Malware Detection and Classification Using Machine Learning Algorithm

Vishwa Swaroop Sonte
1269907
vsonte@nyit.edu

Jagadish Meka
1271127
jmeka@nyit.edu

Kishore Gorijala
1236424
Kgorij01@nyit.edu

***Abstract*—**

Malware detection refers to the process of detecting the presence of malware on a host system or of distinguishing whether a specific program is malicious or benign. In proposed system implement a framework to detect the malwares. The framework mainly uses two machine learning algorithms. This framework uses large dataset from selected host in the network. To classify whether the given input is malware or not, framework use the SVM algorithm. Then classify the malware type by using the k-Means algorithm

Keywords—detecting, benign, SVM, K-Means algorithm

Content:

I. Introduction

Malware is malicious software that is specifically designed to breach a computer systems security policy and disrupt, damage, or gain unauthorized access. Malware can be classified into different types like virus, trojan horse, adware, spyware, etc. according to the way it exploits the vulnerabilities of the system.

The computer system has turned into an essential section of each association, it is a major test to protect the computer system from harmful activities which deal with the frameworks as well as the information put away inside. Malware is a major issue in present-day innovation, it is a kind of software and this software is specially created so that they can go to its system without the permission of the system user, steal the data of that system, delete the file, and crash the system . Malware Detection is an important consideration for the security of personal computer systems. The Malware detection system is used in any given program to find out if it is malware or not. Malware detection has two functions: first to analyze and secondly detection of programs..

II. Problem Statement

In previous research work, proposed a tool for malware detection. The proposed tool achieved better performance and high accuracy. Even though it has high rate of malware detection but need to improve the major implementation. Such as the present system used small dataset, but in modern computer allocations huge amount data generated. So, the present malware detection approaches not support requirements.

Previous research work does not used data preprocess model to supervised data. So, its leads to delay in malware detection. The feature extraction is works only pre-defined malware types only. So, the main problem of the present system is smaller dataset, data pre-process and feature selection,

III. Related Work

The perception algorithms developed by Dragos Gavrilut et al for-Malware detection using machine learning algorithm. For different algorithms, he achieved the accuracy of 69.90%-96.18%. It should be stated that the algorithms that resulted in best accuracy also produced the highest number of false positives: the most accurate one resulted in 48 false positives. The most balanced algorithm with appropriate accuracy and the low false-positive rate.

Singhal and Raul et al. discussed malware detection model using machine learning algorithms to assist in centralized security in enterprise networks. The proposed detection method based on modified Random Forest algorithm in combination with Information gain for better feature representation. It should be noticed that the data set consists purely of portable executable files, for which feature extraction is generally easier.

In “Zero-day Malware Detection based on supervised Learning Algorithms of API call signatures,” the API function were used for feature representation again. The best result was achieved with Support Vector Machine algorithm with normalized polykernal. The precision of 97.6% was achieved, with a false-positive rate of 0.025 (Alazab, et al. 2011) **[1]**

IV. PROJECT PLAN

Week Name/Date	**Task Name**	**Task Description**	**Status**
15th October,2020	Introduction	Introduction of Malwares Type of Malwares	Completed
22nd October,2020	Literature Review	Detailed discussion about the previous machine learning algorithms	Completed
29th October,2020	Existing System	Discussion & drawbacks about the existing system	Completed
5th Novembe r,2020	Proposed System	Proposed Algorithm Advantages of proposed system	Completed
12th Novembe r,2020	Design Analysis	Design of proposed method implementation Workflow Diagram	Completed
19th Novembe r,2020	Requirement Analysis	Analysis of system requirements Software requirements Hardware requirements	Completed
26th Novembe r,2020	Implementation	Proposed Method Implementation Module wise code implementation Data Selection Data Pre-process Data Clustering Data Classification	Completed
3rd Decembe r,2020	Result Analysis	Compare and Analysis of proposed method performance	Completed
17th Decembe r,2020	Final Report	Presentation, Document, Final Output	Completed

V. PROBLEM SOLUTION

In this research work proposed a solution to overcome the challenges of present system. In proposed system implement specific operations to improve the performance. The proposed system divided into different module such as load large data set, data pre-process and detection. In implementation of proposed system utilize different machine learning algorithms. The proposed system is best solution for challenges of present malware detection system.

