# Classification of Malwares

**B Prem Sundar (CB.EN.U4ELC20051)**
**Sagi Sriyank (CB.EN.U4ELC20061)**
**Laxmi Ganesh(CB.EN.U4ELC20048)**

# References:

- Dataset:

https://www.kaggle.com/datasets/saurabhshahane/classification-of-malwares

- Label encoder reference:

https://www.geeksforgeeks.org/how-to-convert-categorical-string-data-into-numeric-in-python/

# Problem Formulation (3 marks)

**<u>Objective</u>**: Classification of Malwares (CLaMP)

**<u>Dataset details</u>**:

- No. of rows: 5184
- No. of columns: 69
- No. of classes: 2

- **<u>Assumptions</u>**:
- The data set "ClaMP_Integrated-5184.csv" was considered for solving where in we have 54 raw features and 15 derived features

# Feature Description (2 marks)

| | |
|---|---|
| Emagic | : Magic number |
| Ecblp | :Bytes on last page of file |
| Ecp | :Pages in file |
| ECrlc | :Relocations |
| ECparhdr | :Size of header in paragraphs |
| EMinalloc | :Minimum extra paragraphs needed |
| Emaxalloc | :Maximum extra paragraphs needed |
| ESs | :Initial (relative) SS value |
| ESs | :Initial SP value |
| ECsum | :Checksum |
| EIp | :Initial IP value |
| ECs | :Initial (relative) CS value |
| ELfarlc | :File address of relocation table |
| EOvno | :overlay number |
| ERes | :Reserved uint16s |
| EOemid | :OEM identifier (for e_oeminfo) |
| EOeminfo | :OEM information; e_oemid specific |
| ERes2 | :Reserved uint16s |
| ELfanew | :File address of new exe header |

- FILE_HEADER:
- Machine,NumberOfSections , CreationYear
- PointerToSymbolTable,NumberOfSymbols
- SizeOfOptionalHeader,Characteristics.

OPTIONAL_HEADER :

- Magic, MajorLinkerVersion, MinorLinkerVersion, SizeOfCode, SizeOfInitializedData,SizeOfUninitializedData, AddressOfEntryPoint,BaseOfCode, BaseOfData, ImageBase, SectionAlignment, FileAlignment","MajorOperatingSystemVersion.
- MinorOperatingSystemVersion, MajorImageVersion, MinorImageVersion, MajorSubsystemVersion, MinorSubsystemVersion, SizeOfImage, SizeOfHeaders, CheckSum,Subsystem, DllCharacteristics, SizeOfStackReserve, SizeOfStackCommit,SizeOfHeapReserve, SizeOfHeapCommit, LoaderFlags, NumberOfRvaAndSizes.

# Common in all methods/calculations

```
In [2]: #authors: B Prem Sundar , Sriyank , Ganesh Peddina
        #objective: To classify a malware
        #input: Dataset
        #output: Accuracy
        import pandas as pd #data analysis toolkit
        import matplotlib.pyplot as plt # for plotting graphs
        import numpy as np # for high level computations
        %matplotlib inline
```

```
In [3]: from sklearn.preprocessing import StandardScaler # standardization of values
        from sklearn.preprocessing import MinMaxScaler # Normalization of values
        from sklearn.model_selection import train_test_split # to split data
        from sklearn.neighbors import KNeighborsClassifier #KNN classifier
        from sklearn.metrics import confusion_matrix,accuracy_score # to get confusion matrix and accuracy
        from sklearn.model_selection import cross_val_score # to perform evaluation and cross-validation
```

```
In [4]: data_set = pd.read_csv("ClaMP_Integrated-5184.csv") # dataset_input
```

```
In [5]: from sklearn.preprocessing import LabelEncoder

        # Creating a instance of label Encoder.
        le = LabelEncoder()

        # Using .fit_transform function to fit label
        # encoder and return encoded label
        data_set['packer_type'] = le.fit_transform(data_set['packer_type'])
```

# Common in all methods/calculations

```
In [6]: data_set = np.round(data_set, decimals=2) # rouding all values in dataset to 2 decimal places
        data_set.head() # first 5 values in dataset
```

Out[6]:

| | e_cblp | e_cp | e_cparhdr | e_maxalloc | e_sp | e_lfanew | NumberOfSections | CreationYear | FH_char0 | FH_char1 | ... | sus_sections | non_sus_sections | packer | p |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 144 | 3 | 4 | 65535 | 184 | 256 | 4 | 1 | 0 | 1 | ... | 1 | 3 | 0 | |
| 1 | 144 | 3 | 4 | 65535 | 184 | 184 | 4 | 1 | 0 | 1 | ... | 1 | 3 | 0 | |
| 2 | 144 | 3 | 4 | 65535 | 184 | 272 | 5 | 1 | 0 | 1 | ... | 1 | 4 | 0 | |
| 3 | 144 | 3 | 4 | 65535 | 184 | 184 | 1 | 1 | 0 | 1 | ... | 0 | 1 | 0 | |
| 4 | 144 | 3 | 4 | 65535 | 184 | 224 | 5 | 1 | 0 | 1 | ... | 1 | 4 | 0 | |

