F-measure: 0.9837



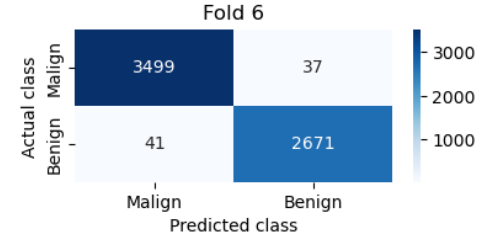
Accuracy: 0.9858
Precision: 0.9788
Recall: 0.9886
F-measure: 0.9837



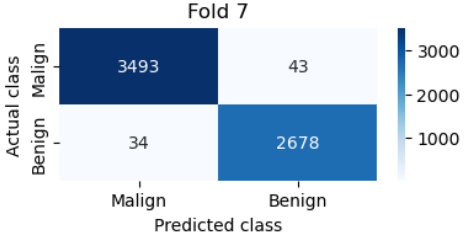
Accuracy: 0.9864
Precision: 0.9799
Recall: 0.9889
F-measure: 0.9844



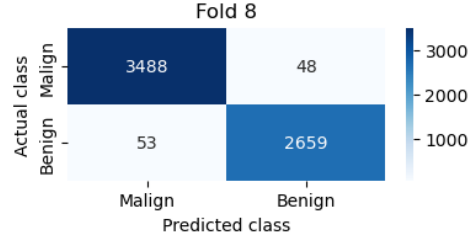
Accuracy: 0.9869
Precision: 0.9845
Recall: 0.9853
F-measure: 0.9849



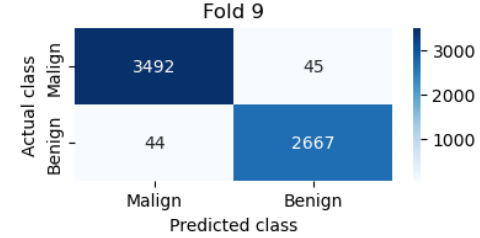
Accuracy: 0.9875
Precision: 0.9863
Recall: 0.9849
F-measure: 0.9856



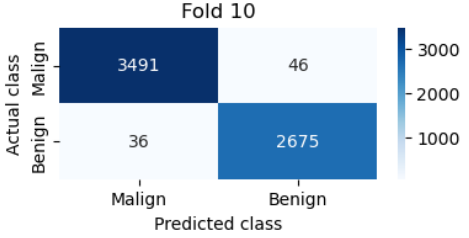
Accuracy: 0.9877
Precision: 0.9842
Recall: 0.9875
F-measure: 0.9858



Accuracy: 0.9838
Precision: 0.9823
Recall: 0.9805
F-measure: 0.9814



Accuracy: 0.9858
Precision: 0.9834
Recall: 0.9838
F-measure: 0.9836



Accuracy: 0.9869
Precision: 0.9831
Recall: 0.9867
F-measure: 0.9849

Mean Accuracy: 0.9861
Mean Precision: 0.9821
Mean Recall: 0.9861
Mean F-measure: 0.9840

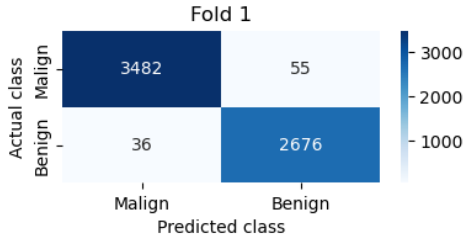
3rd Experiment: Logistic Regression with Whole Dataset

Data Processing

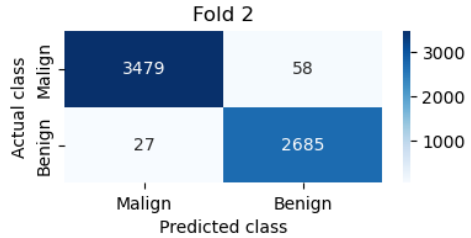
```
X = df.drop(columns=['Benign'])
y = df['Benign']

preprocessor = ColumnTransformer(
    transformers=[
        ('num_transform', StandardScaler(), numeric_columns),
        ('cat_transform', OneHotEncoder(handle_unknown='ignore'), categorical_columns)
    ])

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('feature_selection', SelectFromModel(estimator=LogisticRegression(max_iter=1000))),
    ('classifier', LogisticRegression(max_iter=1000))
])
```

In this experiment, we implemented a structured pre-processing phase consisting of the following steps:

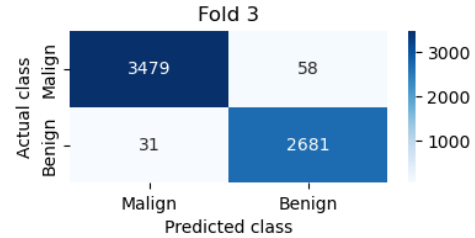
- Normalization of numerical features.
- Encoding of categorical features to make them suitable for the model.
- Feature selection to reduce redundancy and enhance accuracy.



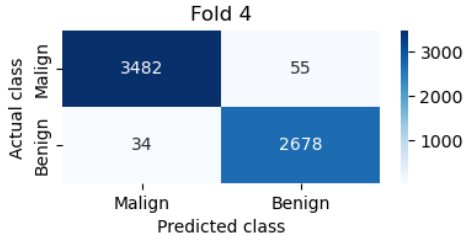
Accuracy: 0.9854
Precision: 0.9799
Recall: 0.9867
F-measure: 0.9833



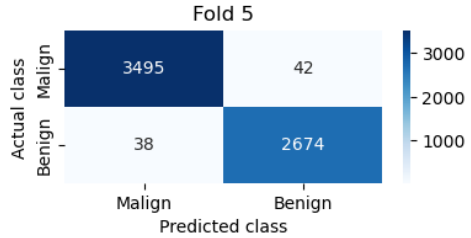
Accuracy: 0.9864
Precision: 0.9789
Recall: 0.9900
F-measure: 0.9844



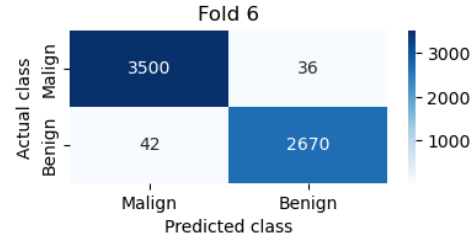
Accuracy: 0.9858
Precision: 0.9788
Recall: 0.9886
F-measure: 0.9837



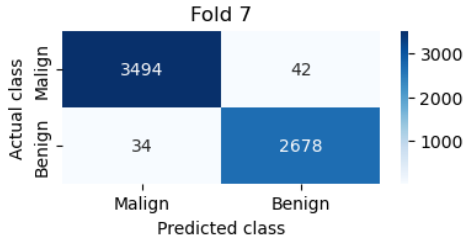
Accuracy: 0.9858
Precision: 0.9799
Recall: 0.9875
F-measure: 0.9837



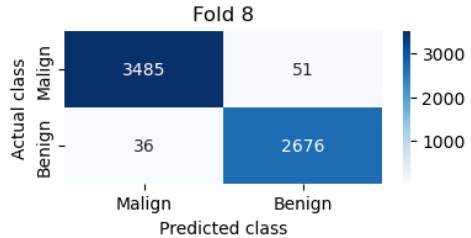
Accuracy: 0.9872
Precision: 0.9845
Recall: 0.9860
F-measure: 0.9853



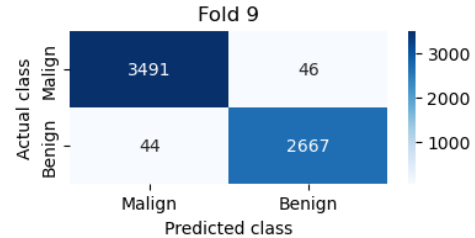
Accuracy: 0.9875
Precision: 0.9867
Recall: 0.9845
F-measure: 0.9856



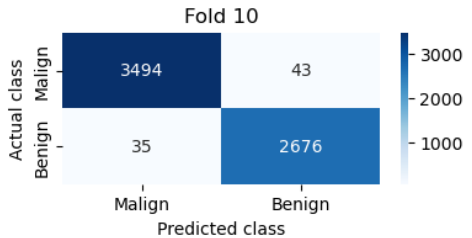
Accuracy: 0.9878
Precision: 0.9846
Recall: 0.9875
F-measure: 0.9860



