Name: Ishan Prabhune

A20538828

In [18]:
```
import numpy as npy
import scipy as spy
import pandas as pds
from IPython.display import display, HTML

# Importing data
DataFrame=pds.read_csv('C:/Users/ishan/Downloads/malware_Binary.csv')
print(DataFrame.shape)

# Striping the column names
DataFrame=DataFrame.rename(columns=lambda x: x.strip())
cols=DataFrame.columns

# Displaying the dataframe as tables in HTML
display(HTML(DataFrame.head(10).to_html()))
```

```
(100000, 36)
```

	hash	millisecond	classi
0	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	0	r
1	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	1	r
2	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	2	r
3	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	3	r
4	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	4	r
5	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	5	r
6	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	6	r
7	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	7	r
8	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	8	r
9	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	9	r

In [19]:
```
# Checking for the missing values
print('ColumnName, DataType, MissingValues')
for i in cols:
    print(i, ',', DataFrame[i].dtype,',',DataFrame[i].isnull().any())
```

```
ColumnName, DataType, MissingValues
hash , object , False
millisecond , int64 , False
classification , object , False
os , object , False
state , int64 , False
usage_counter , int64 , False
prio , int64 , False
static_prio , int64 , False
normal_prio , int64 , False
policy , int64 , False
vm_pgoff , int64 , False
vm_truncate_count , int64 , False
task_size , int64 , False
cached_hole_size , int64 , False
free_area_cache , int64 , False
mm_users , int64 , False
map_count , int64 , False
hiwater_rss , int64 , False
total_vm , int64 , False
shared_vm , int64 , False
exec_vm , int64 , False
reserved_vm , int64 , False
nr_ptes , int64 , False
end_data , int64 , False
last_interval , int64 , False
nvcsw , int64 , False
nivcsw , int64 , False
min_flt , int64 , False
maj_flt , int64 , False
fs_excl_counter , int64 , False
lock , int64 , False
utime , int64 , False
stime , int64 , False
gtime , int64 , False
cgtime , int64 , False
signal_nvcsw , int64 , False
```

In [20]:

```
# Removing the columns which are not required to be included in this classificat

DataFrame=DataFrame.drop('hash',axis=1)
DataFrame=DataFrame.drop('millisecond',axis=1)
DataFrame=DataFrame.drop('state',axis=1)
DataFrame=DataFrame.drop('policy',axis=1)
DataFrame=DataFrame.drop('cached_hole_size',axis=1)
DataFrame=DataFrame.drop('free_area_cache',axis=1)
DataFrame=DataFrame.drop('mm_users',axis=1)
DataFrame=DataFrame.drop('end_data',axis=1)
DataFrame=DataFrame.drop('last_interval',axis=1)
DataFrame=DataFrame.drop('min_flt',axis=1)
DataFrame=DataFrame.drop('maj_flt',axis=1)
DataFrame=DataFrame.drop('fs_excl_counter',axis=1)
DataFrame=DataFrame.drop('lock',axis=1)
DataFrame=DataFrame.drop('utime',axis=1)
DataFrame=DataFrame.drop('stime',axis=1)
DataFrame=DataFrame.drop('gtime',axis=1)
DataFrame=DataFrame.drop('cgtime',axis=1)
# print out and display dataframe as tables in HTML
display(HTML(DataFrame.head(10).to_html()))
```

	classification	os	usage_counter	prio	static_prio	normal_prio	vm_pgoff
0	malware	CentOS	0	3069378560	14274	0	0
1	malware	Windows	0	3069378560	14274	0	0
2	malware	Mac	0	3069378560	14274	0	0
3	malware	Ubuntu	0	3069378560	14274	0	0
4	malware	Mac	0	3069378560	14274	0	0
5	malware	Windows	0	3069378560	14274	0	0
6	malware	Ubuntu	0	3069378560	14274	0	0
7	malware	Mac	0	3069378560	14274	0	0
8	malware	CentOS	0	3069378560	14274	0	0
9	malware	Mac	0	3069378560	14274	0	0

In [21]:
```
# KNN Classification
# Data preprocessing
print('Column Datatypes:\n',DataFrame.dtypes)

# Converting all the nominal variables to binary variables
df_raw=DataFrame.copy(deep=True)
df_knn=DataFrame.copy(deep=True)

# Creating new binary columns
df_dummies=pds.get_dummies(df_knn[['classification','os']], dtype=float)

# Adding them to dataframe
df_knn=df_knn.join(df_dummies)

# Drop the original columns
df_knn=df_knn.drop('classification',axis=1)
df_knn=df_knn.drop('os', axis=1)
print(' Data after dropping orgininal columns:')
display(HTML(df_knn.head(10).to_html()))

# Dropping the extra binary columns, since we only require N-1 binary columns
print(df_knn.columns)
df_knn=df_knn.drop('classification_benign', axis=1)
df_knn=df_knn.drop('os_CentOS', axis=1)
print('Data after dropping binary columns:')
display(HTML(df_knn.head(10).to_html()))
```

```
Column Datatypes:
 classification       object
os                   object
usage_counter         int64
prio                  int64
static_prio           int64
normal_prio           int64
vm_pgoff              int64
vm_truncate_count     int64
task_size             int64
map_count             int64
hiwater_rss           int64
total_vm              int64
shared_vm             int64
exec_vm               int64
reserved_vm           int64
nr_ptes               int64
nvcsw                 int64
nivcsw                int64
signal_nvcsw          int64
dtype: object
 Data after dropping orgininal columns:
```