VI. ALGORITHMS

SVM: Support Vector Machines is a machine learning algorithm that is generally used for classification problems. The main idea relies on finding such a hyperplane, that would separate the classes in the best way. The distance between the support vector and the hyperplane is referred to as margin. The goal of SVM algorithm is to find such a hyperplane that would result in the maximum margins. SVM’s are generally able to result in good accuracy, especially on clean datasets.

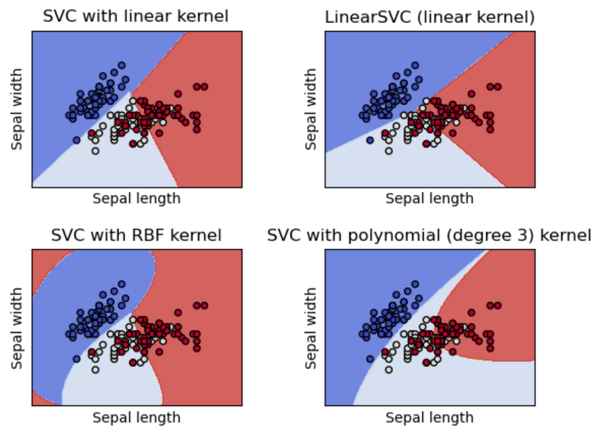


Fig. SVM kernels [8]

KNN: K-Nearest Neighbors is one of the simplest, though, accurate machine learning algorithms. KNN is non-parametric algorithm, meaning that it does not make any assumptions about the data structure. KNN model representation is as simple as the dataset- there is no learning required, the entire training set if stored. It can be used for both classification and regression problems. In both problems, the prediction is based on the k training instances that are closest to the input instance. The drawback of the KNN algorithm is the bad performance on the unevenly distributed datasets. Thus, if one class vastly dominates the other ones, it is more likely to have more neighbours of that class due to their large number, and therefore, make incorrect predictions.

VII. Methodology

Feature representation: From the features presented in the reports generated, we can now think about the way to represent the features to be used for the machine learning algorithms. Since the feature set, containing the failed and successful API's as well as the return codes, is quite large, we have to find a way to present it in a clear, compact and non-redundant way.

Binary representation: The binary representation is the most simple and straightforward way to represent the feature of the failed and successful API calls. Although this approach is simple and straightforward, it does not take into account the return codes generated, as well as a number of times the certain API call was triggered, resulting in lower accuracy.

Combining representation: To utilize the maximum amount of useful data presented in the API calls information, the best approach is to combine the features of the previous representation methods. The usage of the combination method resulted in the dramatic increase in the number of features, since they are represented by the combination of passed API's and failed API's and return codes. Since the feature set became more than two times bigger, some feature selection should be performed.

This dataset is created from a set of APK (application package) files collected from the Opera Mobile Store over the period of January to September of 2014. Just like Windows (PC) systems using an .exe file for installing software, Android uses APK files for installing software on the Android operating system.

[7] [11] The permission system is applied as a measure to restrict access to privileged system resources and is considered as the first barrier to malware. Application developers have to explicitly declare the permissions in the AndroidManifest.xml file contained in the APK.

All official Android permissions are categorized into four types: Normal, Dangerous, Signature and SignatureOrSystem. As dangerous permissions have access to restricted resources and can have a negative impact if used incorrectly, they require user's approval at installation.

A set of APIs will be invoked during the runtime of the application. Each API is associated with a particular permission. When an API call is made, the approval of its associated permission will be checked. The execution of the API could only be successful in the case that the permission is granted by the user. In this way, the permissions are engaged in the process to protect the users private information from unauthorized access. API calls of the Android application exist

in the smali file, which can be obtained by reverse engineering tools such as apktool. The number of critical APIs ranges from a few hundred to thousands.

To be taken as the input of a machine-learning algorithm, permissions/APIs are commonly coded as binary variables i.e., an element in the vector could only take on two values: 1 for a requested permission and 0 otherwise. The number of all possible Android permissions/APIs varies based on the version of the OS. In this task, for each APK file under consideration, we provide a list of APIs obtained by reverse engineering the APK files The class label of the APK file -- +1 if it is regarded as malicious and -1 otherwise -- is determined by the detection results of security appliances hosted by VirusTotal. Note that adware was not counted as malware in our setting. The participants of CDMC 2017 competition are invited to design a classifier that matches this result best.