Accuracy: 0.9861
Precision: 0.9813
Recall: 0.9867
F-measure: 0.9840



Accuracy: 0.9856
Precision: 0.9830
Recall: 0.9838
F-measure: 0.9834



Accuracy: 0.9875
Precision: 0.9842
Recall: 0.9871
F-measure: 0.9856

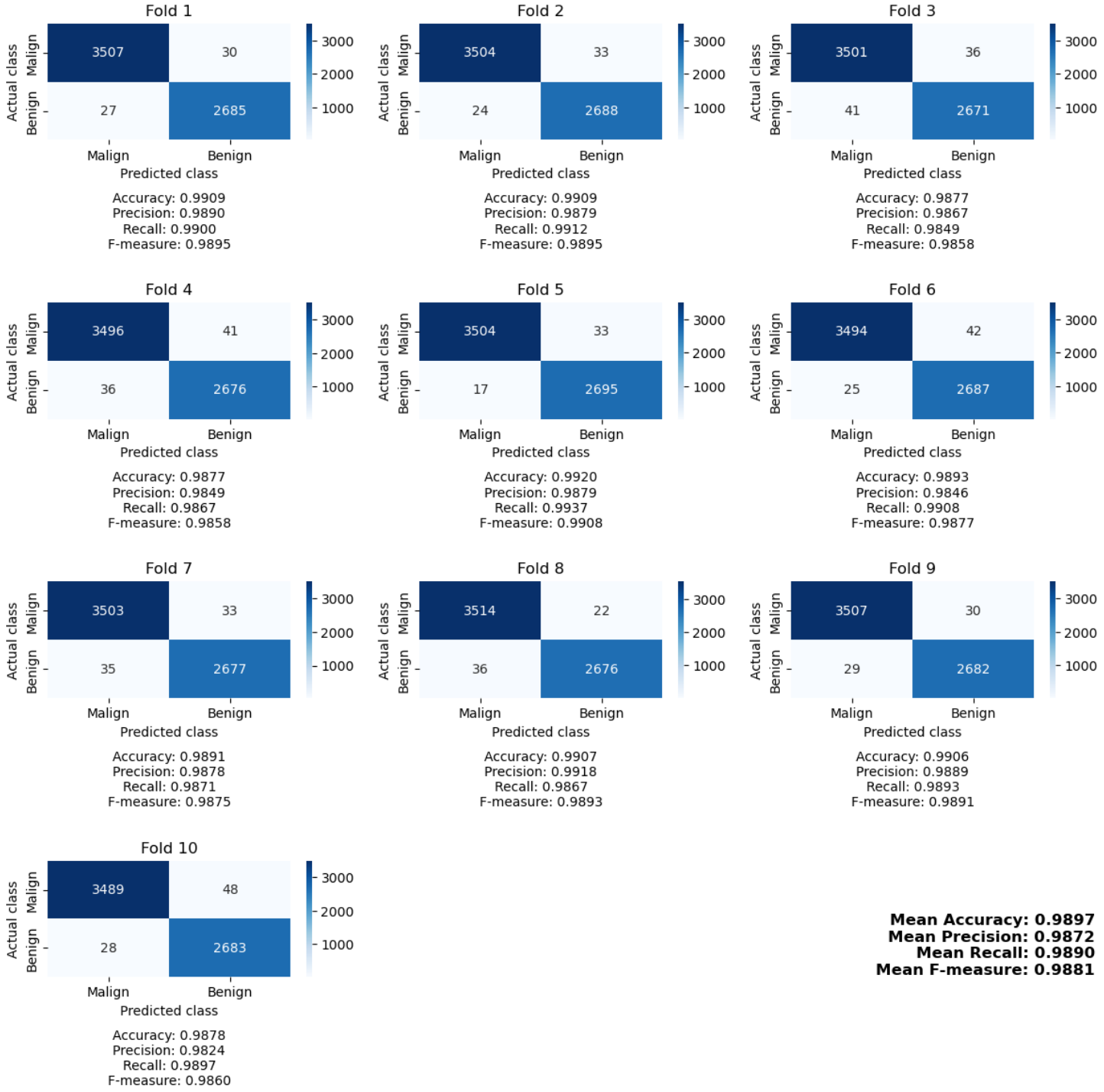
Mean Accuracy: 0.9865
Mean Precision: 0.9822
Mean Recall: 0.9868
Mean F-measure: 0.9845

4th Experiment: Random Forest with Numeric Features

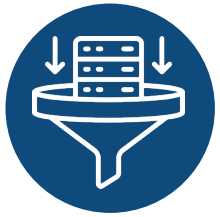
Data Processing

```
X = df[numeric_columns]
y = df['Benign']

pipeline = Pipeline(steps=[
    ('classifier', RandomForestClassifier())
])
```

We do not scale variables because Random Forest is based on decision trees, which are scale invariant.



Mean Accuracy: 0.9897
Mean Precision: 0.9872
Mean Recall: 0.9890
Mean F-measure: 0.9881

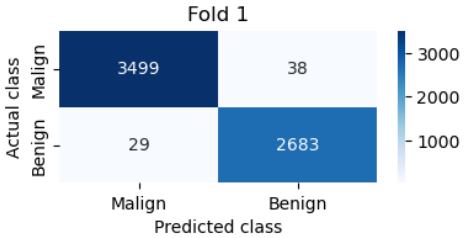


5th Experiment: Random Forest with Categorical Features

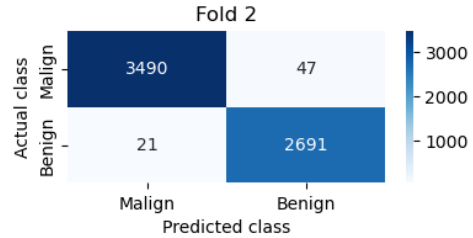
Data Processing

```
X = df[categorical_columns]
y = df['Benign']

preprocessor = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'), categorical_columns)
    ]
)

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier())
])
```

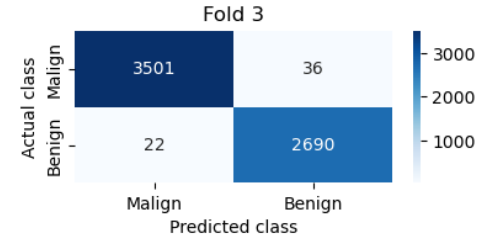
Since Random Forest of scikit-learn does not manage categorical variables, we first encode them using One Hot Encoding.



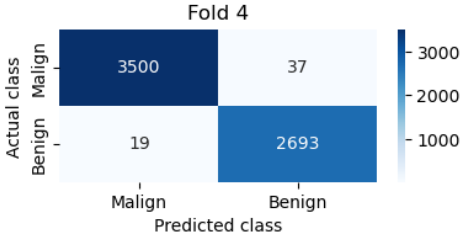
Accuracy: 0.9893
Precision: 0.9860
Recall: 0.9893
F-measure: 0.9877



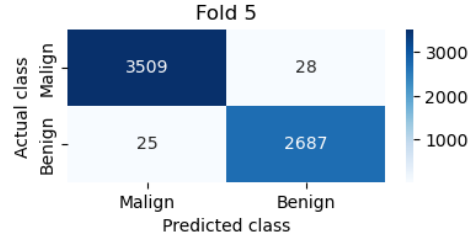
Accuracy: 0.9891
Precision: 0.9828
Recall: 0.9923
F-measure: 0.9875



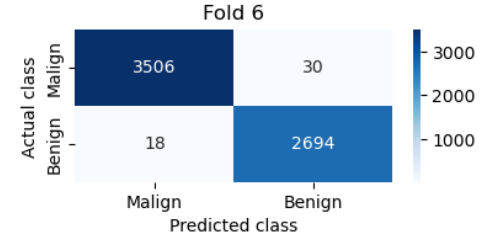
Accuracy: 0.9907
Precision: 0.9868
Recall: 0.9919
F-measure: 0.9893



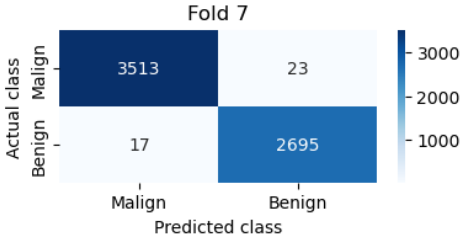
Accuracy: 0.9910
Precision: 0.9864
Recall: 0.9930
F-measure: 0.9897



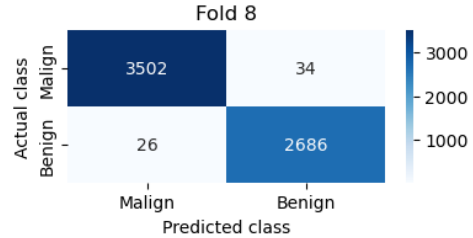
Accuracy: 0.9915
Precision: 0.9897
Recall: 0.9908
F-measure: 0.9902



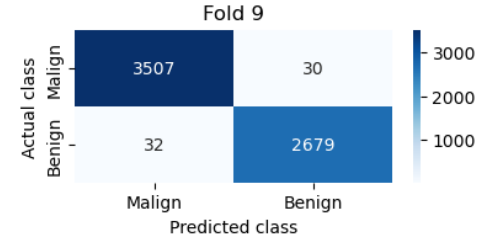
Accuracy: 0.9923
Precision: 0.9890
Recall: 0.9934
F-measure: 0.9912



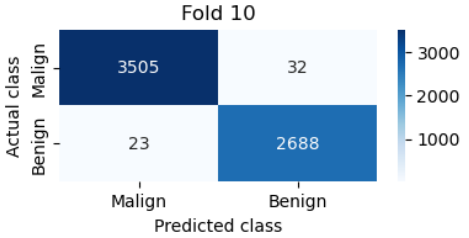
Accuracy: 0.9936
Precision: 0.9915
Recall: 0.9937
F-measure: 0.9926



Accuracy: 0.9904
Precision: 0.9875
Recall: 0.9904
F-measure: 0.9890



Accuracy: 0.9901
Precision: 0.9889
Recall: 0.9882
F-measure: 0.9886



Accuracy: 0.9912
Precision: 0.9882
Recall: 0.9915
F-measure: 0.9899

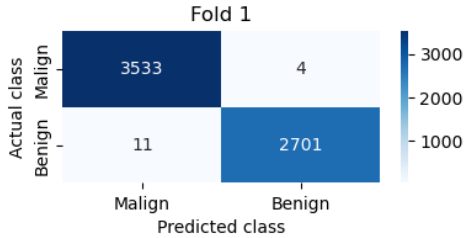
Mean Accuracy: 0.9909
Mean Precision: 0.9877
Mean Recall: 0.9914
Mean F-measure: 0.9896

6th Experiment: Random Forest with Whole Dataset

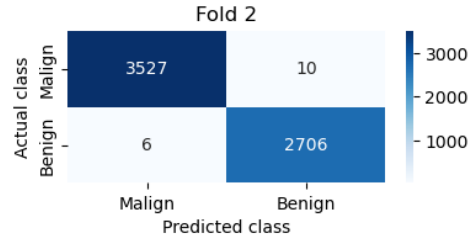
Data Processing

```
X = df.drop(columns=['Benign'])
y = df['Benign']

preprocessor = ColumnTransformer(
    transformers=[
        ('cat_transform', OneHotEncoder(handle_unknown='ignore'), categorical_columns)],
        remainder='passthrough'     # necessary otherwise numerical columns are discarded
    )

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier())
])
```

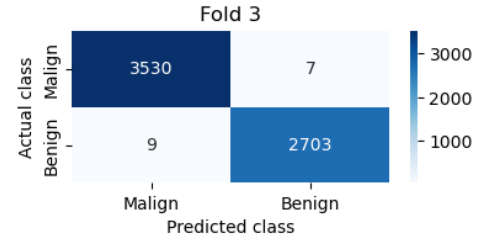
- No need of scaling numeric features.
- As always, we encode categorical variables with One Hot Encoding.
- Feature selection is not necessary because Random Forest inherently handles it during the tree construction process.