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	ta
0	0	3069378560	14274	0	0	13173	
1	0	3069378560	14274	0	0	13173	
2	0	3069378560	14274	0	0	13173	
3	0	3069378560	14274	0	0	13173	
4	0	3069378560	14274	0	0	13173	
5	0	3069378560	14274	0	0	13173	
6	0	3069378560	14274	0	0	13173	
7	0	3069378560	14274	0	0	13173	
8	0	3069378560	14274	0	0	13173	
9	0	3069378560	14274	0	0	13173	

```
Index(['usage_counter', 'prio', 'static_prio', 'normal_prio', 'vm_pgoff',
       'vm_truncate_count', 'task_size', 'map_count', 'hiwater_rss',
       'total_vm', 'shared_vm', 'exec_vm', 'reserved_vm', 'nr_ptes', 'nvcsw',
       'nivcsw', 'signal_nvcsw', 'classification_benign',
       'classification_malware', 'os_CentOS', 'os_Debian', 'os_Mac',
       'os_Ubuntu', 'os_Windows'],
      dtype='object')
Data after dropping binary columns:
```

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	ta
0	0	3069378560	14274	0	0	13173	
1	0	3069378560	14274	0	0	13173	
2	0	3069378560	14274	0	0	13173	
3	0	3069378560	14274	0	0	13173	
4	0	3069378560	14274	0	0	13173	
5	0	3069378560	14274	0	0	13173	
6	0	3069378560	14274	0	0	13173	
7	0	3069378560	14274	0	0	13173	
8	0	3069378560	14274	0	0	13173	
9	0	3069378560	14274	0	0	13173	

In [22]:
```
# Import MinMaxScaler Library
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler (feature_range=(0,1))
#cols = ['usage_counter','prio' ,'normal_prio']

# find numeric columns
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
# get column names
cols_numeric = df_knn.select_dtypes(include=numerics).columns.tolist()

df_knn = df_knn[cols_numeric]

df_knn[cols_numeric]=scaler.fit_transform(df_knn)

#display the result
display(HTML(df_knn.head(10).to_html()))
```

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_si
0	0.0	0.18254	0.016007	0.0	0.0	0.199175	
1	0.0	0.18254	0.016007	0.0	0.0	0.199175	
2	0.0	0.18254	0.016007	0.0	0.0	0.199175	
3	0.0	0.18254	0.016007	0.0	0.0	0.199175	
4	0.0	0.18254	0.016007	0.0	0.0	0.199175	
5	0.0	0.18254	0.016007	0.0	0.0	0.199175	
6	0.0	0.18254	0.016007	0.0	0.0	0.199175	
7	0.0	0.18254	0.016007	0.0	0.0	0.199175	
8	0.0	0.18254	0.016007	0.0	0.0	0.199175	
9	0.0	0.18254	0.016007	0.0	0.0	0.199175	

In [23]:
```
# Encoding label, since KNN requires label encoding
from sklearn import preprocessing

y = df_knn['classification_malware'] # define label as nominal values
le = preprocessing.LabelEncoder()
le.fit(y)
y_encoded = le.transform(y) # encode nominal labels to integers

print(y_encoded)

df_knn['classification_malware'] = y_encoded
x_features = df_knn.drop('classification_malware',axis=1)

display(HTML(df_knn.head(10).to_html()))
```

```
[1 1 1 ... 1 1 1]
```

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_si
0	0.0	0.18254	0.016007	0.0	0.0	0.199175	
1	0.0	0.18254	0.016007	0.0	0.0	0.199175	
2	0.0	0.18254	0.016007	0.0	0.0	0.199175	
3	0.0	0.18254	0.016007	0.0	0.0	0.199175	
4	0.0	0.18254	0.016007	0.0	0.0	0.199175	
5	0.0	0.18254	0.016007	0.0	0.0	0.199175	
6	0.0	0.18254	0.016007	0.0	0.0	0.199175	
7	0.0	0.18254	0.016007	0.0	0.0	0.199175	
8	0.0	0.18254	0.016007	0.0	0.0	0.199175	
9	0.0	0.18254	0.016007	0.0	0.0	0.199175	