The dataset consists of API information for 61,730 APK files. The first half (30,897 files) of the dataset is used as training data provided with class labels, and the rest of the data (30,833 files) are used for testing. The total number of features constitutes up to 37,107 unique APIs: including official Android APIs and third-party ones.

Detailed information of the files is listed below:

1) Data file for training:
CDMC2017AndroidAPITrainData.csv, 3.4M,

Each line corresponds to an APK file in the training set. The numbers sorted in ascending order on a line lists up the unique APIs parsed from the APK file.
2) Label file for training:
CDMC2017AndroidAPITrainLabel.csv, 84K,

Each line constrains a number that shows the class label of the corresponding line of the data file. +1 stands for a malware and -1 for a benign file.
3) Data file for testing:
CDMC2017AndroidAPITestData.csv 3.4M,

Each line corresponds to an APK file in the test set. Class label information is not provided.
4) List of all 37,107 API names:
CDMC2017AndroidAPIList.csv

The number in each line before the API names corresponds to the feature numbers in data file 1) and 3).
NOTE: to keep the completeness of the API set, some of them may not be in any of the APK files.

MD5 (CDMC2017AndroidAPITrainData.csv) =
05d39653c618a82fac38c43eb0429b63
MD5 (CDMC2017AndroidAPITrainLabel.csv) =
4f9fd94f4bf5abc44716bbc258c78876
MD5 (CDMC2017AndroidAPITestData.csv) =
3f737a7c2c2e443639efb2e2a9e93aca
MD5 (CDMC2017AndroidAPIList.csv) =
0f9b728afbaf16b11a9f0907420a6b0d

VIII. IMPLENTATION

Stage 1. Loading the large dataset from Indian committee Data Mining Competition.

Stage 2. Data Pre-processing will be done by using the panda's framework. Here then it will be randomly trained for 70% of data and rest will be kept for testing.

Stage 3. Identify the malware threat type before going into these methods, it is essential to understand the basic of two malware analysis approaches: static and dynamic malware analysis. As it implies from the name, static analysis is performed "statically", i.e. without execution of the file. In Contrast, dynamic analysis is conducted on the file while it is being executed for example in virtual machine.

Stage 4. From machine learning perspective, malware detection be a problem of classification or clusterization: unknown malware types should be clustered into several clusters, based on certain properties, identified by the algorithm. On the other hand, having trained a model on the wide dataset of malicious and benign files, we can reduce this problem to classification. For known malware families, this problem can be narrowed down to classification only – having a limited set of classes, to one of which malware sample certainly belongs, it is easier to identify the proper class, and the result would be more accurate than with clusterization algorithms. In this section, the theoretical background is given on all the methods used in this project

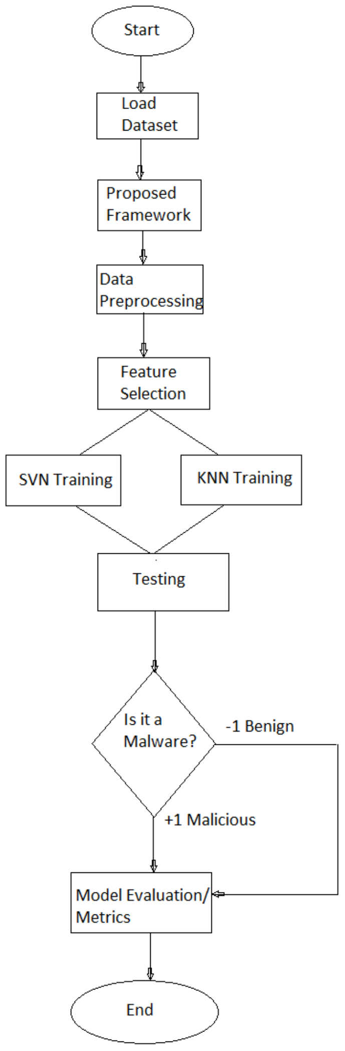
Stage 5. Discussion about the results of the assessment of the implemented machine learning methods. The accuracy of detection is measured as the percentage of correctly identified instances.