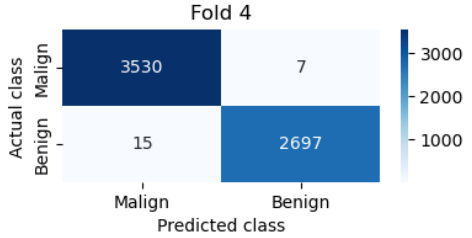
Accuracy: 0.9976
Precision: 0.9985
Recall: 0.9959
F-measure: 0.9972



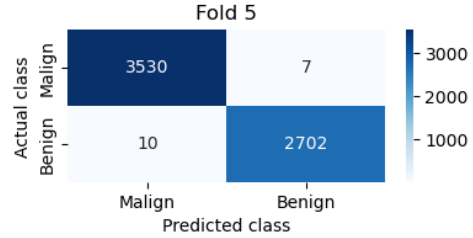
Accuracy: 0.9974
Precision: 0.9963
Recall: 0.9978
F-measure: 0.9971



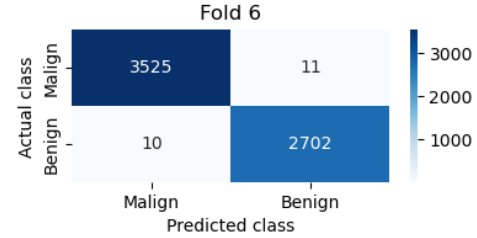
Accuracy: 0.9974
Precision: 0.9974
Recall: 0.9967
F-measure: 0.9970



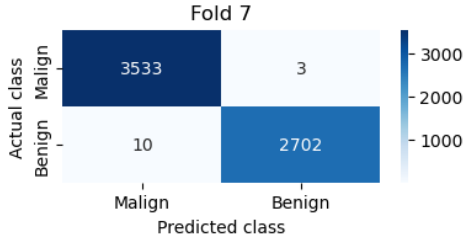
Accuracy: 0.9965
Precision: 0.9974
Recall: 0.9945
F-measure: 0.9959



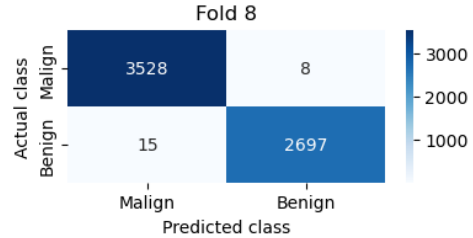
Accuracy: 0.9973
Precision: 0.9974
Recall: 0.9963
F-measure: 0.9969



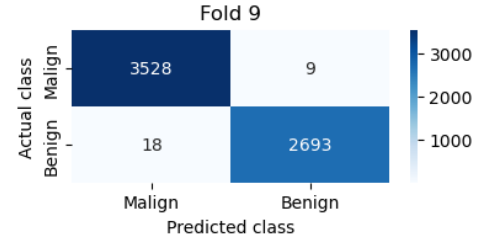
Accuracy: 0.9966
Precision: 0.9959
Recall: 0.9963
F-measure: 0.9961



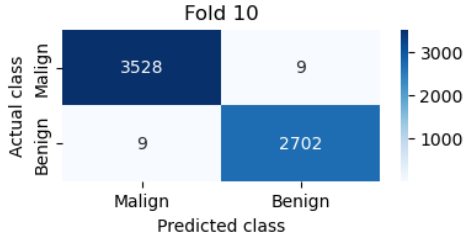
Accuracy: 0.9979
Precision: 0.9989
Recall: 0.9963
F-measure: 0.9976



Accuracy: 0.9963
Precision: 0.9970
Recall: 0.9945
F-measure: 0.9958



Accuracy: 0.9957
Precision: 0.9967
Recall: 0.9934
F-measure: 0.9950



Accuracy: 0.9971
Precision: 0.9967
Recall: 0.9967
F-measure: 0.9967

Mean Accuracy: 0.9970
Mean Precision: 0.9972
Mean Recall: 0.9958
Mean F-measure: 0.9965

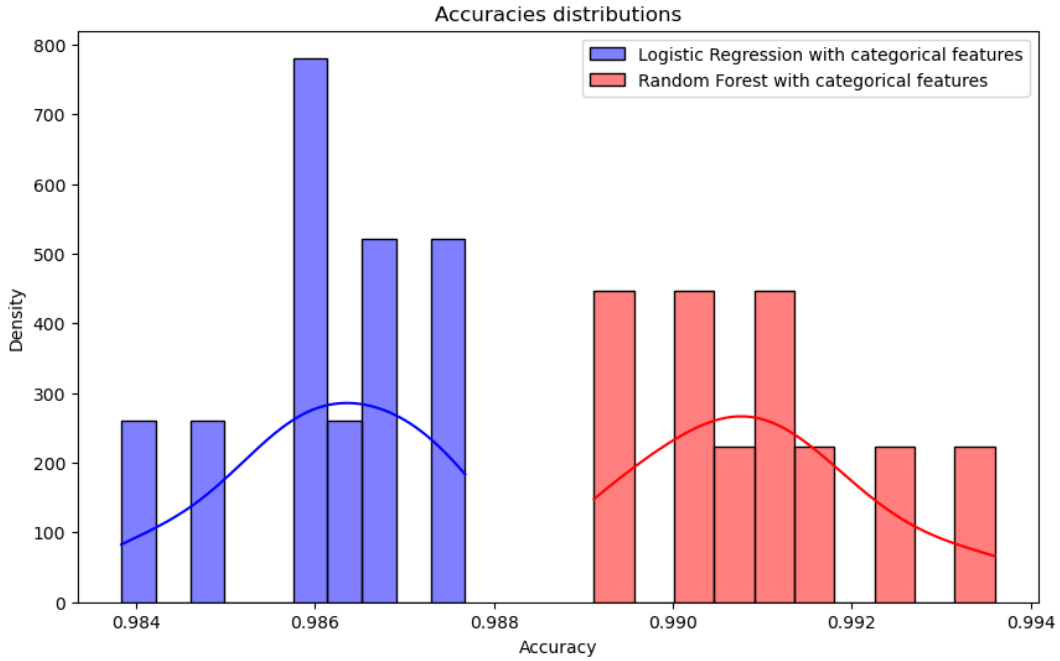
Identification of malwares from Windows PE files

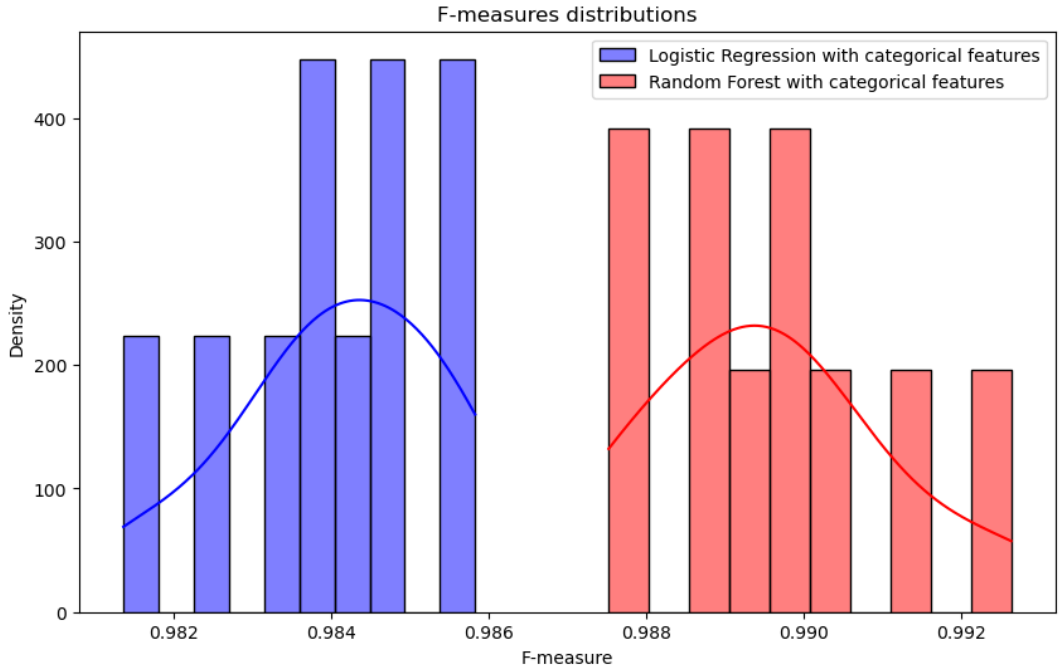
Table of Contents

Results and Insights Summary

Results Distribution

	Experiment 1	Experiment 2	Experiment 3	Experiment 4	Experiment 5	Experiment 6
Acccuracy	0.7969	0.9861	0.9865	0.9898	0.9910	0.9970
F-Measure	0.7273	0.9840	0.9845	0.9883	0.9897	0.9965





From the distribution of the accuracies and F-Measures over each fold of Logistic Regression and Random Forest we can see that those measures have a Gaussian distribution. In fact, the Gaussian bell is quite visible even though the number of samples is quite low (10)

T-Test

Result Summary

Insights

```
# t-test for accuracy
from scipy.stats import ttest_ind

t_stat, p_value = ttest_ind(accuracy_list_lr_cat, accuracy_list_rf_cat)

print(f"t-statistic: {t_stat:.10f}")
print(f"p-value: {p_value:.10f}")
```

→ t-statistic: -8.4086810542
p-value: 0.0000001197

```
# t-test for F-measures
from scipy.stats import ttest_ind

t_stat, p_value = ttest_ind(f1_list_lr_cat, f1_list_rf_cat)

print(f"t-statistic: {t_stat:.10f}")
print(f"p-value: {p_value:.10f}")
```

→ t-statistic: -8.4126922279
p-value: 0.0000001189

↓

Given the statistical evidence of the difference in performance between the two models, we can conclude that, since Random Forest is better in both accuracy (0.9910 vs 0.9861) and F-measure (0.9897 vs 0.9840), it performs better than Logistic Regression.

Identification of malwares from Windows PE files

Table of Contents

X
A
I

For our classification problem, we used Logistic Regression and Random Forest, which have notable differences in terms of explainability:

- Logistic Regression is a white box model, meaning that is easily explainable by displaying the coefficients used for the classification.
- Random Forest is a black box model, meaning that we need post-hoc techniques such as SHAP values to be able to explain the model.

Since we've seen that categorical features offer a very good balance between performances and computational effort, we're going to explain Logistic Regression and Random Forest when using categorical variables for classification.

We explain the Linear Regression classifier using both coefficients and SHAP

We explain the Random Forest classifier using SHAP

Logistic Regression Explainability using Coefficients

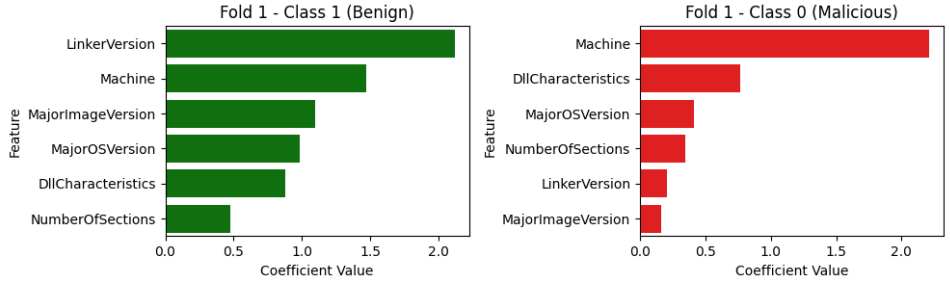
x
A
I

```python
# Extract model coefficients
coefficients = pipeline.named_steps['classifier'].coef_[0]

# Extract the one-hot encoded feature names
ohe = pipeline.named_steps['preprocessor'].named_transformers_['cat_transform']
ohe_feature_names = ohe.get_feature_names_out(categorical_columns)

# Create a mapping from one-hot encoded features to their original categorical feature
feature_coefficients_pos = {}
feature_coefficients_neg = {}

for feature in categorical_columns:
    # Find all one-hot encoded features that belong to the current categorical feature
    matching_features = [f for f in ohe_feature_names if f.startswith(feature)]

    # Get their indices in the coefficient array
    indices = [ohe_feature_names.tolist().index(f) for f in matching_features]

    # Aggregate coefficients (e.g., by taking the mean)
    feature_coefficients_pos[feature] = np.mean([coefficients[i] for i in indices if coefficients[i] > 0])
    feature_coefficients_neg[feature] = np.mean(np.abs(([coefficients[i] for i in indices if coefficients[i] < 0])))
```

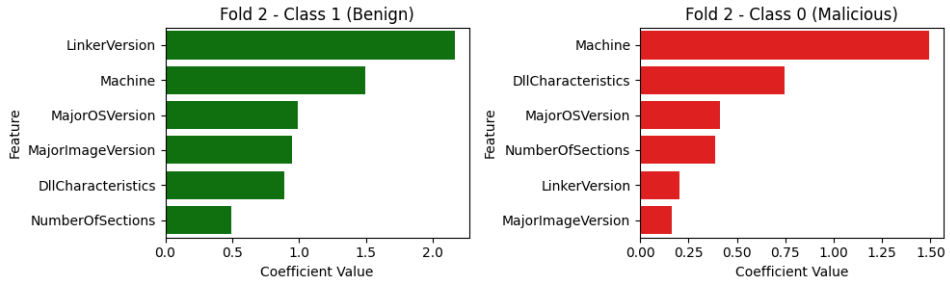
For each fold of the 10-fold cross-validation, we evaluate how much each categorical feature contributes to distinguishing between the "Benign" and "Malicious" classes. Since one-hot encoding expands each categorical variable into multiple binary variables, the coefficients of these encoded features are grouped and averaged to reflect the importance of the original categorical variable.
Positive coefficients push the prediction toward "Benign," while negative coefficients push it toward "Malicious." This approach allows us to understand which features consistently support or oppose the target class across the folds.

XAI

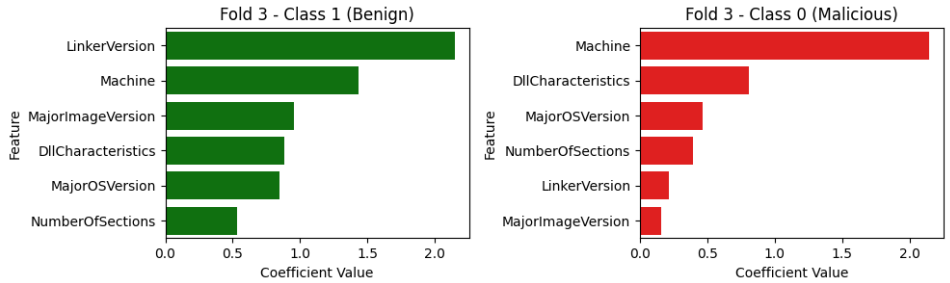

IN SUPREMÆ DIGNITATIS
1343


Fold 1 - Class 1 (Benign)
LinkerVersion
Machine
MajorImageVersion
MajorOSVersion
DllCharacteristics
NumberOfSections
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value

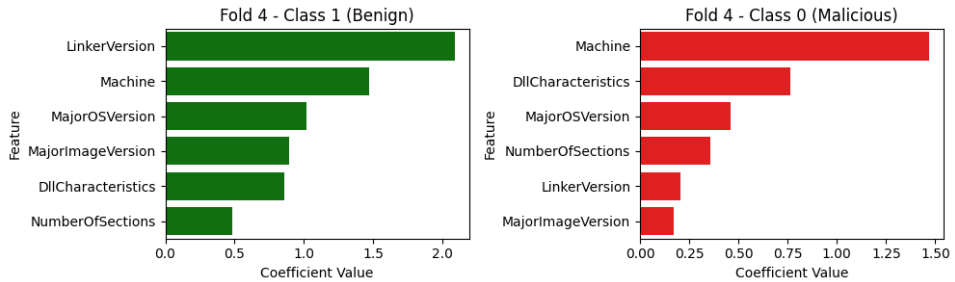
Fold 1 - Class 0 (Malicious)
Machine
DllCharacteristics
MajorOSVersion
NumberOfSections
LinkerVersion
MajorImageVersion
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value


Fold 2 - Class 1 (Benign)
LinkerVersion
Machine
MajorOSVersion
MajorImageVersion
DllCharacteristics
NumberOfSections
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value

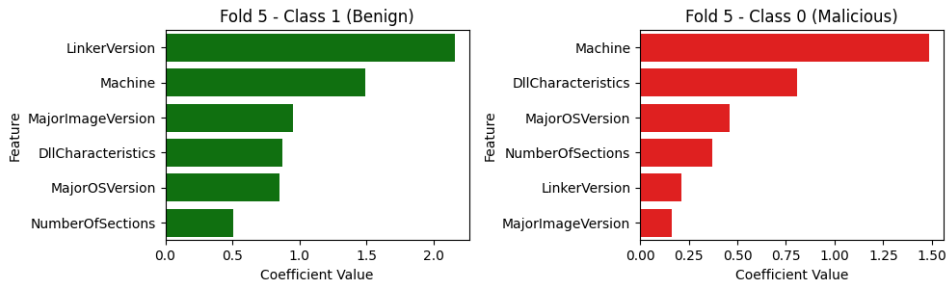
Fold 2 - Class 0 (Malicious)
Machine
DllCharacteristics
MajorOSVersion
NumberOfSections
LinkerVersion
MajorImageVersion
Feature
0.00 0.25 0.50 0.75 1.00 1.25 1.50
Coefficient Value


Fold 3 - Class 1 (Benign)
LinkerVersion
Machine
MajorImageVersion
DllCharacteristics
MajorOSVersion
NumberOfSections
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value

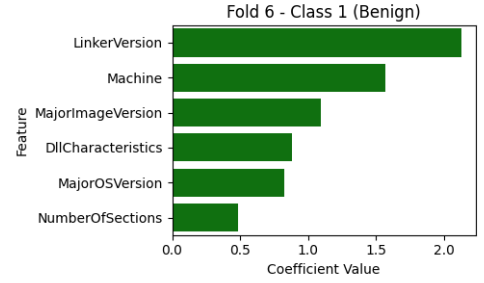
Fold 3 - Class 0 (Malicious)
Machine
DllCharacteristics
MajorOSVersion
NumberOfSections
LinkerVersion
MajorImageVersion
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value

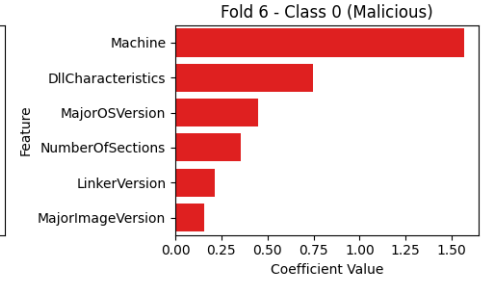

Fold 4 - Class 1 (Benign)
LinkerVersion
Machine
MajorOSVersion
MajorImageVersion
DllCharacteristics
NumberOfSections
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value

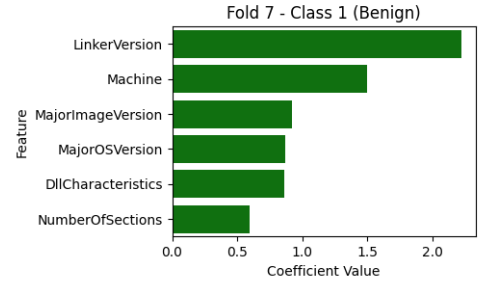
Fold 4 - Class 0 (Malicious)
Machine
DllCharacteristics
MajorOSVersion
NumberOfSections
LinkerVersion
MajorImageVersion
Feature
0.00 0.25 0.50 0.75 1.00 1.25 1.50
Coefficient Value

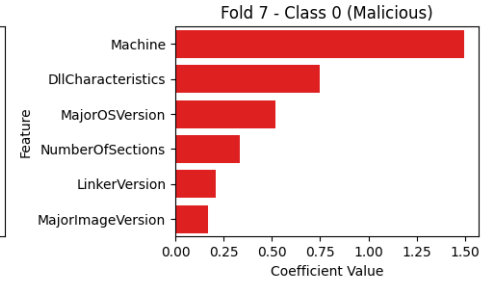

Fold 5 - Class 1 (Benign)
LinkerVersion
Machine
MajorImageVersion
DllCharacteristics
MajorOSVersion
NumberOfSections
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value

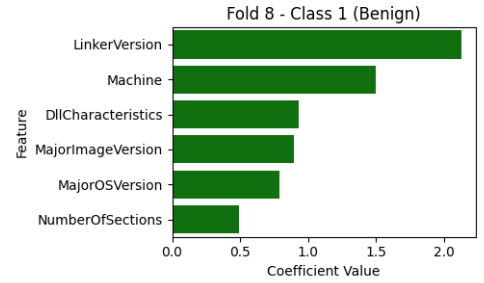
Fold 5 - Class 0 (Malicious)
Machine
DllCharacteristics
MajorOSVersion
NumberOfSections
LinkerVersion
MajorImageVersion
Feature
0.00 0.25 0.50 0.75 1.00 1.25 1.50
Coefficient Value

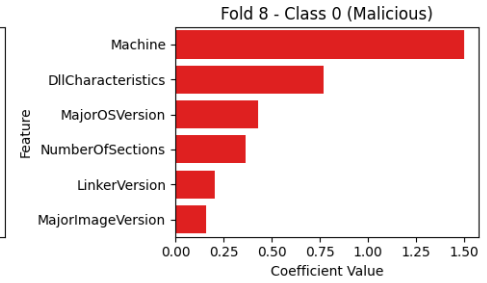

Fold 6 - Class 1 (Benign)
LinkerVersion
Machine
MajorImageVersion
DllCharacteristics
MajorOSVersion
NumberOfSections
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value

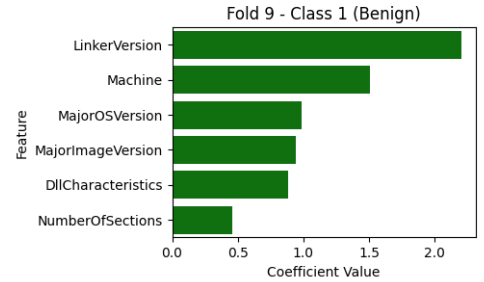

Fold 6 - Class 0 (Malicious)
Machine
DllCharacteristics
MajorOSVersion
NumberOfSections
LinkerVersion
MajorImageVersion
Feature
0.00 0.25 0.50 0.75 1.00 1.25 1.50
Coefficient Value

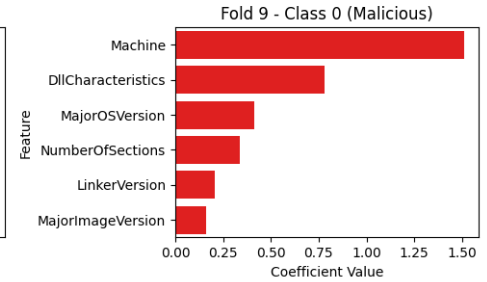

Fold 7 - Class 1 (Benign)
LinkerVersion
Machine
MajorImageVersion
MajorOSVersion
DllCharacteristics
NumberOfSections
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value

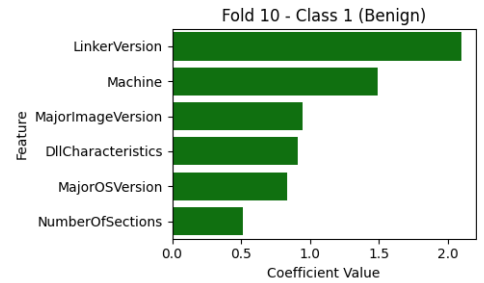

Fold 7 - Class 0 (Malicious)
Machine
DllCharacteristics
MajorOSVersion
NumberOfSections
LinkerVersion
MajorImageVersion
Feature
0.00 0.25 0.50 0.75 1.00 1.25 1.50
Coefficient Value

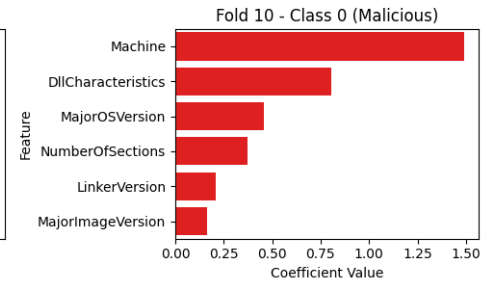

Fold 8 - Class 1 (Benign)
LinkerVersion
Machine
DllCharacteristics
MajorImageVersion
MajorOSVersion
NumberOfSections
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value


Fold 8 - Class 0 (Malicious)
Machine
DllCharacteristics
MajorOSVersion
NumberOfSections
LinkerVersion
MajorImageVersion
Feature
0.00 0.25 0.50 0.75 1.00 1.25 1.50
Coefficient Value


Fold 9 - Class 1 (Benign)
LinkerVersion
Machine
MajorOSVersion
MajorImageVersion
DllCharacteristics
NumberOfSections
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value


Fold 9 - Class 0 (Malicious)
Machine
DllCharacteristics
MajorOSVersion
NumberOfSections
LinkerVersion
MajorImageVersion
Feature
0.00 0.25 0.50 0.75 1.00 1.25 1.50
Coefficient Value


Fold 10 - Class 1 (Benign)
LinkerVersion
Machine
MajorImageVersion
DllCharacteristics
MajorOSVersion
NumberOfSections
Feature
0.0 0.5 1.0 1.5 2.0
Coefficient Value


Fold 10 - Class 0 (Malicious)
Machine
DllCharacteristics
MajorOSVersion
NumberOfSections
LinkerVersion
MajorImageVersion
Feature
0.00 0.25 0.50 0.75 1.00 1.25 1.50
Coefficient Value

Logistic Regression Explainability using SHAP

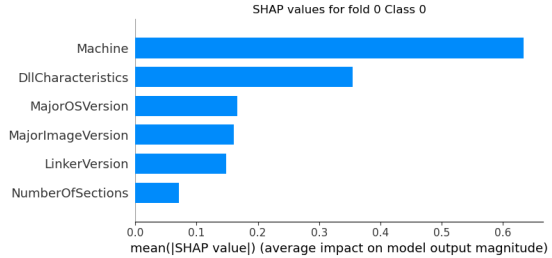
X
A
I

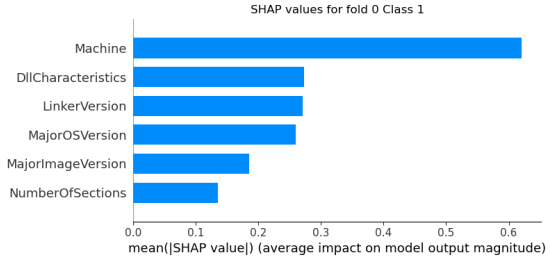
```
for i, (train_idx, test_idx) in enumerate(cv.split(X, y)):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

    pipeline.fit(X_train, y_train)

    # Get feature names from one-hot encoder
    ohe = pipeline.named_steps["preprocessor"].named_transformers_["cat_transform"]
    ohe_feature_names = ohe.get_feature_names_out(categorical_columns)

    # Create SHAP explainer on the trained LogisticRegression
    explainer = shap.Explainer(pipeline.named_steps["classifier"],pipeline.named_steps["preprocessor"].transform(X_test))
    shap_values = explainer(pipeline.named_steps["preprocessor"].transform(X_test))

    shap_df = pd.DataFrame(shap_values.values, columns=ohe_feature_names)

    # Build a map from each encoded column to its original feature
    feature_map = {}
    for encoded_col in ohe_feature_names:
        orig_feature = encoded_col.split('_')[0]
        feature_map.setdefault(orig_feature, []).append(encoded_col)

    agg_shap_per_row_class0 = []
    agg_shap_per_row_class1 = []

    for index, row in shap_df.iterrows():   # Iterate on all rows of shap_df (all the objects in X_test)
        row_agg_shap_class0 = {}
        row_agg_shap_class1 = {}

        for orig_feat, enc_feats in feature_map.items():
            shap_values_class0 = row[enc_feats][row[enc_feats] < 0].mean()
            shap_values_class1 = row[enc_feats][row[enc_feats] > 0].mean()

            if shap_values_class0 != 0:
                row_agg_shap_class0[orig_feat] = shap_values_class0
            if shap_values_class1 != 0:
                row_agg_shap_class1[orig_feat] = shap_values_class1

        agg_shap_per_row_class0.append(row_agg_shap_class0)
        agg_shap_per_row_class1.append(row_agg_shap_class1)
```

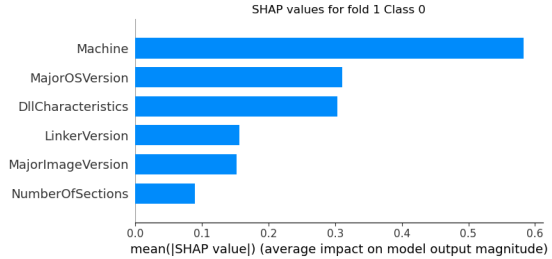
Since SHAP values are computed for the one-hot encoded features, we group them back into their original categorical feature for both class 0 (Malicious) and class 1 (Benign). Therefore, for each fold, we display the aggregated SHAP value of each original categorical feature for both classes.

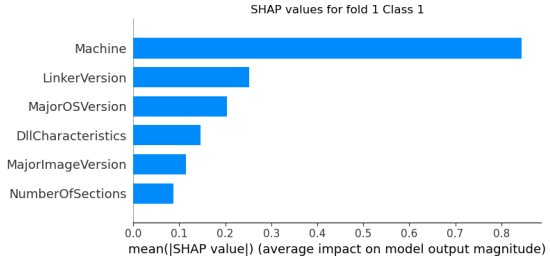
XAI

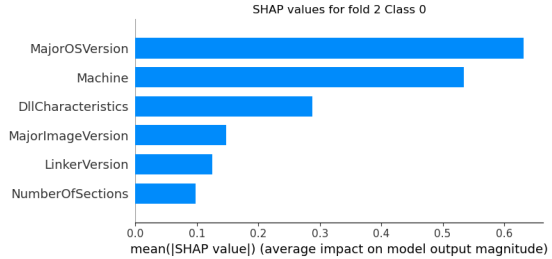

IN SUPREMÆ DIGNITATIS
1343

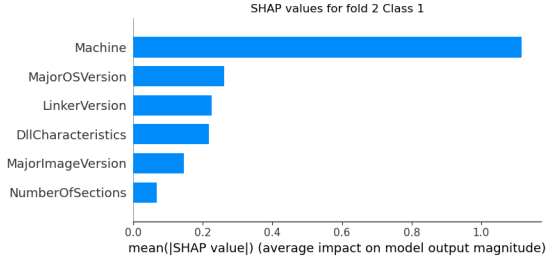

SHAP values for fold 0 Class 0
Machine
DllCharacteristics
MajorOSVersion
MajorImageVersion
LinkerVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6
mean(|SHAP value|) (average impact on model output magnitude)

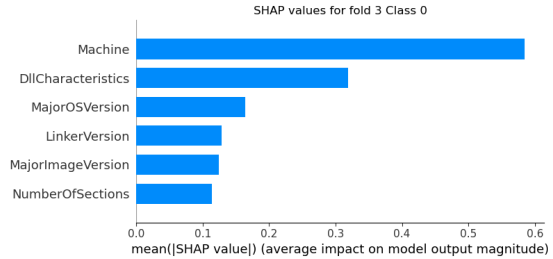

SHAP values for fold 0 Class 1
Machine
DllCharacteristics
LinkerVersion
MajorOSVersion
MajorImageVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6
mean(|SHAP value|) (average impact on model output magnitude)

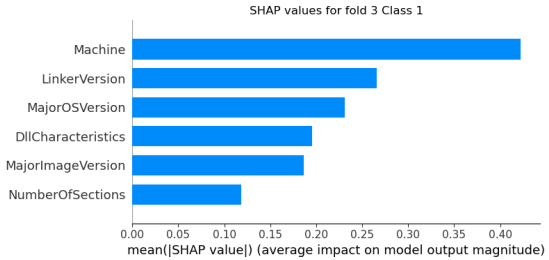

SHAP values for fold 1 Class 0
Machine
MajorOSVersion
DllCharacteristics
LinkerVersion
MajorImageVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6
mean(|SHAP value|) (average impact on model output magnitude)

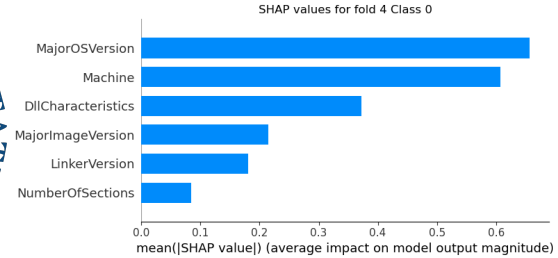

SHAP values for fold 1 Class 1
Machine
LinkerVersion
MajorOSVersion
DllCharacteristics
MajorImageVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
mean(|SHAP value|) (average impact on model output magnitude)

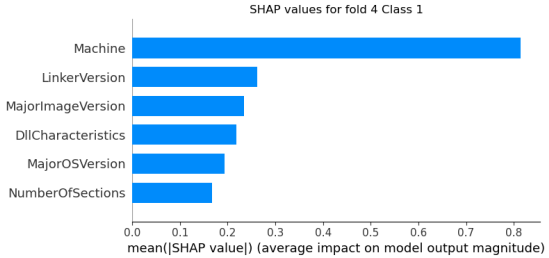

SHAP values for fold 2 Class 0
MajorOSVersion
Machine
DllCharacteristics
MajorImageVersion
LinkerVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6
mean(|SHAP value|) (average impact on model output magnitude)

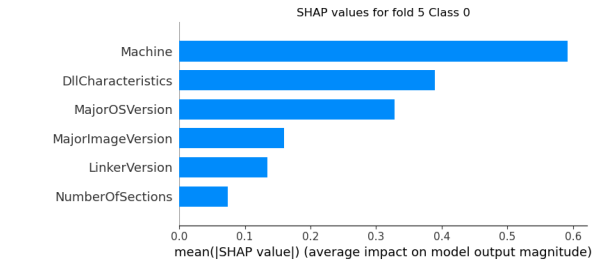

SHAP values for fold 2 Class 1
Machine
MajorOSVersion
LinkerVersion
DllCharacteristics
MajorImageVersion
NumberOfSections
0.0 0.2 0.4 0.6 0.8 1.0
mean(|SHAP value|) (average impact on model output magnitude)

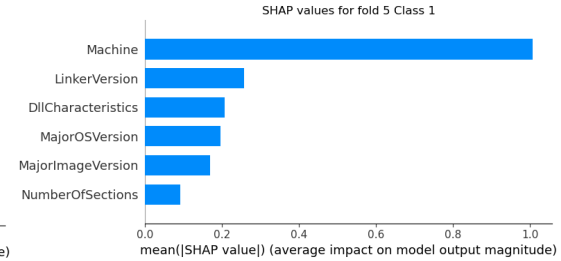

SHAP values for fold 3 Class 0
Machine
DllCharacteristics
MajorOSVersion
LinkerVersion
MajorImageVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6
mean(|SHAP value|) (average impact on model output magnitude)

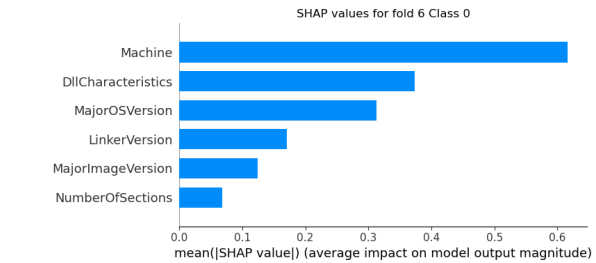

SHAP values for fold 3 Class 1
Machine
LinkerVersion
MajorOSVersion
DllCharacteristics
MajorImageVersion
NumberOfSections
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40
mean(|SHAP value|) (average impact on model output magnitude)

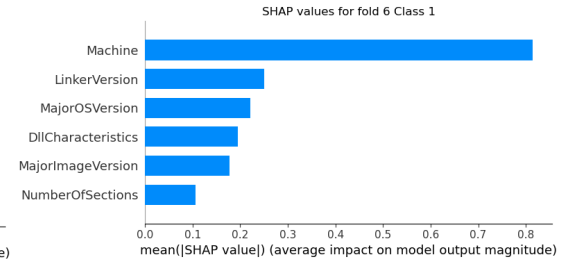

SHAP values for fold 4 Class 0
MajorOSVersion
Machine
DllCharacteristics
MajorImageVersion
LinkerVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6
mean(|SHAP value|) (average impact on model output magnitude)

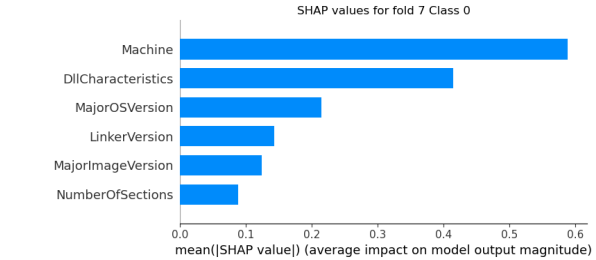

SHAP values for fold 4 Class 1
Machine
LinkerVersion
MajorImageVersion
DllCharacteristics
MajorOSVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
mean(|SHAP value|) (average impact on model output magnitude)

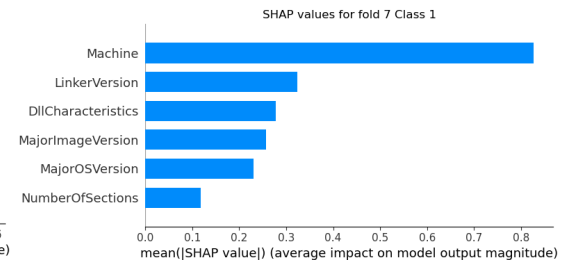

SHAP values for fold 5 Class 0
Machine
DllCharacteristics
MajorOSVersion
MajorImageVersion
LinkerVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6
mean(|SHAP value|) (average impact on model output magnitude)

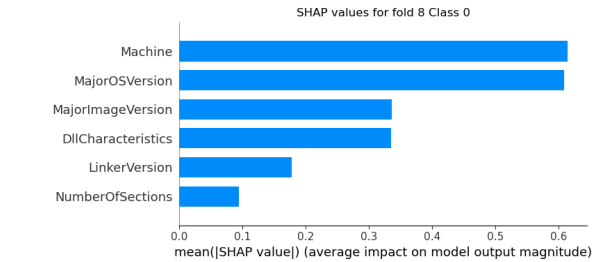

SHAP values for fold 5 Class 1
Machine
LinkerVersion
DllCharacteristics
MajorOSVersion
MajorImageVersion
NumberOfSections
0.0 0.2 0.4 0.6 0.8 1.0
mean(|SHAP value|) (average impact on model output magnitude)

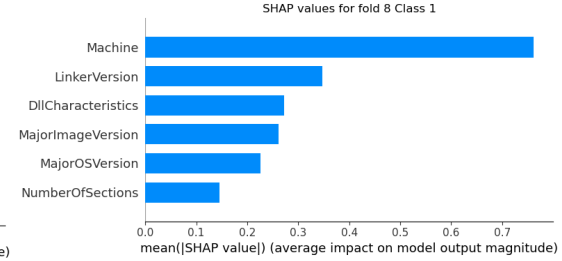

SHAP values for fold 6 Class 0
Machine
DllCharacteristics
MajorOSVersion
LinkerVersion
MajorImageVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6
mean(|SHAP value|) (average impact on model output magnitude)

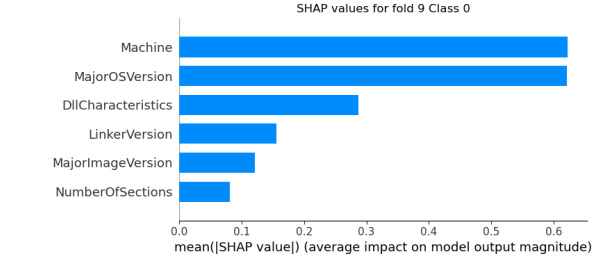

SHAP values for fold 6 Class 1
Machine
LinkerVersion
MajorOSVersion
DllCharacteristics
MajorImageVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
mean(|SHAP value|) (average impact on model output magnitude)

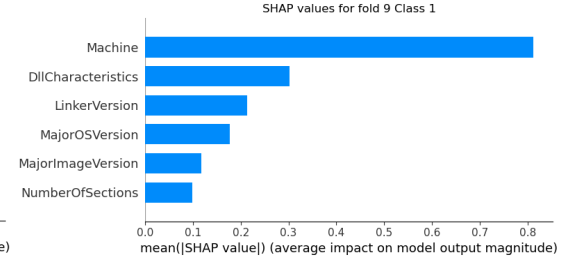

SHAP values for fold 7 Class 0
Machine
DllCharacteristics
MajorOSVersion
LinkerVersion
MajorImageVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6
mean(|SHAP value|) (average impact on model output magnitude)


SHAP values for fold 7 Class 1
Machine
LinkerVersion
DllCharacteristics
MajorImageVersion
MajorOSVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
mean(|SHAP value|) (average impact on model output magnitude)


SHAP values for fold 8 Class 0
Machine
MajorOSVersion
MajorImageVersion
DllCharacteristics
LinkerVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6
mean(|SHAP value|) (average impact on model output magnitude)


SHAP values for fold 8 Class 1
Machine
LinkerVersion
DllCharacteristics
MajorImageVersion
MajorOSVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7
mean(|SHAP value|) (average impact on model output magnitude)


SHAP values for fold 9 Class 0
Machine
MajorOSVersion
DllCharacteristics
LinkerVersion
MajorImageVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6
mean(|SHAP value|) (average impact on model output magnitude)


SHAP values for fold 9 Class 1
Machine
DllCharacteristics
LinkerVersion
MajorOSVersion
MajorImageVersion
NumberOfSections
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
mean(|SHAP value|) (average impact on model output magnitude)

Random Forest Explainability using SHAP

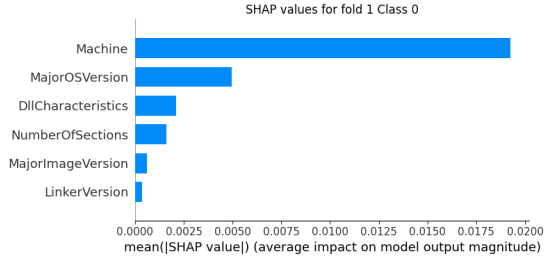
XAI

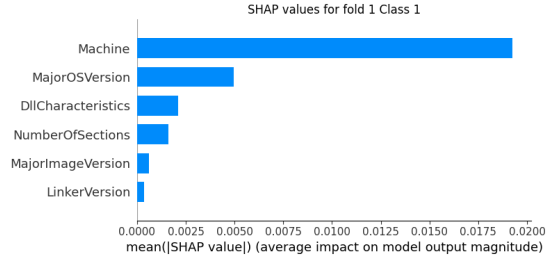
```
...

explainer = shap.TreeExplainer(pipeline.named_steps["classifier"], X_test_dense)
shap_values = explainer.shap_values(X_test_dense)

shap_df_class0 = pd.DataFrame(shap_values[0], columns=ohe_feature_names)
shap_df_class1 = pd.DataFrame(shap_values[1], columns=ohe_feature_names)

feature_map = {}
for encoded_col in ohe_feature_names:
    orig_feature = encoded_col.split('_')[0]
    feature_map.setdefault(orig_feature, []).append(encoded_col)

agg_shap_per_row_class0 = []
agg_shap_per_row_class1 = []

for index, row in shap_df_class0.iterrows():
    row_agg_shap_class0 = {}

    for orig_feat, enc_feats in feature_map.items():
        shap_values_class0 = row[enc_feats].mean()
        if shap_values_class0 != 0:
            row_agg_shap_class0[orig_feat] = shap_values_class0

    agg_shap_per_row_class0.append(row_agg_shap_class0)

for index, row in shap_df_class1.iterrows():
    row_agg_shap_class1 = {}

    for orig_feat, enc_feats in feature_map.items():
        shap_values_class1 = row[enc_feats].mean()
        if shap_values_class1 != 0:
            row_agg_shap_class1[orig_feat] = shap_values_class1

    agg_shap_per_row_class1.append(row_agg_shap_class1)
```

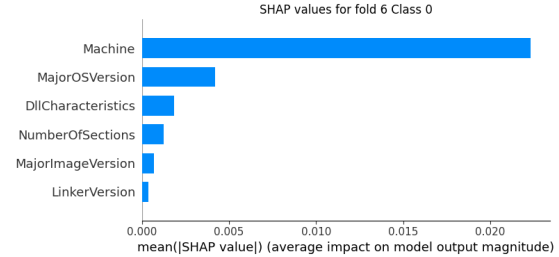
As before, SHAP values computed for the one-hot encoded features are grouped into their original categorical feature for both classes.
At the same time, for each fold, we display the aggregated SHAP value of each original categorical feature for both classes.

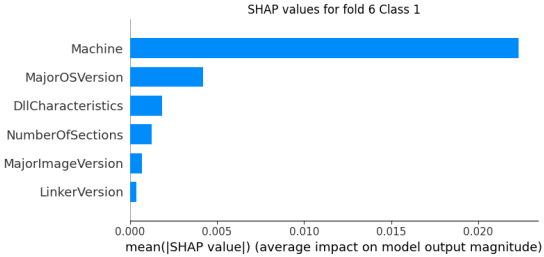
XAI

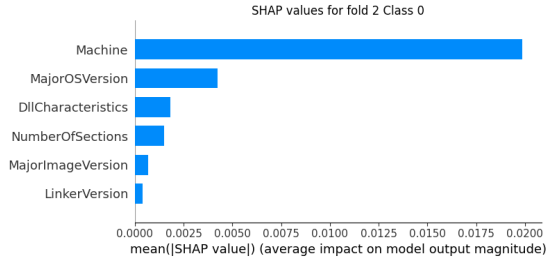

IN SUPREMÆ DIGNITATIS
1343

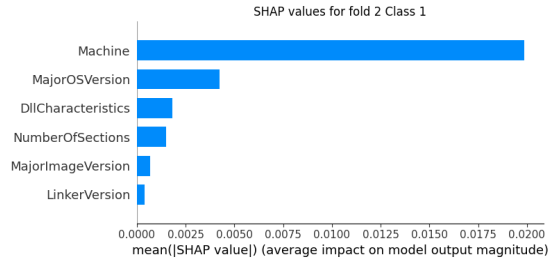

SHAP values for fold 1 Class 0
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)

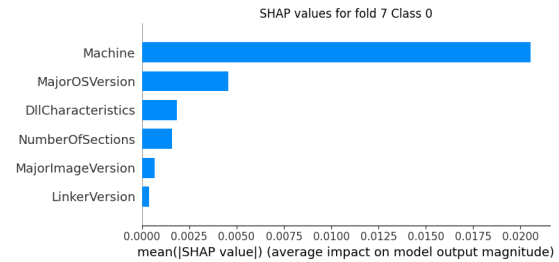

SHAP values for fold 1 Class 1
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)

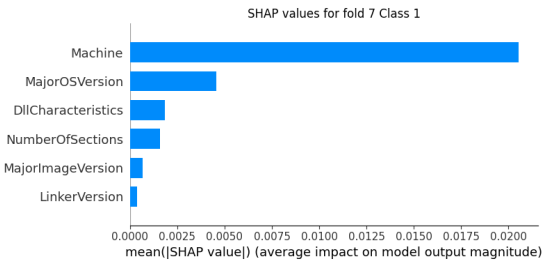

SHAP values for fold 6 Class 0
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.000 0.005 0.010 0.015 0.020
mean(|SHAP value|) (average impact on model output magnitude)

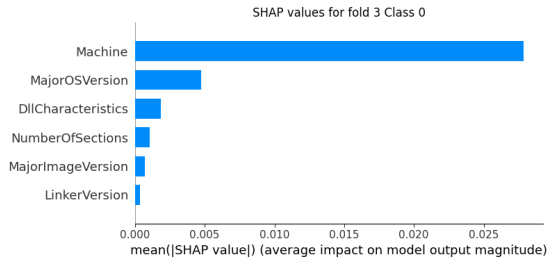

SHAP values for fold 6 Class 1
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.000 0.005 0.010 0.015 0.020
mean(|SHAP value|) (average impact on model output magnitude)

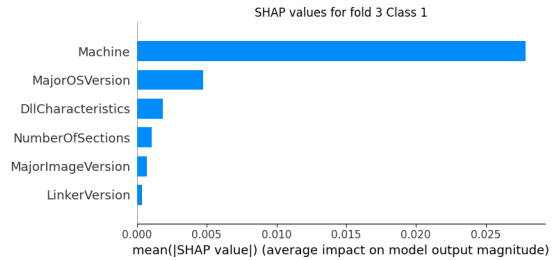

SHAP values for fold 2 Class 0
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)

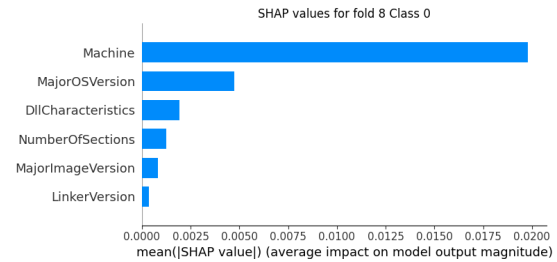

SHAP values for fold 2 Class 1
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)

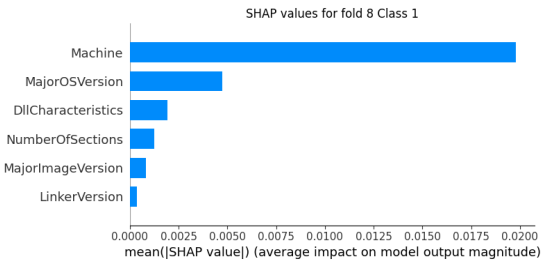

SHAP values for fold 7 Class 0
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)

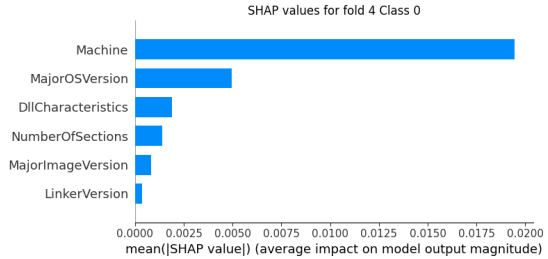

SHAP values for fold 7 Class 1
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)

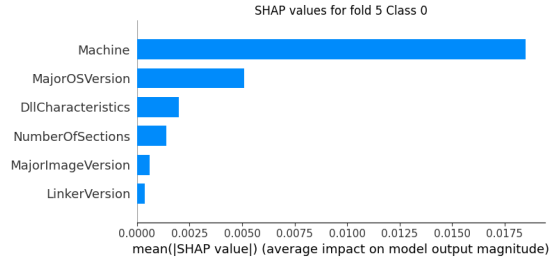

SHAP values for fold 3 Class 0
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.000 0.005 0.010 0.015 0.020 0.025
mean(|SHAP value|) (average impact on model output magnitude)

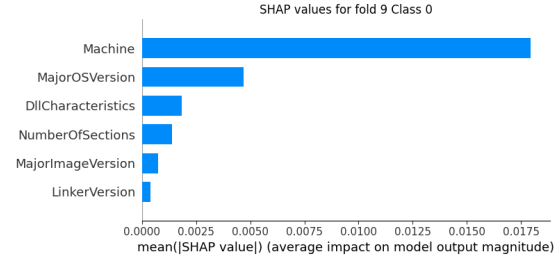

SHAP values for fold 3 Class 1
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.000 0.005 0.010 0.015 0.020 0.025
mean(|SHAP value|) (average impact on model output magnitude)

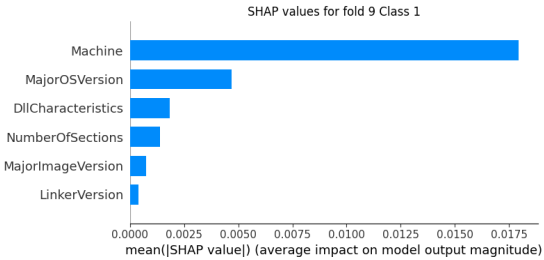

SHAP values for fold 8 Class 0
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)

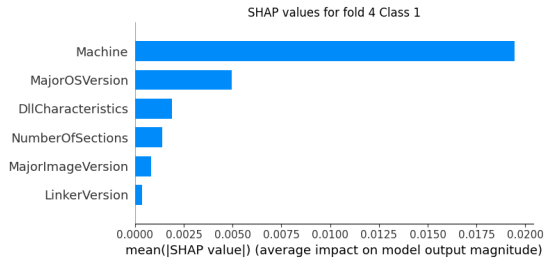

SHAP values for fold 8 Class 1
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)

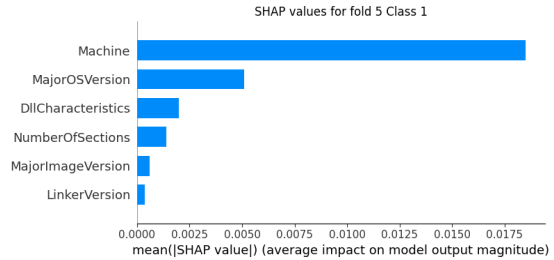

SHAP values for fold 4 Class 0
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)

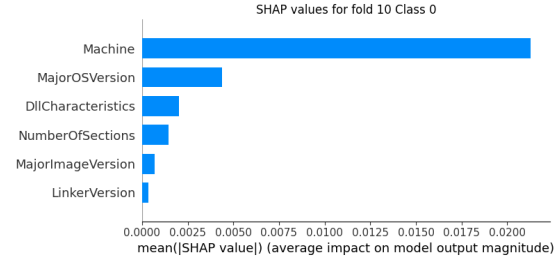

SHAP values for fold 5 Class 0
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175
mean(|SHAP value|) (average impact on model output magnitude)

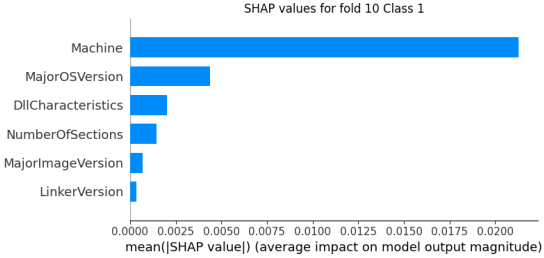

SHAP values for fold 9 Class 0
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175
mean(|SHAP value|) (average impact on model output magnitude)


SHAP values for fold 9 Class 1
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175
mean(|SHAP value|) (average impact on model output magnitude)


SHAP values for fold 4 Class 1
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)


SHAP values for fold 5 Class 1
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175
mean(|SHAP value|) (average impact on model output magnitude)


SHAP values for fold 10 Class 0
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)


SHAP values for fold 10 Class 1
Machine
MajorOSVersion
DllCharacteristics
NumberOfSections
MajorImageVersion
LinkerVersion
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
mean(|SHAP value|) (average impact on model output magnitude)

Thanks for your attention

References

Identification of malwares from Windows PE files

Dataset

Info on PE files

Scikit

Seaborn

SHAP

Pandas

Numpy