In [24]:
```
# Performing 10 fold Cross Validation

from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score

#import matplotlib.pyplot as plt

import warnings
# Disable warnings for demonstration purposes
warnings.filterwarnings("ignore")

kvalues = [21, 41, 61, 81, 101, 121, 141, 161, 181, 201]
metrics = ['euclidean','manhattan']

for m in metrics:
    print('metric: ', m)
    for k in kvalues:
    # Create KNN classifier with Brute and the specified k value
        clf = KNeighborsClassifier(n_neighbors=k, algorithm='brute', metric=m)

    # Perform 10-fold cross-validation
        y_pred = cross_val_predict(clf, x_features, y_encoded, cv=10,)

    # Calculate metrics
        acc = cross_val_score(clf, x_features, y_encoded, cv=10, scoring='accura
        precision = precision_score(y_encoded, y_pred)
        recall = recall_score(y_encoded, y_pred)

        y_pred_proba = cross_val_predict(clf, x_features, y_encoded, cv=10, meth
        auc = roc_auc_score(y_encoded, y_pred_proba)

    # Print the result
        print('K =', k, ', Accuracy: ', acc, ', Precision: ', precision, ', Reca
```

```
metric:  euclidean
K = 21 , Accuracy:  0.7825 , Precision:  0.7779308173625595 , Recall:  0.79072 AU
C: 0.8146395738000001
K = 41 , Accuracy:  0.77819 , Precision:  0.7706918361389511 , Recall:  0.79204 A
UC: 0.8204628407999999
K = 61 , Accuracy:  0.77815 , Precision:  0.7753033632242612 , Recall:  0.78332 A
UC: 0.8175734626
K = 81 , Accuracy:  0.78056 , Precision:  0.7807509106192211 , Recall:  0.78022 A
UC: 0.8207923431999999
K = 101 , Accuracy:  0.7777799999999999 , Precision:  0.7788731828768773 , Recal
l:  0.77582 AUC: 0.823303238
K = 121 , Accuracy:  0.7766599999999999 , Precision:  0.7757445281664873 , Recal
l:  0.77832 AUC: 0.8265105684
K = 141 , Accuracy:  0.77176 , Precision:  0.7694427919888955 , Recall:  0.77606
AUC: 0.8263837097999999
K = 161 , Accuracy:  0.76626 , Precision:  0.7607324716020368 , Recall:  0.77686
AUC: 0.829020863
K = 181 , Accuracy:  0.7673699999999999 , Precision:  0.761394521244354 , Recall:
0.7788 AUC: 0.8334569180000001
K = 201 , Accuracy:  0.76545 , Precision:  0.7580842747972855 , Recall:  0.77972
AUC: 0.8375658743999999
metric:  manhattan
K = 21 , Accuracy:  0.80658 , Precision:  0.8057239728759473 , Recall:  0.80798 A
UC: 0.829946129
K = 41 , Accuracy:  0.8074899999999999 , Precision:  0.8035619088988489 , Recall:
0.81396 AUC: 0.8420454288
K = 61 , Accuracy:  0.8039099999999999 , Precision:  0.7985422110454037 , Recall:
0.8129 AUC: 0.8451509006
K = 81 , Accuracy:  0.8044100000000001 , Precision:  0.7989746410261447 , Recall:
0.8135 AUC: 0.850966154
K = 101 , Accuracy:  0.8071400000000001 , Precision:  0.798716203073332 , Recall:
0.82124 AUC: 0.8537129016
K = 121 , Accuracy:  0.80543 , Precision:  0.7950045395715417 , Recall:  0.8231 A
UC: 0.8548580714
K = 141 , Accuracy:  0.804 , Precision:  0.7912770197761766 , Recall:  0.82584 AU
C: 0.8561017524
K = 161 , Accuracy:  0.8040499999999999 , Precision:  0.7882482319258262 , Recal
l:  0.83146 AUC: 0.8575223766
K = 181 , Accuracy:  0.80525 , Precision:  0.7872292376310293 , Recall:  0.83662
AUC: 0.8620265025999999
K = 201 , Accuracy:  0.8007500000000001 , Precision:  0.7786528305383119 , Recal
l:  0.8404 AUC: 0.866212116
```

In [27]:
```
print('The Manhattan distance metric and K=121 achieved highest values for accur
print('The Euclidean distance metric and K=121 achieved lowest values for accura
```

```
The Manhattan distance metric and K=121 achieved highest values for accuracy, pre
cision, recall and AUC.
The Euclidean distance metric and K=121 achieved lowest values for accuracy, prec
ision, recall and AUC.
```