Feature selection: The goal of the feature selection is to remove the non-important features from the feature set as it goes too big. Bigger feature sets are harder to operate with but some features in this set might not put any weight on the decision of the algorithm and therefore can be removed. Three general classes of feature selection methods are filtering methods, wrapper methods and embedded methods (Guyon and Elisseef 2006).

Feature extraction: The chosen feature representation method is the combining matrix that includes successful API's, failed API's and their return codes. The reports are used as an input to the feature extraction script which produces the.csv file with the combining inside.

Applying machine learning methods: After the feature were extracted and selected, we can apply the machine learning methods to the data that we obtained. The machine learning methods to be applied, as discussed previously, are KNN, SVM, etc.

IX. FLOW CHART



X. RESULTS AND DISCUSSION

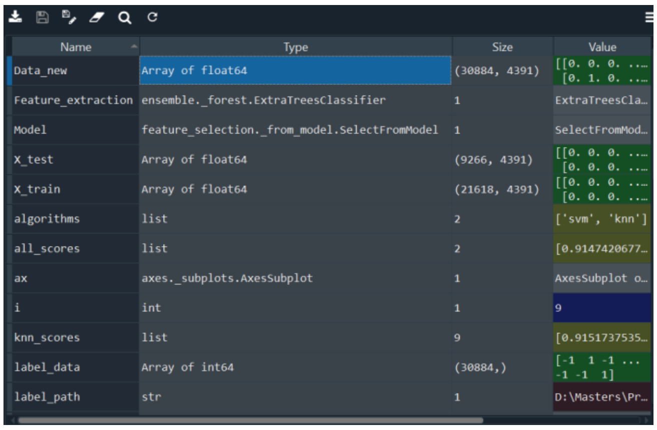
The Sample Train Data need to be separated with each api request and need to be allocated as 1 on api request column. These 1 will be classifier with the android Train label sets and provide the model evaluation graphs.

NAMES	SAMPLES,FEATURES	PERCENTAGE
X_train	(21618,4391)	.70
X_test	(9266,4391)	.30
Y_train	(21618,)	.70
Y_test	(9266,)	.30
new_data	(30884,37107)	
Data_new	(30884,4391)	.12 (features selected)

Table.1. AndroidTrain Data

Names	Value	Y_train	Y_test
Malware	+1 (21%)	(21618,)	(9266,)
Benign	-1 (79%)		
Percentage		.70	.30

Table.2. AndroidLabel Data



```
Name               Type                                         Size           Value
Data_new           Array of float64                             (30884, 4391)  [[0. 0. 0. ...…
                                                                                [0. 1. 0. ...…
Feature_extraction ensemble._forest.ExtraTreesClassifier        1              ExtraTreesCla…
Model              feature_selection._from_model.SelectFromModel 1             SelectFromMod…
X_test             Array of float64                             (9266, 4391)   [[0. 0. 0. ...…
                                                                                [0. 0. 0. ...…
X_train            Array of float64                             (21618, 4391)  [[0. 0. 0. ...…
                                                                                [0. 0. 0. ...…
algorithms         list                                         2              ['svm', 'knn']
all_scores         list                                         2              [0.9147420677…
ax                 axes._subplots.AxesSubplot                   1              AxesSubplot o…
i                  int                                          1              9
knn_scores         list                                         9              [0.9151737535…
label_data         Array of int64                               (30884,)       [-1  1 -1 ...
                                                                               -1 -1  1]
label_path         str                                          1              D:\Masters\Pr…
```

Fig.1 Variable content

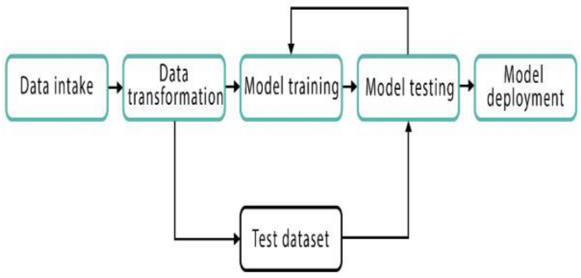
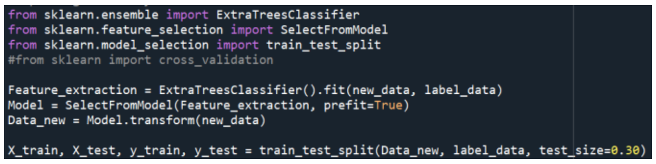


Fig.2 General Workflow process

XI. Analysis

Feature Selection: we have used the ExtraTreesClassifier method from Scikit learn and fit the data into train datasets which has almost removed more than 50% of unwanted data and provided the unique features to train the datasets.



```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split
#from sklearn import cross_validation

Feature_extraction = ExtraTreesClassifier().fit(new_data, label_data)
Model = SelectFromModel(Feature_extraction, prefit=True)
Data_new = Model.transform(new_data)

X_train, X_test, y_train, y_test = train_test_split(Data_new, label_data, test_size=0.30)
```

Fig.3. Feature selection

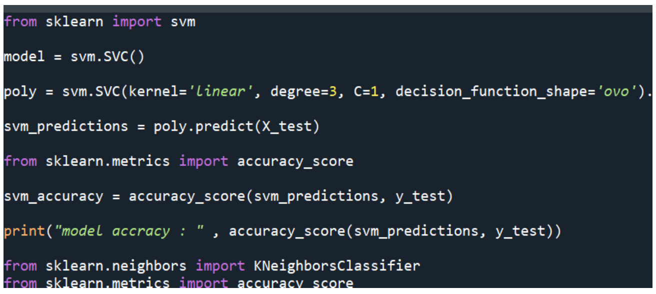
Support Vector Machine:
A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

Formulae:
Given training vectors xi∈Rp, i=1,…, n, in two classes, and a vector y∈{1,−1}n, our goal is to find w∈Rp and b∈R such that the prediction given by sign(wTϕ(x)+b) is correct for most samples.

linear: ⟨x,x′⟩.

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$

$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$
$$\zeta_i \geq 0, i = 1, \ldots, n$$



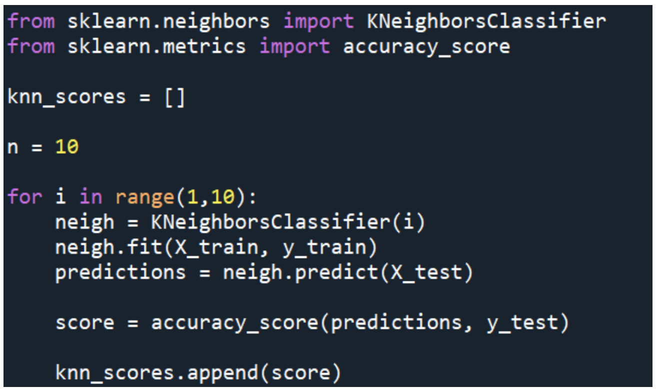
```
from sklearn import svm

model = svm.SVC()

poly = svm.SVC(kernel='linear', degree=3, C=1, decision_function_shape='ovo').

svm_predictions = poly.predict(X_test)

from sklearn.metrics import accuracy_score

svm_accuracy = accuracy_score(svm_predictions, y_test)

print("model accracy : " , accuracy_score(svm_predictions, y_test))

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

Fig.4. SVM Code

KNN Algorithm:
KNN is a distance metric learning algorithm which aims to improve the accuracy of nearest neighbor's classification compared to the standard Euclidean distance. The algorithm directly maximizes a stochastic variant of the leave-one-out k-nearest neighbors (KNN) score on the training set. It can also learn a low-dimensional linear projection of data that can be used for data visualization and fast classification

Formulae: [9]

$$p_{ij} = \frac{\exp(-||Lx_i - Lx_j||^2)}{\sum_{k \neq i} \exp -(||Lx_i - Lx_k||^2)}, \quad p_{ii} = 0$$



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

knn_scores = []

n = 10

for i in range(1,10):
    neigh = KNeighborsClassifier(i)
    neigh.fit(X_train, y_train)
    predictions = neigh.predict(X_test)

    score = accuracy_score(predictions, y_test)

    knn_scores.append(score)
```

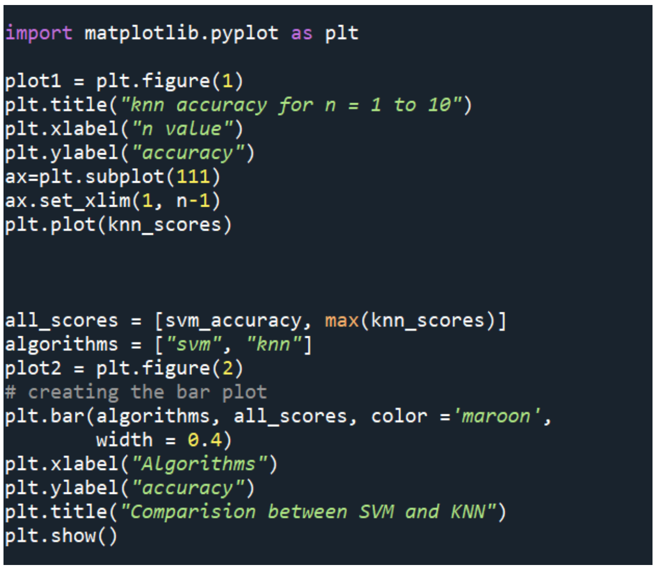
Fig.5. Knn Code

Matplotlib :

In this a collection of functions that make matplotlib work like MATLAB. Each `pyplot` function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

It various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the *axes* part of a figure and not the strict mathematical term for more than one axis).

the pyplot API is generally less-flexible than the object-oriented API. Most of the function calls you see here can also be called as methods from an Axes object [10]

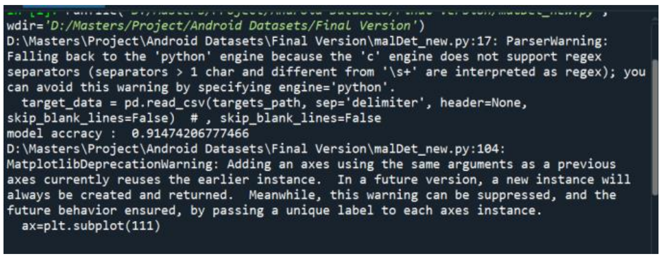


```
import matplotlib.pyplot as plt

plot1 = plt.figure(1)
plt.title("knn accuracy for n = 1 to 10")
plt.xlabel("n value")
plt.ylabel("accuracy")
ax=plt.subplot(111)
ax.set_xlim(1, n-1)
plt.plot(knn_scores)



all_scores = [svm_accuracy, max(knn_scores)]
algorithms = ["svm", "knn"]
plot2 = plt.figure(2)
# creating the bar plot
plt.bar(algorithms, all_scores, color ='maroon',
        width = 0.4)
plt.xlabel("Algorithms")
plt.ylabel("accuracy")
plt.title("Comparision between SVM and KNN")
plt.show()
```

Fig.6. Plot graphs

XII. Graphs

SVM: The algorithm that was tested was Support Vector Machines. The result of the predictions can be outlined in Figure 25. The overall accuracy achieved was 91.4% for binary classification



```
wdir='D:/Masters/Project/Android Datasets/Final Version')
D:\Masters\Project\Android Datasets\Final Version\malDet_new.py:17: ParserWarning:
Falling back to the 'python' engine because the 'c' engine does not support regex
separators (separators > 1 char and different from '\s+' are interpreted as regex); you
can avoid this warning by specifying engine='python'.
  target_data = pd.read_csv(targets_path, sep='delimiter', header=None,
skip_blank_lines=False)  # , skip_blank_lines=False
model accracy :  0.91474206777466
D:\Masters\Project\Android Datasets\Final Version\malDet_new.py:104:
MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous
axes currently reuses the earlier instance.  In a future version, a new instance will
always be created and returned.  Meanwhile, this warning can be suppressed, and the
future behavior ensured, by passing a unique label to each axes instance.
  ax=plt.subplot(111)
```

Fig.7. SVM Accuracy

K nearest Neighbour:

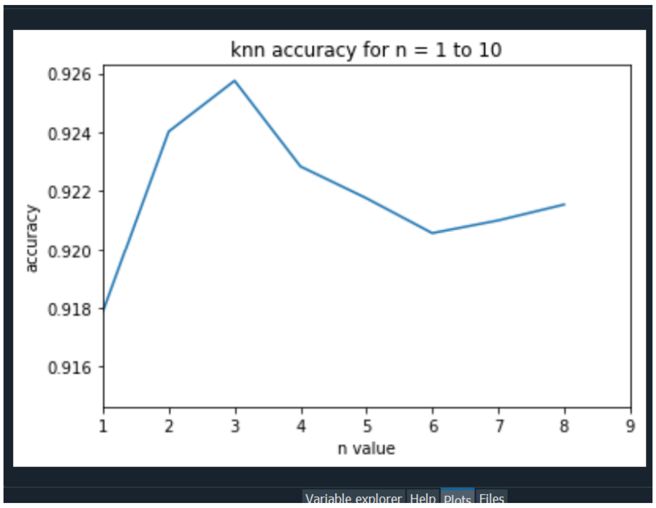


Fig.8. KNN Graphs

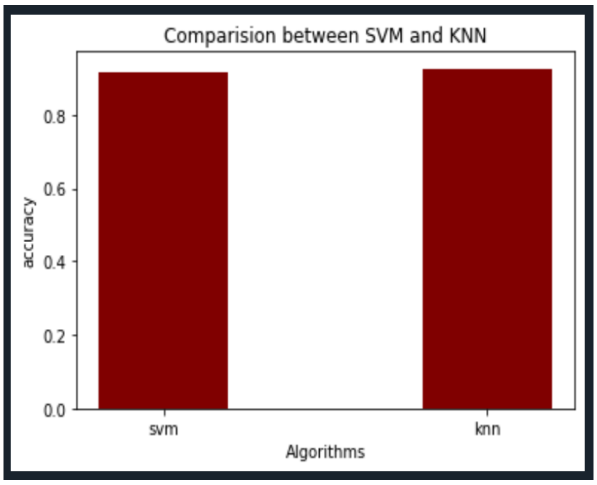


Fig.9 Comparison

XIII. Conclusion

Overall, the goals defined for this study were achieved. The desired feature extraction and representation methods were selected, and the selected machine learning algorithms were applied and evaluated.

The desired feature representation method was selected to be the combined matrix, outlining the frequency of successful and failed API calls along with the return codes for them. This was chosen, because it outlines the actual behavior of the file. Unlike other methods, it combines information about different changes in the system, including the changes in the registry, mutexes, files, etc.

Based on the results described before, it is recommended to implement the classification based on the K Nearest Neighbour, as it resulted in the best accuracy and high performance. Although this method achieved the highest result for binary classification as well, it is recommended to consider implementing Support Vector Machines instead. This is because this method resulted in 0 false-negatives, i.e. no malware samples were classified as benign. Although in the binary problem accuracy is still the main 73 concern, the number of false-negatives is an important factor as well, since they can result in massive infections.

XIV. Future Work

The study performed in this project was a proof-of-concept. Therefore, several future improvements related to the practical implementation of this project can be identified.

Depending on accuracy we can state that it will be very useful to predict the appropriate algorithm for such type of datasets. Additionally, we can even use the same code for different datasets but need to do a bit of feature selection and training before use.

Use a wider dataset:
Although the dataset that was used in this study is broad, covering most of the malware types that are relevant to the modern world, it does not cover all possible types. Collecting a malware dataset is a tedious task that requires a lot of time and effort. For more accurate evaluation of the predictors

Use pre-selected:
APIs In this work, the big overhead in the data processing was created by the need of selecting the relevant API calls and removing the redundant ones. For further implementation, only the APIs that were identified as relevant in this study can be used. This will decrease the amount of time required for data preprocessing, reduce the performance requirements of the machine on which the analysis is being done and decrease the level of feature selection to be made. However, it should be noted, that for more accurate description, the relevant APIs should be extracted from the biggest possible dataset. Also, it is advised to select the relevant APIs per malware family, as this will result in another level of flexibility and accuracy

XV. Appendices

Written Language: Python

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 14 18:24:27 2020


@author: Swaroop Sonte
"""


import pandas as pd
import numpy as np


#path to load malware data from .cvs file
targets_path = r"D:\Masters\Project\Android Datasets\Final Version\CDMC2017_T1_AndroidAPITrainData.csv"


label_path = r"D:\Masters\Project\Android Datasets\Final Version\CDMC2017_T1_AndroidAPITrainLabel.csv"


#reading data
target_data = pd.read_csv(targets_path, sep='delimiter', header=None, skip_blank_lines=False) # , skip_blank_lines=False


label_data = pd.read_csv(label_path, header= None)


no_of_samples,_ = target_data.shape
#no_of_samples = 2000  # max 30875
target_data = np.asarray(target_data[0].iloc[0: no_of_samples])


label_data = np.asarray(label_data[0].iloc[0:no_of_samples])


#target_data = np.asarray(target_data)


target_data3 = []
for sample in target_data:
    #print(type(sample))
    #print(sample)
    target_data2= list([])
    target_data3.append(target_data2)
    for i in sample.split(" "):
        #print(i)
        target_data2.append(i)

    target_data3 = np.asarray(target_data3)

    new_data = np.zeros((no_of_samples,37107))

    x = 0

    while x < no_of_samples:

      for sample in target_data3:
        #print(type(sample))

        for i in sample:

          new_data[x, (int(i)-1)]=1

        x += 1

    #importing necessary functions from sklearn
    from sklearn.ensemble import ExtraTreesClassifier
    from sklearn.feature_selection import SelectFromModel
    from sklearn.model_selection import train_test_split
    #from sklearn import cross_validation

    Feature_extraction = ExtraTreesClassifier().fit(new_data, label_data)
    Model = SelectFromModel(Feature_extraction, prefit=True)
    Data_new = Model.transform(new_data)

    X_train, X_test, y_train, y_test = train_test_split(Data_new, label_data, test_size=0.30)

    from sklearn import svm

    model = svm.SVC()

    poly = svm.SVC(kernel='linear', degree=3, C=1, decision_function_shape='ovo').fit(X_train, y_train)

    svm_predictions = poly.predict(X_test)

    from sklearn.metrics import accuracy_score

    svm_accuracy = accuracy_score(svm_predictions, y_test)

    print("model accracy : " , accuracy_score(svm_predictions, y_test))

    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import accuracy_score

    knn_scores = []

    n = 10

    for i in range(1,10):
      neigh = KNeighborsClassifier(i)
      neigh.fit(X_train, y_train)
      predictions = neigh.predict(X_test)

      score = accuracy_score(predictions, y_test)

      knn_scores.append(score)

    import matplotlib.pyplot as plt

    plot1 = plt.figure(1)
    plt.title("knn accuracy for n = 1 to 10")
    plt.xlabel("n value")
    plt.ylabel("accuracy")
    ax=plt.subplot(111)
    ax.set_xlim(1, n-1)
    plt.plot(knn_scores)



    all_scores = [svm_accuracy, max(knn_scores)]
    algorithms = ["svm", "knn"]
    plot2 = plt.figure(2)
    # creating the bar plot
    plt.bar(algorithms, all_scores, color ='maroon',
            width = 0.4)
    plt.xlabel("Algorithms")
    plt.ylabel("accuracy")
    plt.title("Comparision between SVM and KNN")
```

```
plt.show()

#np.savetxt("new_data1.csv", new_data, delimiter=",")

#new_d =
pd.read_csv(r"U:\MalwareData.csv\new_data1.csv",
header=None)
```

XVI. References

[1]. Kateryna Chumachenko. (2017). Machine Learning Methods for Malware Detection and Classification, Kaakkois-Suomen ammattikorkeakoulu.

[2]. Daniel Gibert, Carles Mateu, Jordi Planes. (2020) , The rise of machine learning for detection and classification of malware, Elsevier Ltd.

[3]. https://archive.ics.uci.edu/ml/datasets/Adult

[4]. https://www.kaggle.com/mlg-ulb/creditcardfraud

[5]. https://docs.python.org/3/library/tkinter.html

[6].https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library//

[7] http://archive.csmining.org/

[8] https://scikit-learn.org/stable/modules/svm.html

[9] https://scikit-learn.org/stable/modules/neighbors.html

[10] https://matplotlib.org/tutorials/introductory/pyplot.html

[11] Tao Ban, Takeshi Takahashi, Shanqing Guo, Daisuke Inoue, Koji Nakao, Integration of Multi-modal Features for Android Malware Detection Using Linear SVM, The 11th Asia Joint Conference on Information Security (ASIAJCIS 2016), Fukuoka, Japan, Aug. 2016.