5 rows × 70 columns

```
In [7]: dset_modified=data_set.drop('class',axis=1)
```

```
In [8]: data_set_feat = pd.DataFrame(dset_modified, columns=data_set.columns[:-1])
```

```
In [9]: data_set_feat = np. round(data_set_feat, decimals=2)
```

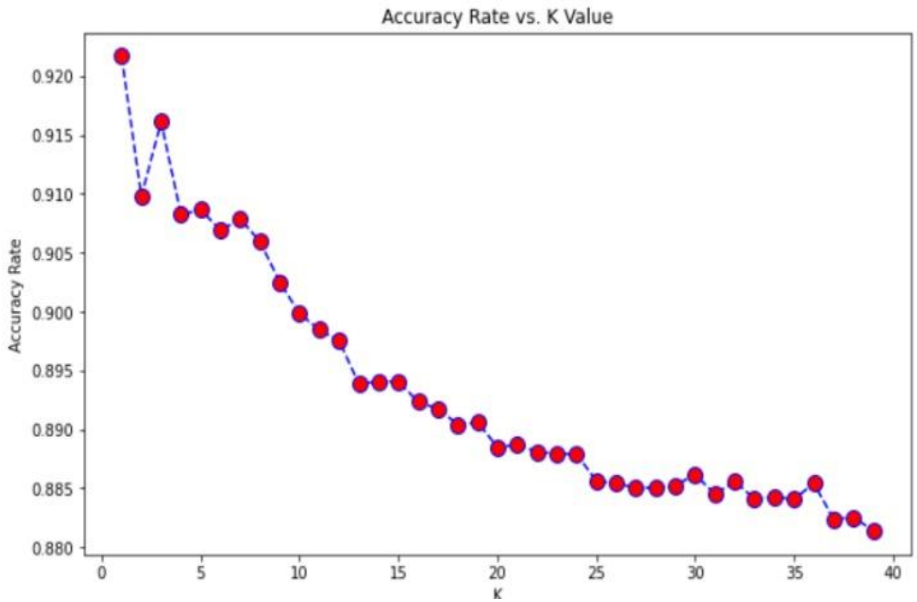
```
In [11]: one_train, one_test, two_train, two_test = train_test_split(data_set,data_set['class'],test_size=0.30)
         # test_train split with test size =30% and train size =70%
```

```
In [12]: # Computation of accuracy rates for various neighbour values
         Accurate_rates = []
         for i in range(1,40):
             k_nearest_neighbour = KNeighborsClassifier(n_neighbors=i)
             final_score=cross_val_score(k_nearest_neighbour, data_set_feat,data_set['class'],cv=5)
             Accurate_rates.append(final_score.mean())
```

# Knn classifier (5 marks)

- Minkowski Distance is the distance metric utilized in this approach
- The data set is divided into 30% for testing and 70% for training.
- The entire dataset has been divided into five groups because the cv value for this model is 5. The model is tested using the first fold, and trained using the next four folds.
- In the second model, the second fold serves as a testing set while the remaining folds are used for training. This technique is performed for each fold

```
In [13]: plt.figure(figsize=(10,6))

         plt.plot(range(1,40),Accurate_rates, color='blue', linestyle='dashed', marker='o',markerfacecolor='red', markersize=10)

         plt.title('Accuracy Rate vs. K Value')

         plt.xlabel('K')

         plt.ylabel( 'Accuracy Rate')

Out[13]: Text(0, 0.5, 'Accuracy Rate')
```



```
In [5]: from sklearn.preprocessing import LabelEncoder

        # Creating a instance of label Encoder.
        le = LabelEncoder()

        # Using .fit_transform function to fit label
        # encoder and return encoded label
        data_set['packer_type'] = le.fit_transform(data_set['packer_type'])
```

```
In [25]: # Best case identifier
         max_index = Accurate_rates.index(max(Accurate_rates))
         k_nearest_neighbour = KNeighborsClassifier(n_neighbors=(max_index+1))
         k_nearest_neighbour.fit(one_train,two_train)
         prediction = k_nearest_neighbour.predict(one_test)
         print('For K=',max_index+1)
         print('Confusion matrix:')
         print('\n')
         print(confusion_matrix(two_test,prediction)) # Confusion Matrix
         print('\n')
         print('Accuracy rate: ',round(accuracy_score(two_test,prediction),2)*100,'%')
         #Accuracy rate
```

```
For K= 1
Confusion matrix:


[[688  61]
 [ 56 758]]


Accuracy rate:  93.0 %
```

- The corresponding confusion matrix has been printed

# Normalization (2 marks)

```
In [17]: scaled = MinMaxScaler()

In [18]: scaled. fit(data_set.drop('class',axis=1))

Out[18]: MinMaxScaler()

In [19]: dset_modified=scaled. transform(data_set.drop('class',axis=1)) #dropping class-feature

In [20]: data_set_feat = pd.DataFrame(dset_modified, columns=data_set.columns[:-1])

In [21]: data_set_feat = np.round(data_set_feat, decimals=2) #rounding all values to 2 decimals

         data_set_feat.head() #dataset_after_normalization

Out[21]:
```

| | e_cblp | e_cp | e_cparhdr | e_maxalloc | e_sp | e_lfanew | NumberOfSections | CreationYear | FH_char0 | FH_char1 | ... | LoaderFlags | sus_sections | non_sus_sectio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.38 | 0.09 | 1.0 | 0.0 | 0.0 | ... | 1.0 | 0.03 | 0.: |
| **1** | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.27 | 0.09 | 1.0 | 0.0 | 0.0 | ... | 1.0 | 0.03 | 0.: |
| **2** | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.41 | 0.12 | 1.0 | 0.0 | 0.0 | ... | 1.0 | 0.03 | 0.! |
| **3** | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.27 | 0.00 | 1.0 | 0.0 | 0.0 | ... | 1.0 | 0.00 | 0. |
| **4** | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.33 | 0.12 | 1.0 | 0.0 | 0.0 | ... | 1.0 | 0.03 | 0.! |

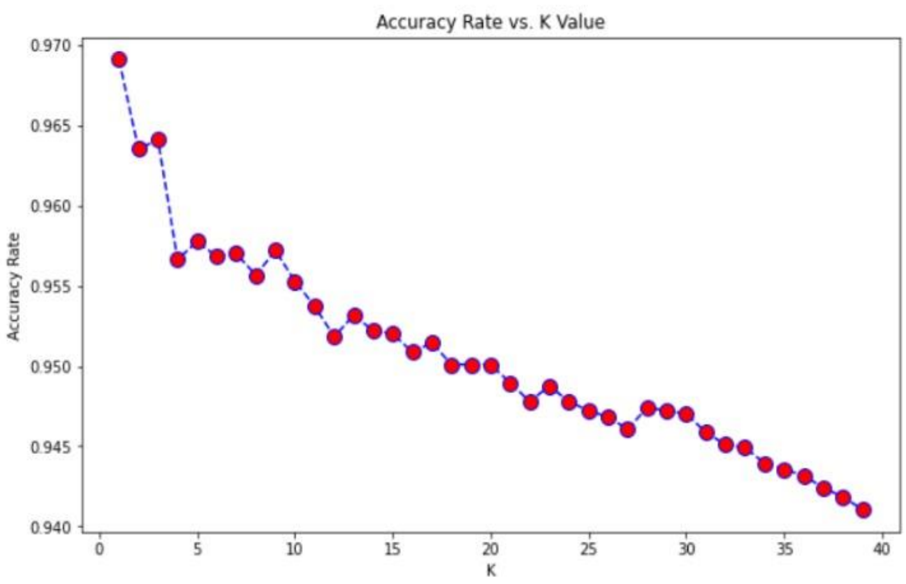
5 rows × 69 columns

```
In [29]: # Computation of accuracy rates for various neighbour values
Accurate_rates = []
for i in range(1,40):
    k_nearest_neighbour = KNeighborsClassifier(n_neighbors=i)
    final_score=cross_val_score(k_nearest_neighbour, data_set_feat,data_set['class'],cv=5)
    Accurate_rates.append(final_score.mean())
print('Accuracy rate: ',round(max(Accurate_rates),2)*100,'%')
```

```
Accuracy rate:  97.0 %
```

```
In [30]: plt.figure(figsize=(10,6))

plt.plot(range(1,40),Accurate_rates, color='blue', linestyle='dashed', marker='o',markerfacecolor='red', markersize=10)

plt.title('Accuracy Rate vs. K Value')

plt.xlabel('K')

plt.ylabel( 'Accuracy Rate')
```

```
Out[30]: Text(0, 0.5, 'Accuracy Rate')
```



# Inference (3 marks)

- Before normalization of the data set the accuracy was 92% , after normalization of data set it increased to 97%.
- We used MinMaxScaler for Normalising the data set which scaled all the features within the range of [0,1].
- The accuracy when compared to KNN classifier after the normalization is increased by 5%
- It is important that during the preprocessing of data , we convert the datatypes to integer for applying the knn algorithm

# Miscellaneous (5 marks)

- Through this project we have got the basic understanding of python libraries like pandas,matplotlib,numpy and seikitlearn
- We learnt how to convert data-types of objects to data types of integer during the pre processing.
- To accomplish the above we have used the label encoder module from the sklearn.preprocessing library.
- We have used .fit_transform method to do so.
- We also learnt how to normalize data through the "Min –Max scaler method which has helped us to improve the accuracy of the model.