In [28]:
```
print('Testing KNN with Ball Tree')

from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score

import warnings
```

```
# Disable warnings for demonstration purposes
warnings.filterwarnings("ignore")

kvalues = [21, 41, 61, 81, 101, 121, 141, 161, 181, 201]
metrics = ['euclidean','manhattan']

for m in metrics:
    print('metric: ', m)
    for k in kvalues:
    # Create KNN classifier with Ball Tree and the specified k value
        clf = KNeighborsClassifier(n_neighbors=k, algorithm='ball_tree', metric=

    # Perform 10-fold cross-validation
        y_pred = cross_val_predict(clf, x_features, y_encoded, cv=10,)

    # Calculate metrics
        acc = cross_val_score(clf, x_features, y_encoded, cv=10, scoring='accura
        precision = precision_score(y_encoded, y_pred)
        recall = recall_score(y_encoded, y_pred)

        y_pred_proba = cross_val_predict(clf, x_features, y_encoded, cv=10, meth
        auc = roc_auc_score(y_encoded, y_pred_proba)

    # Print the result
        print('K =', k, ', Accuracy: ', acc, ', Precision: ', precision, ', Reca
```

```
Testing KNN with Ball Tree
metric:  euclidean
K = 21 , Accuracy:  0.7825 , Precision:  0.7779308173625595 , Recall:  0.79072 AU
C: 0.8146395738000001
K = 41 , Accuracy:  0.77819 , Precision:  0.7706918361389511 , Recall:  0.79204 A
UC: 0.8204628407999999
K = 61 , Accuracy:  0.77815 , Precision:  0.7753033632242612 , Recall:  0.78332 A
UC: 0.8175734626
K = 81 , Accuracy:  0.78056 , Precision:  0.7807509106192211 , Recall:  0.78022 A
UC: 0.8207923431999999
K = 101 , Accuracy:  0.7777799999999999 , Precision:  0.7788731828768773 , Recal
l:  0.77582 AUC: 0.823303238
K = 121 , Accuracy:  0.7766599999999999 , Precision:  0.7757445281664873 , Recal
l:  0.77832 AUC: 0.8265105684
K = 141 , Accuracy:  0.77176 , Precision:  0.7694427919888955 , Recall:  0.77606
AUC: 0.8263837097999999
K = 161 , Accuracy:  0.76626 , Precision:  0.7607324716020368 , Recall:  0.77686
AUC: 0.829020863
K = 181 , Accuracy:  0.7673699999999999 , Precision:  0.761394521244354 , Recall:
0.7788 AUC: 0.8334569180000001
K = 201 , Accuracy:  0.76545 , Precision:  0.7580842747972855 , Recall:  0.77972
AUC: 0.8375658743999999
metric:  manhattan
K = 21 , Accuracy:  0.80658 , Precision:  0.8057239728759473 , Recall:  0.80798 A
UC: 0.829946129
K = 41 , Accuracy:  0.8074899999999999 , Precision:  0.8035619088988489 , Recall:
0.81396 AUC: 0.8420454288
K = 61 , Accuracy:  0.8039099999999999 , Precision:  0.7985422110454037 , Recall:
0.8129 AUC: 0.8451509006
K = 81 , Accuracy:  0.8044100000000001 , Precision:  0.7989746410261447 , Recall:
0.8135 AUC: 0.850966154
K = 101 , Accuracy:  0.8071400000000001 , Precision:  0.798716203073332 , Recall:
0.82124 AUC: 0.8537129016
K = 121 , Accuracy:  0.80543 , Precision:  0.7950045395715417 , Recall:  0.8231 A
UC: 0.8548580714
K = 141 , Accuracy:  0.804 , Precision:  0.7912770197761766 , Recall:  0.82584 AU
C: 0.8561017524
K = 161 , Accuracy:  0.8040499999999999 , Precision:  0.7882482319258262 , Recal
l:  0.83146 AUC: 0.8575223766
K = 181 , Accuracy:  0.80525 , Precision:  0.7872292376310293 , Recall:  0.83662
AUC: 0.8620265025999999
K = 201 , Accuracy:  0.8007500000000001 , Precision:  0.7786528305383119 , Recal
l:  0.8404 AUC: 0.866212116
```

In [29]:
```
print('The Manhattan distance metric and K=121 achieved highest values for accur
print('The Euclidean distance metric and K=121 achieved lowest values for accura
```

```
The Manhattan distance metric and K=121 achieved highest values for accuracy, pre
cision, recall and AUC.
The Euclidean distance metric and K=121 achieved lowest values for accuracy, prec
ision, recall and AUC.
```

Conlusion and Comparision

In both the Brute and Ball Tree algorithms, The Manhattan distance metric and K=121 achieved highest values for accuracy, precision, recall and

AUC. While the Euclidean distance metric and K=121 achieved lowest values for accuracy, precision, recall and AUC.

The performance trends are similar between both the Brute and Ball Tree algorithms.

In [ ]: