COMPUTER NETWORKS (SE-304a)

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly DCE)

<u>PROJECT REPORT</u>

Malware Classification using Machine Learning and Deep Learning Techniques

Submitted to:
Irfan Alam

Submitted by:
Aryan Bansal - 2K18 / SE / 038
Ashish Kumar - 2K18 / SE / 041

Table of Contents

Abstract

A Malware is a malicious software, program or a script which can be harmful to the user's computer. We are facing a malware crisis due to various types of malicious programs or scripts available in the huge virtual world - the Internet. These malicious programs can perform a variety of functions, including stealing, encrypting or deleting sensitive data, altering or hijacking core computing functions and monitoring users' computer activity without their permission. In this research, we classified programs as malwares or legitimate using different algorithms such as Logistic Regression, SVM (Support vector machine), XGBoost, Random forest and ANN (Artificial Neural Network). Moreover, we developed a voting classifier using three classifiers with maximum accuracy. The maximum accuracy achieved was 99.52%. We also deployed the web-app made using streamlit and NGROK for public use.

Introduction

The Problem Statement

The most common defenses against malware threats involve the use of signatures derived from instances of known malware. But now we are encountering more and more malware coming from sophisticated developers that evade signature detection. Thus, all good antimalware software these days must employ some sort of heuristic algorithms to prevent zero day attacks.

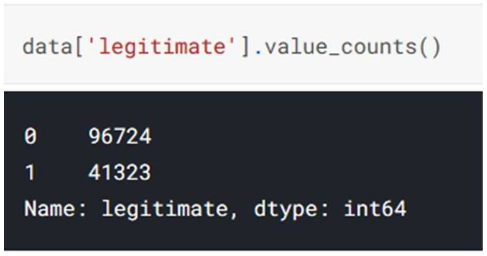
Proposed Solution

The main idea is to extract important features by performing static analysis techniques on the malwares and benign files. We will then compare the performance of various machine algorithms like Support Vector Machines, Random Forest, Logistic regression, AdaBoost, XG boost and Artificial Neural Networks to differentiate legitimate files (benign) from malware. We then develop a voting classifier which will take the majority vote of all the algorithms as the final verdict. Finally, we will deploy the web app for public use.

Preliminaries

Data Acquisition and Storage

The dataset consists of legitimate and malicious files. For legitimate file, we gathered 41323 Windows binaries (exe + dll) and for malicious files, we gathered 96724 different PE files from a part of Virus Share collection: 134th archive. The legitimate files belong to Windows XP, Windows 7 and Windows 10.



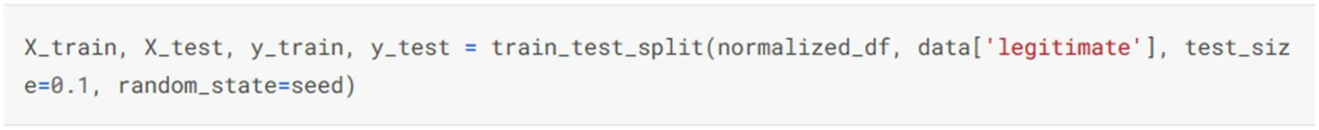
```
data['legitimate'].value_counts()
```

```
0    96724
1    41323
Name: legitimate, dtype: int64
```

We have standardized the data by subtracting each value from the mean value and dividing it by standard deviation.

$$X' = \frac{X - \mu}{\sigma}$$

This way, the data has a mean of 0 and standard deviation of 1.
Furthermore, we have split the data into train and test sets with 0.1 as the split ratio.



```
X_train, X_test, y_train, y_test = train_test_split(normalized_df, data['legitimate'], test_siz
e=0.1, random_state=seed)
```

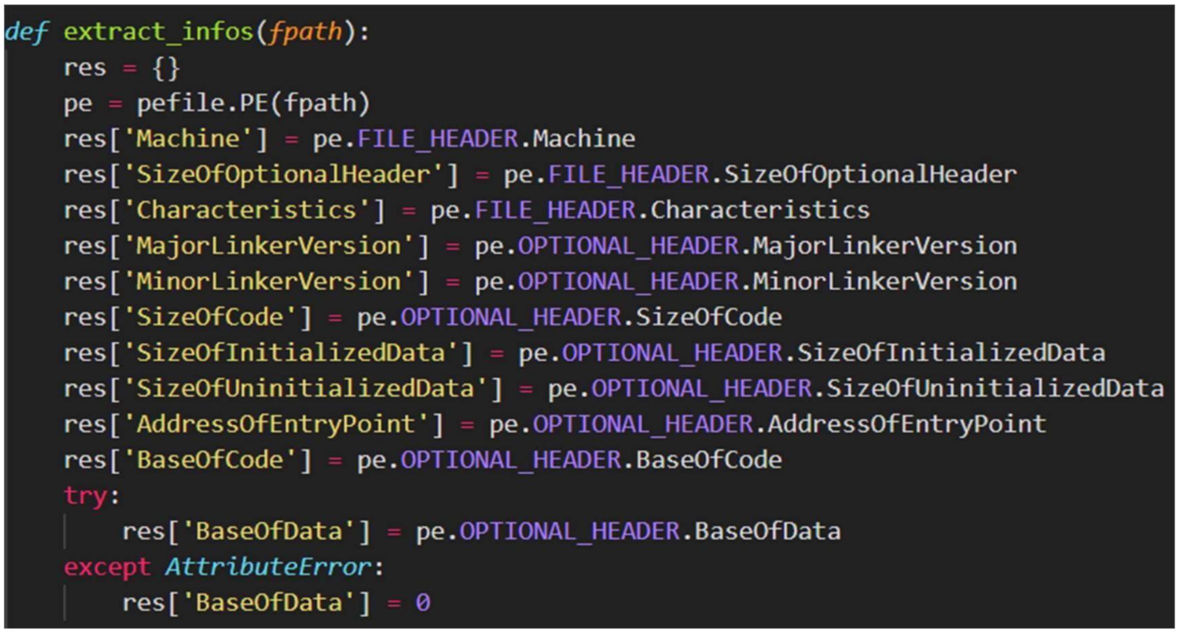
Proposed Technique

Feature Extraction

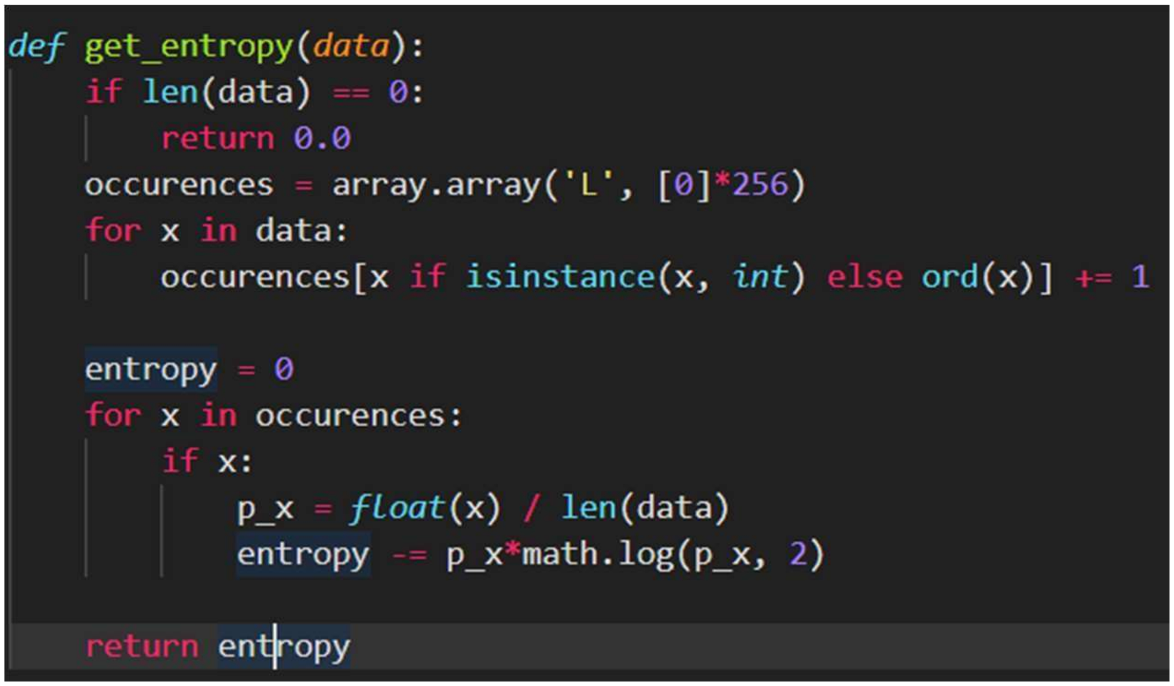
PE file header: Python

It is a python module used to work with portable executable files. Windows PE files can be either executable files or files that contain binary code used by other executable files. We parse the PE fileaccording to the PE structure and extract the features from the parsed file. PE files have a header that contains information about portable executable files. A portable executable can import various functions from different PE files to avoid rewriting the code.

Below we have attached some code snippets which we used to extract data. We have also used entropy as a feature for our extracted data. Similarly, we have extracted 57 features from the given files.



```python
def extract_infos(fpath):
    res = {}
    pe = pefile.PE(fpath)
    res['Machine'] = pe.FILE_HEADER.Machine
    res['SizeOfOptionalHeader'] = pe.FILE_HEADER.SizeOfOptionalHeader
    res['Characteristics'] = pe.FILE_HEADER.Characteristics
    res['MajorLinkerVersion'] = pe.OPTIONAL_HEADER.MajorLinkerVersion
    res['MinorLinkerVersion'] = pe.OPTIONAL_HEADER.MinorLinkerVersion
    res['SizeOfCode'] = pe.OPTIONAL_HEADER.SizeOfCode
    res['SizeOfInitializedData'] = pe.OPTIONAL_HEADER.SizeOfInitializedData
    res['SizeOfUninitializedData'] = pe.OPTIONAL_HEADER.SizeOfUninitializedData
    res['AddressOfEntryPoint'] = pe.OPTIONAL_HEADER.AddressOfEntryPoint
    res['BaseOfCode'] = pe.OPTIONAL_HEADER.BaseOfCode
    try:
        res['BaseOfData'] = pe.OPTIONAL_HEADER.BaseOfData
    except AttributeError:
        res['BaseOfData'] = 0
```



```python
def get_entropy(data):
    if len(data) == 0:
        return 0.0
    occurences = array.array('L', [0]*256)
    for x in data:
        occurences[x if isinstance(x, int) else ord(x)] += 1

    entropy = 0
    for x in occurences:
        if x:
            p_x = float(x) / len(data)
            entropy -= p_x*math.log(p_x, 2)

    return entropy
```

Final Features Extracted

We extracted a total of 57 features from the given files.

```
print(data.columns.values)
```

```
['Name' 'md5' 'Machine' 'SizeOfOptionalHeader' 'Characteristics'
 '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' 'SectionsNb' 'SectionsMeanEntropy'
 'SectionsMinEntropy' 'SectionsMaxEntropy' 'SectionsMeanRawsize'
 'SectionsMinRawsize' 'SectionMaxRawsize' 'SectionsMeanVirtualsize'
 'SectionsMinVirtualsize' 'SectionMaxVirtualsize' 'ImportsNbDLL'
 'ImportsNb' 'ImportsNbOrdinal' 'ExportNb' 'ResourcesNb'
 'ResourcesMeanEntropy' 'ResourcesMinEntropy' 'ResourcesMaxEntropy'
 'ResourcesMeanSize' 'ResourcesMinSize' 'ResourcesMaxSize'
 'LoadConfigurationSize' 'VersionInformationSize' 'legitimate']
```

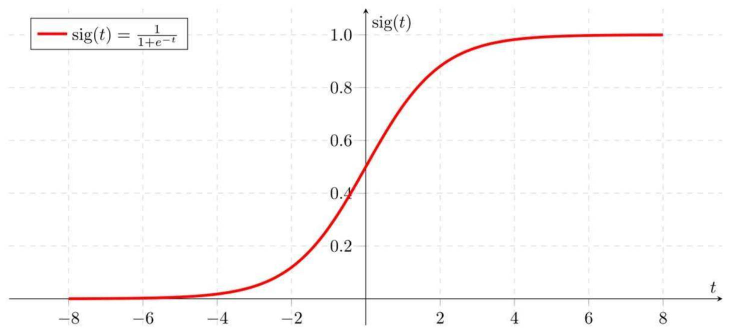
To further improve the performance, we used Random Forest feature selection algorithm to detect the important features and the degree to which they affect the probability of whether the file belongsto legitimate or malicious class.

```
1. feature VersionInformationSize (0.123376)
2. feature ResourcesMinSize (0.112341)
3. feature SizeOfInitializedData (0.084510)
4. feature LoaderFlags (0.083573)
5. feature MinorSubsystemVersion (0.073866)
6. feature MajorLinkerVersion (0.073776)
7. feature MinorImageVersion (0.044605)
8. feature SectionsMaxEntropy (0.040771)
9. feature SizeOfHeapCommit (0.037903)
10. feature NumberOfRvaAndSizes (0.035179)
11. feature LoadConfigurationSize (0.034461)
12. feature Characteristics (0.023391)
13. feature SizeOfCode (0.020393)
14. feature MajorSubsystemVersion (0.018924)
```

Machine Learning Models and their Architectures

Logistic Regression

It is the most basic algorithm used in binary classification. It calculates the probability of a filebelonging to one of the 2 classes.



Loss Function:

$$\text{Log Loss} = \sum_{(x,y)\in D} -y\log(y') - (1-y)\log(1-y')$$

where:

- $(x, y) \in D$ is the data set containing many labeled examples, which are (x, y) pairs.
- y is the label in a labeled example. Since this is logistic regression, every value of y must either be 0 or 1.
- y' is the predicted value (somewhere between 0 and 1), given the set of features in x.

Here we have used the "saga" solver, as it is a great option for large datasets along with "sag" solverwhere saga is also a great choice for sparse data. The drawback of "saga" is that it requires scaled dataset (normalized data), which was handled in preprocessing.

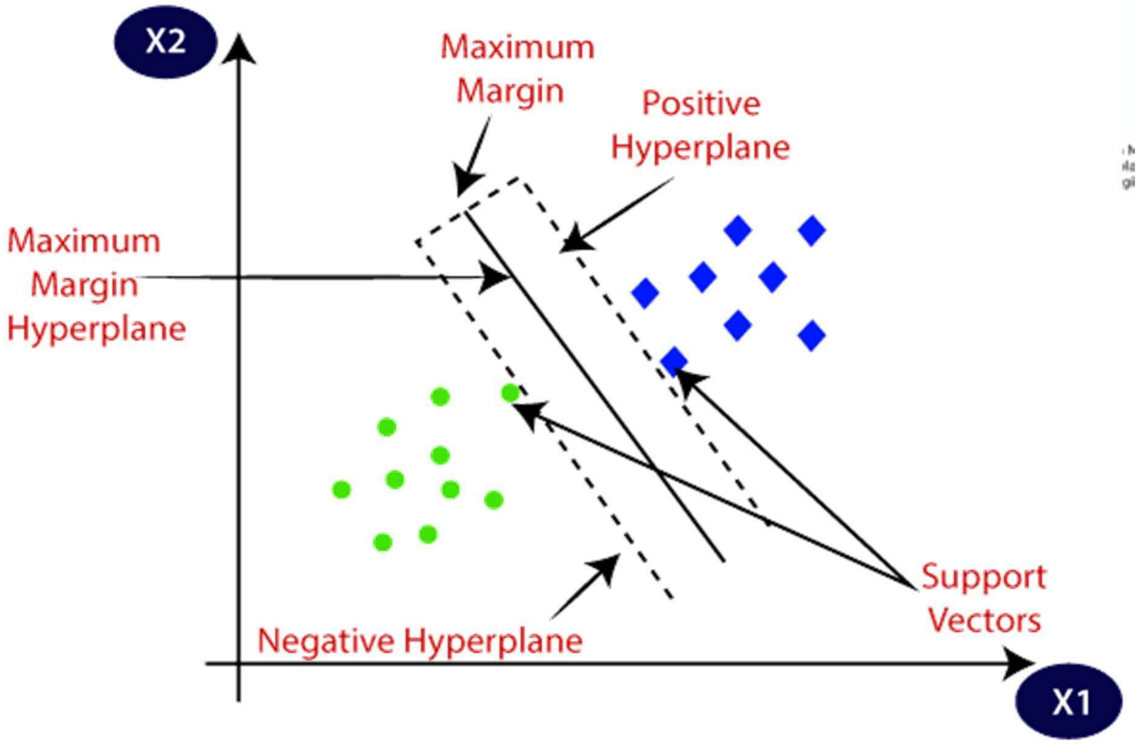
```
model_lr = LogisticRegression(random_state=101,solver='saga',max_iter=500)
model_lr.fit(X_train,y_train)
lr_pred = model_lr.predict(X_test)
```

Supply Vector Machine (SVM)

Support Vector Machines, predominantly a classification algorithm that can be utilised for both classification and regression analysis, uses the concept of a hyperplane to extrapolate the data to extended dimensions using multiple kernels. The algorithm based upon the concepts of width of theprediction margin which consequently depends on the support vectors in the dataset.

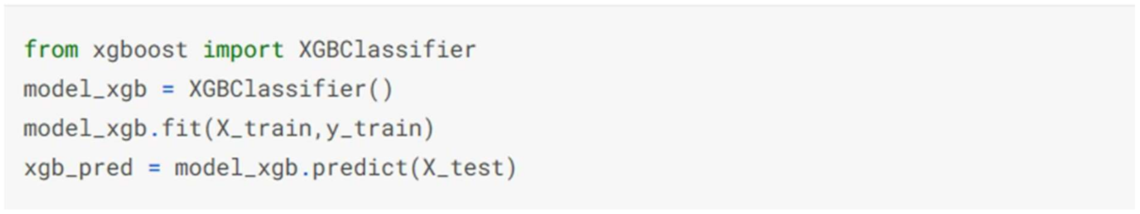
SVM's are a really powerful class of algorithms and usually provide a good accuracy, which was alsoobserved in our analysis.

```
model_lsvc = LinearSVC(random_state=101)
model_lsvc.fit(X_train,y_train)
lsvc_pred = model_lsvc.predict(X_test)
```

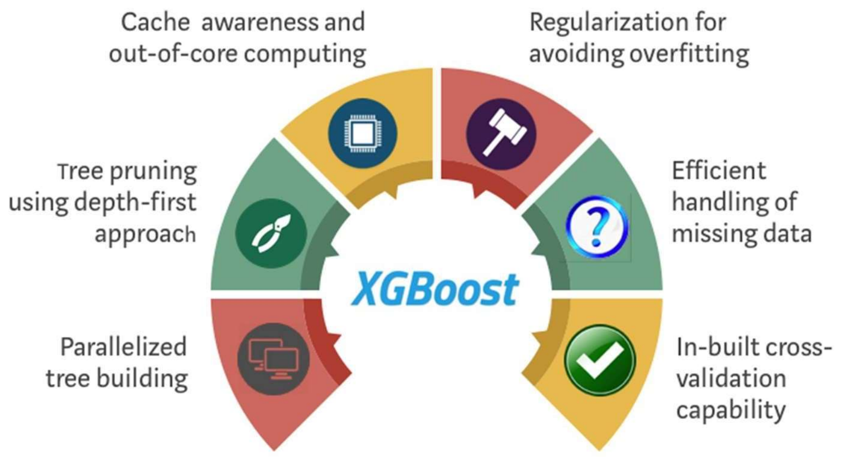


Extreme Gradient Boosting

Extreme Gradient Boosting or XGBoost is an infamous machine learning algorithm that was put forward by Chen T, and Guestrin C in 2016 that took the field of machine learning by a surprise as the algorithm has now become one of the most widely used algorithms in the field with the achievability of very high accuracy.[1] XG Boost is a tree based ensemble, Gradient Boosting Algorithm, which hence utilizes the capabilities of both the ensemble techniques of bagging and boosting. XG Boost is better than gradient boosting in terms of the optimized hardware utilization and the robust software, which again proved its capability by outperforming all the algorithms mentioned alongside Random Forests, where RF and XGBoost had comparable accuracies.



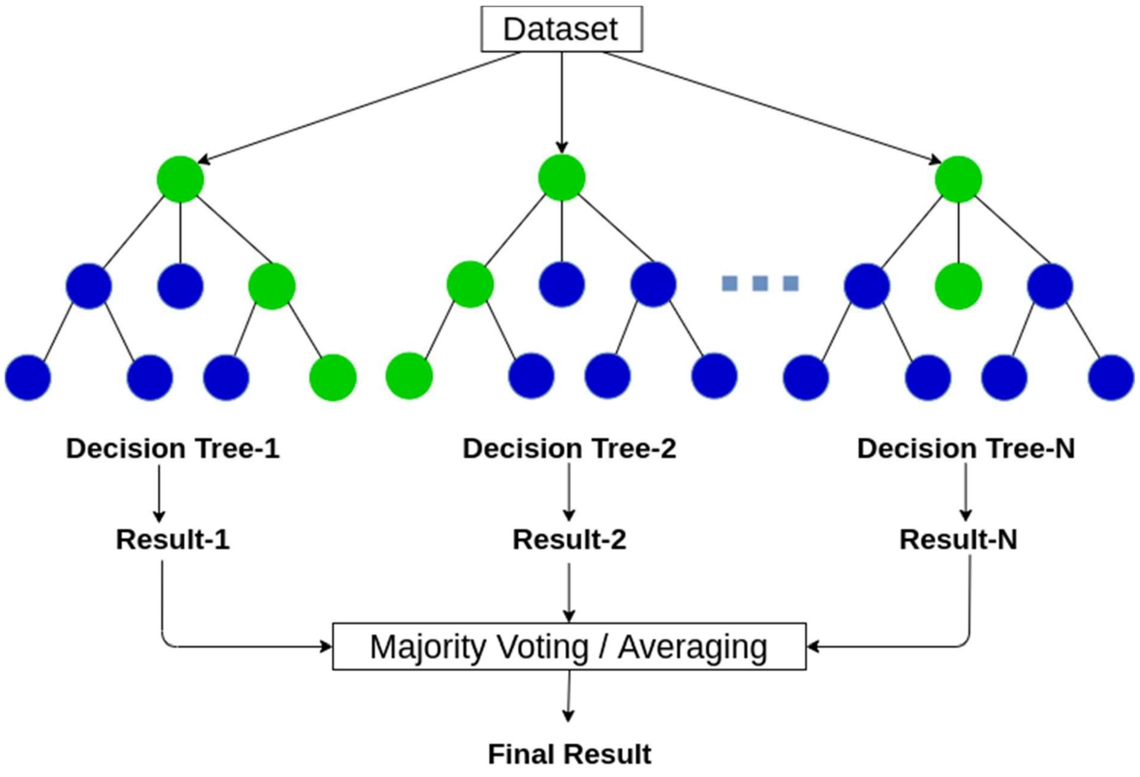
```
from xgboost import XGBClassifier
model_xgb = XGBClassifier()
model_xgb.fit(X_train,y_train)
xgb_pred = model_xgb.predict(X_test)
```


Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient handling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

Random Forest

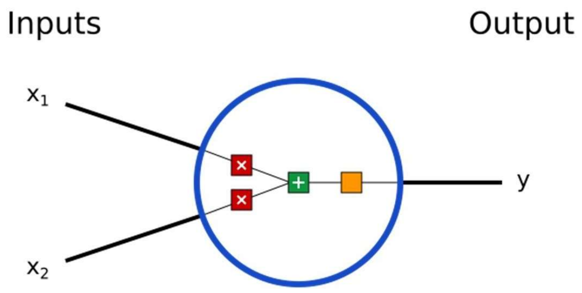
The random forest is another very famous ensembling technique that utilizes the randomized selection of features across multiple classification trees, which aids in the inclusion of all of the features without exceptional computational power and the problem of overfitting. Random Forests are historically proven to give excellent results in medium sized datasets (about 20k-80k) which has also been realized by our instance of the forest, which outperformed all the algorithms in mention alongside XG Boost, where RF and XGBoost had comparable accuracies.

```
model_rf = RandomForestClassifier(random_state=101)
model_rf.fit(X_train,y_train)
rf_pred = model_rf.predict(X_test)
```



Artificial Neural Network

A simple artificial neural network consists of various hidden layers where each layer has some number of neurons. Unlike machine learning algorithms, neural networks arrange themselves in such a way such that they can make accurate decisions by themselves without the need of human intervention. In an artificial neural network, the data passes through several layers of interconnected nodes, wherein each node classifies the characteristics and information of the previous layer before passing the results on to other nodes in subsequent layers.



Architecture

In our model, we have used keras framework to develop the architecture. We have used a dense network containing 1024 neurons with relu activation followed by batch normalization. The batch normalization layer is followed by a dense layer with 1 neuron and sigmoid activation to predict the probability of the file belonging to one of the 2 classes. We have used adaptive momentum as our optimizer and binary cross entropy as the loss function. We have used a batch size of 32 and trained on 50 epochs with validation split ratio as 0.1.

```
model=Sequential()
model.add(Dense(units = 1024, kernel_initializer = 'normal', activation = 'relu', input_dim =X_
train.shape[1]))
model.add(BatchNormalization())
model.add(Dense(units = 1, kernel_initializer = 'normal', activation = 'sigmoid'))
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
model.summary()
```

Hyper-parameter tuning

To avoid overfitting of data, we have used various callbacks: Reduce Learning Rate on Plateau, Model Checkpoint and Early Stopping.

```
rlrplateau = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1, patience=5, verbose=1)
checkpoint = ModelCheckpoint('mymodel.h5', verbose=1, save_best_only=True)
earlystopping = EarlyStopping(monitor='val_accuracy',restore_best_weights=True, patience=10,ver
bose=1)
```

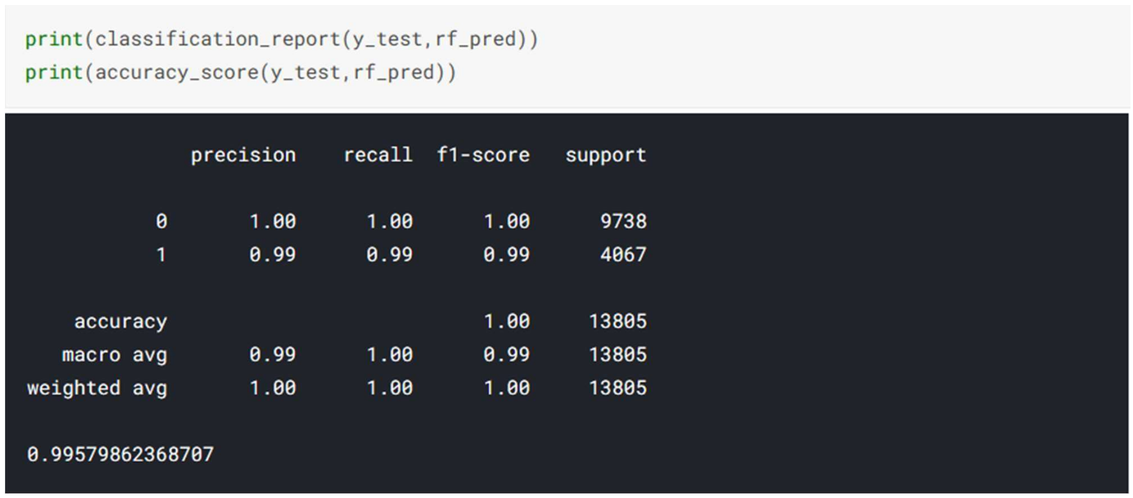
We monitor validation accuracy to avoid overfitting and reduce the learning rate by a factor of 10 if our validation accuracy does not improve after patience number of epochs. Finally, we will restore the weights where validation accuracy was highest.

Voting Classifier (Novel Idea)

We finally propose the usage of a Voting Classifier, an aggregation of the algorithms that performed exceptionally well on the dataset to account for any outliers or discrepancies in any one of the classifiers and uses a majority vote-based approach to classify the file to be malware or legitimate. We utilized 3 algorithms to formulate the voting classifier viz. Random Forests, XGBoost and Artificial neural network, all of which scored an accuracy of above 99%. It can generally be constructed from n number of algorithms where every algorithm to be given the weight in respect to its accuracy.

Results

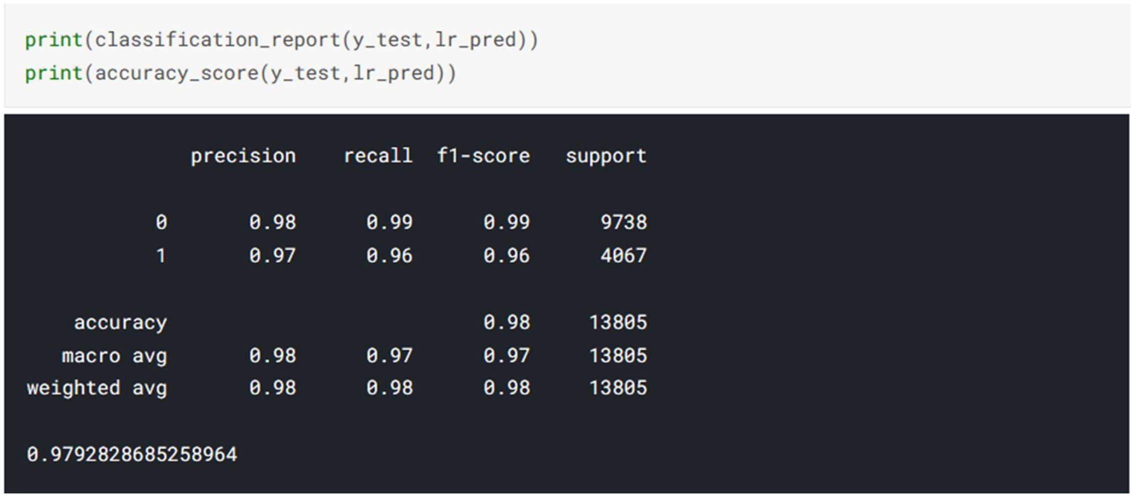
We have trained the dataset on 5 different algorithms and have developed a voting classifier which will take the majority vote to give the final verdict as legitimate or malicious.
Results of all the classifiers are given below:



```
print(classification_report(y_test,rf_pred))
print(accuracy_score(y_test,rf_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9738
           1       0.99      0.99      0.99      4067

    accuracy                           1.00     13805
   macro avg       0.99      1.00      0.99     13805
weighted avg       1.00      1.00      1.00     13805

0.99579862368707
```

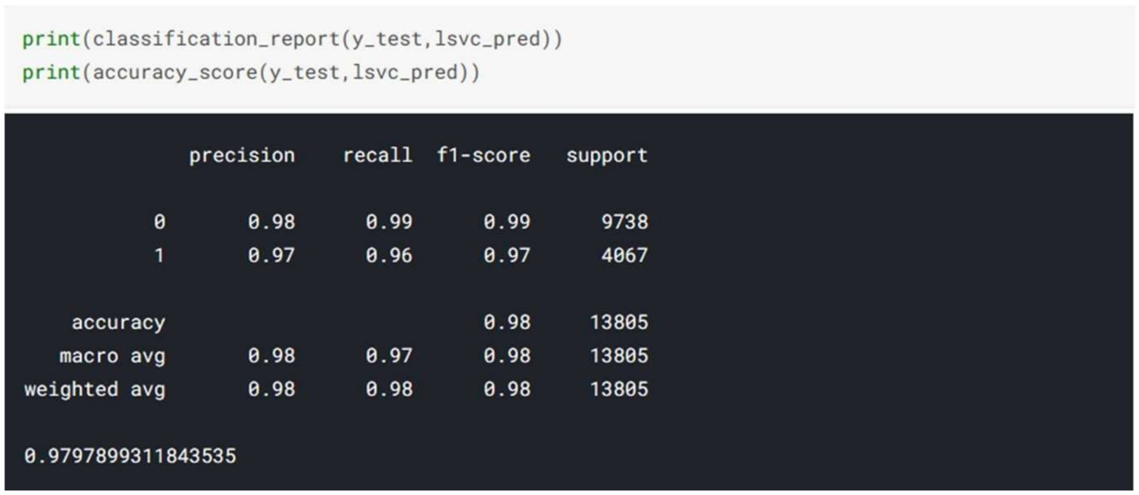
Random Forest



```
print(classification_report(y_test,lr_pred))
print(accuracy_score(y_test,lr_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      9738
           1       0.97      0.96      0.96      4067

    accuracy                           0.98     13805
   macro avg       0.98      0.97      0.97     13805
weighted avg       0.98      0.98      0.98     13805

0.9792828685258964
```

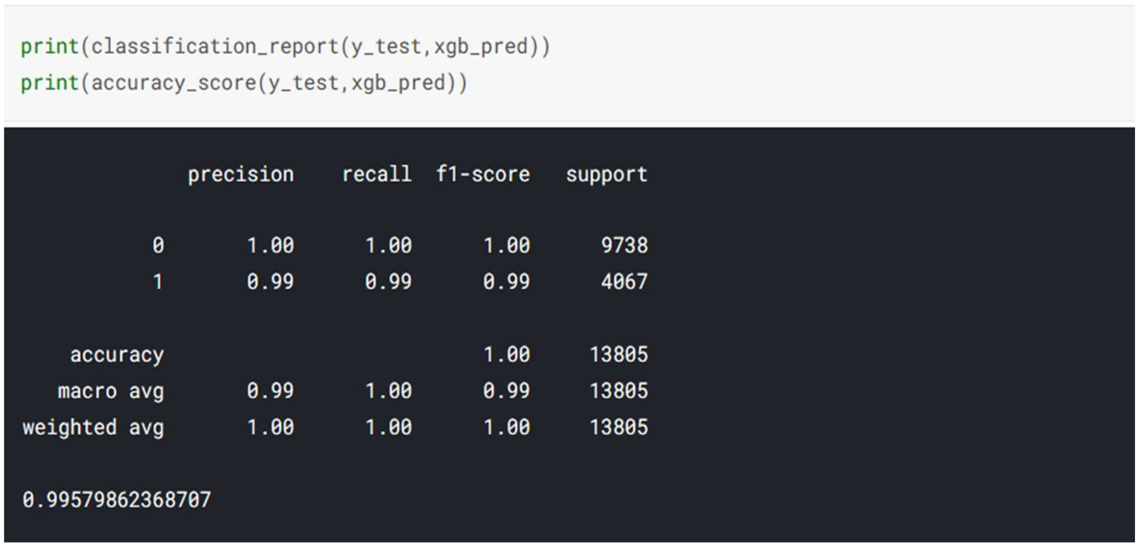
Logistic Regression



```python
print(classification_report(y_test,lsvc_pred))
print(accuracy_score(y_test,lsvc_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      9738
           1       0.97      0.96      0.97      4067

    accuracy                           0.98     13805
   macro avg       0.98      0.97      0.98     13805
weighted avg       0.98      0.98      0.98     13805

0.9797899311843535
```

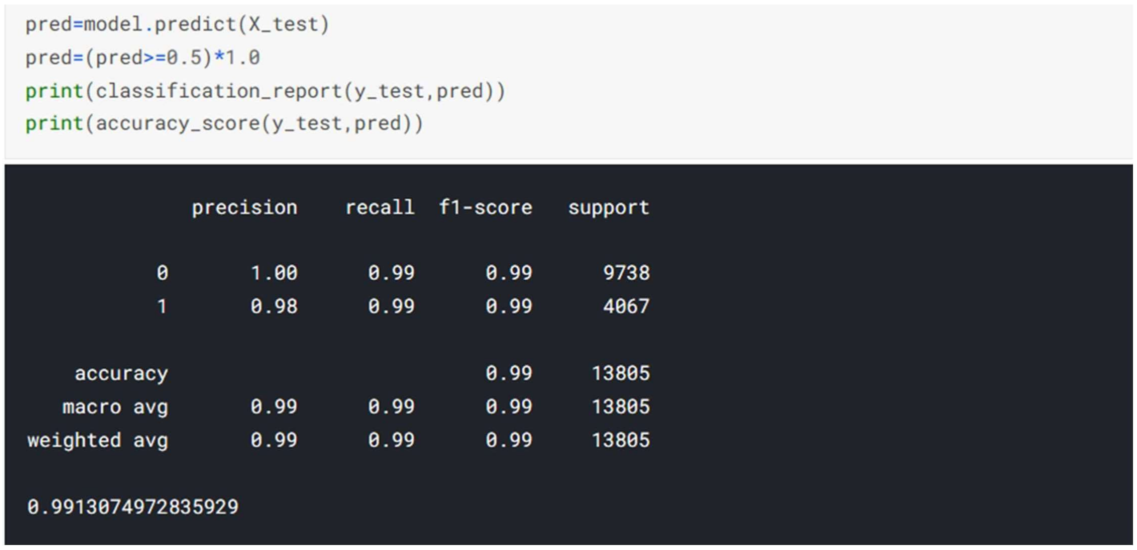
Support Vector Machine



```python
print(classification_report(y_test,xgb_pred))
print(accuracy_score(y_test,xgb_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9738
           1       0.99      0.99      0.99      4067

    accuracy                           1.00     13805
   macro avg       0.99      1.00      0.99     13805
weighted avg       1.00      1.00      1.00     13805

0.99579862368707
```

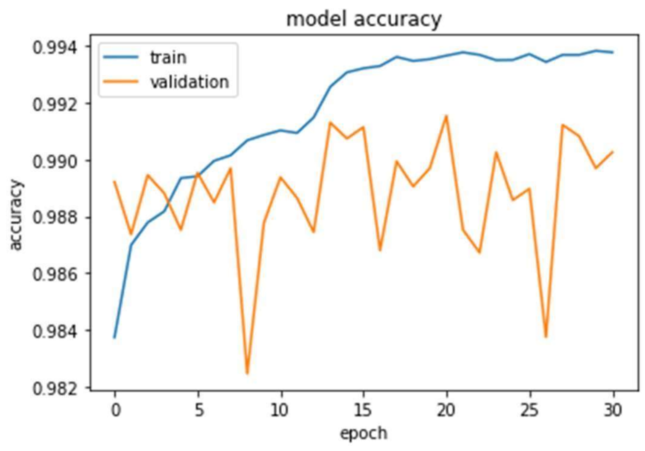
XGBoost

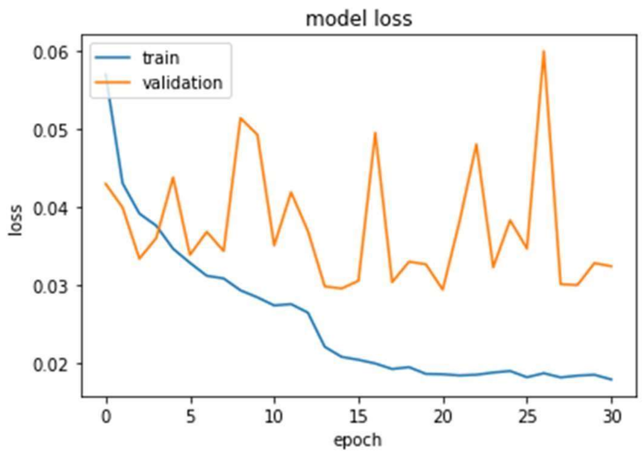


```
pred=model.predict(X_test)
pred=(pred>=0.5)*1.0
print(classification_report(y_test,pred))
print(accuracy_score(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      9738
           1       0.98      0.99      0.99      4067

    accuracy                           0.99     13805
   macro avg       0.99      0.99      0.99     13805
weighted avg       0.99      0.99      0.99     13805

0.9913074972835929
```

Artificial Neural Network





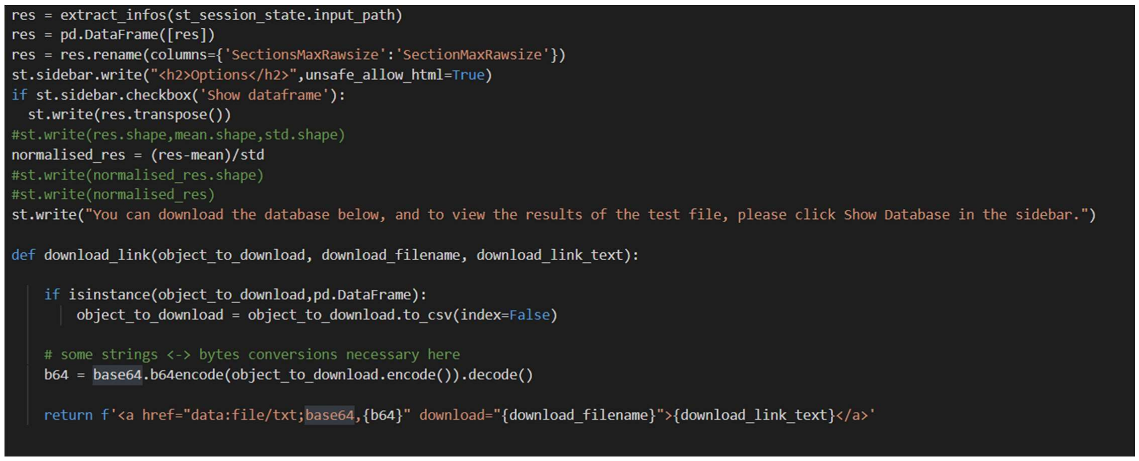
Deployment

Streamlit & NGROK

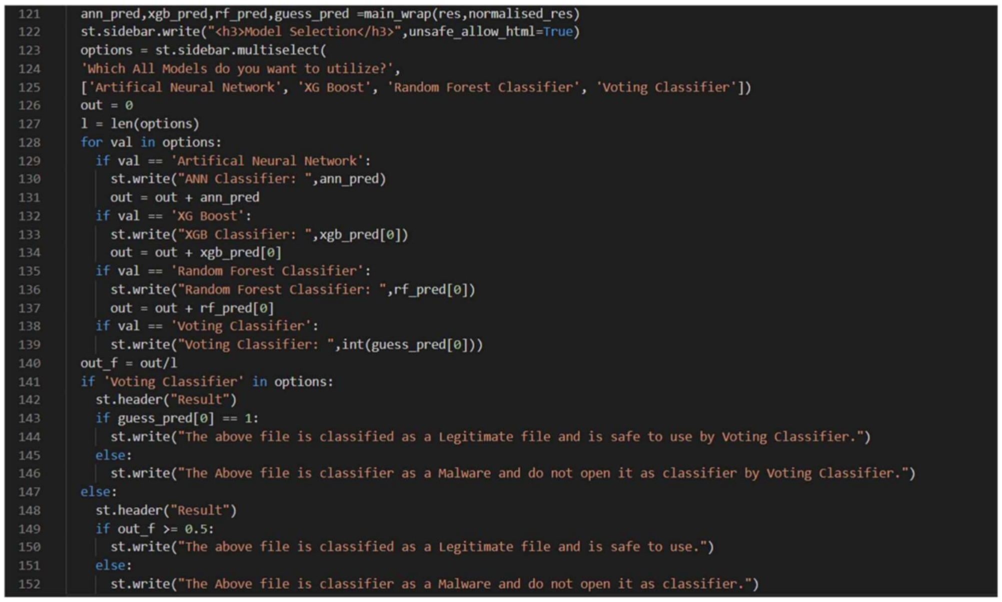
We have created the source code and hence have also used colab to deploy the application, as we could utilize the GPU capabilities of Google Colab. The application was created using streamlit and was wrapped to be hosted online using NGROK.

Streamlit is an open-source Python based tool that can aid any user to create web applications seamlessly and can turn the native python scripts into websites, which are hosted on the local host. Streamlit Provides the availability of multiple components, with the usage of instances of the functions that can be used to create the applications without the in-depth knowledge of web development frameworks.

We have used streamlit to create and locally host the applications designed, and have used NGROK as a wrapper which can host the application over the web as well. The application would only be in the running state for the duration, when the application is hosted locally using Google Colab and streamlit.

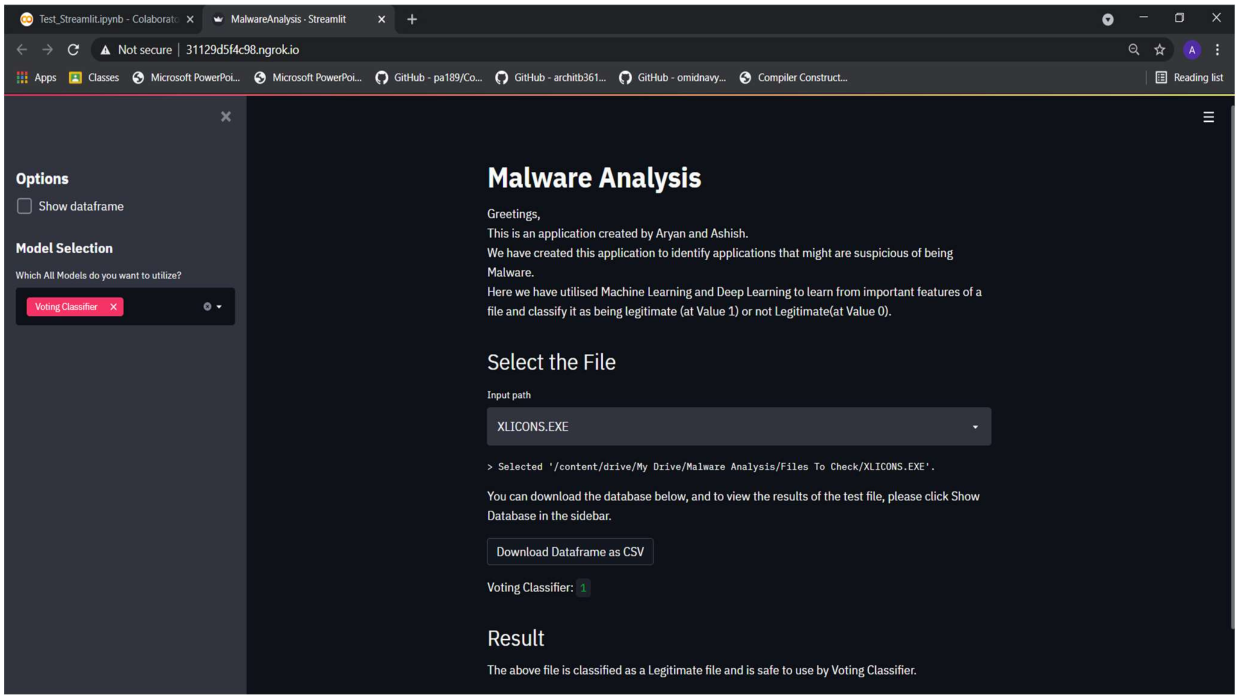


```
res = extract_infos(st_session_state.input_path)
res = pd.DataFrame([res])
res = res.rename(columns={'SectionsMaxRawsize':'SectionMaxRawsize'})
st.sidebar.write("<h2>Options</h2>",unsafe_allow_html=True)
if st.sidebar.checkbox('Show dataframe'):
  st.write(res.transpose())
#st.write(res.shape,mean.shape,std.shape)
normalised_res = (res-mean)/std
#st.write(normalised_res.shape)
#st.write(normalised_res)
st.write("You can download the database below, and to view the results of the test file, please click Show Database in the sidebar.")

def download_link(object_to_download, download_filename, download_link_text):

    if isinstance(object_to_download,pd.DataFrame):
        object_to_download = object_to_download.to_csv(index=False)

    # some strings <-> bytes conversions necessary here
    b64 = base64.b64encode(object_to_download.encode()).decode()

    return f'<a href="data:file/txt;base64,{b64}" download="{download_filename}">{download_link_text}</a>'
```



```
ann_pred,xgb_pred,rf_pred,guess_pred =main_wrap(res,normalised_res)
st.sidebar.write("<h3>Model Selection</h3>",unsafe_allow_html=True)
options = st.sidebar.multiselect(
'Which All Models do you want to utilize?',
['Artifical Neural Network', 'XG Boost', 'Random Forest Classifier', 'Voting Classifier'])
out = 0
l = len(options)
for val in options:
  if val == 'Artifical Neural Network':
    st.write("ANN Classifier: ",ann_pred)
    out = out + ann_pred
  if val == 'XG Boost':
    st.write("XGB Classifier: ",xgb_pred[0])
    out = out + xgb_pred[0]
  if val == 'Random Forest Classifier':
    st.write("Random Forest Classifier: ",rf_pred[0])
    out = out + rf_pred[0]
  if val == 'Voting Classifier':
    st.write("Voting Classifier: ",int(guess_pred[0]))
out_f = out/l
if 'Voting Classifier' in options:
  st.header("Result")
  if guess_pred[0] == 1:
    st.write("The above file is classified as a Legitimate file and is safe to use by Voting Classifier.")
  else:
    st.write("The Above file is classifier as a Malware and do not open it as classifier by Voting Classifier.")
else:
  st.header("Result")
  if out_f >= 0.5:
    st.write("The above file is classified as a Legitimate file and is safe to use.")
  else:
    st.write("The Above file is classifier as a Malware and do not open it as classifier.")
```

Final Application



Options

Show dataframe

Model Selection

Which All Models do you want to utilize?

Voting Classifier

Malware Analysis

Greetings,
This is an application created by Aryan and Ashish.
We have created this application to identify applications that might are suspicious of being Malware.
Here we have utilised Machine Learning and Deep Learning to learn from important features of a file and classify it as being legitimate (at Value 1) or not Legitimate(at Value 0).

Select the File

Input path

XLICONS.EXE

```
> Selected '/content/drive/My Drive/Malware Analysis/Files To Check/XLICONS.EXE'.
```

You can download the database below, and to view the results of the test file, please click Show Database in the sidebar.

Download Dataframe as CSV

Voting Classifier: 1

Result

The above file is classified as a Legitimate file and is safe to use by Voting Classifier.

Conclusion

We have successfully created and deployed an aggregate machine learning classifier which utilizes the features extracted using PE File, the elements present in the PE header of a file which helped us to classify the file to be Malware or Legitimate. We achieved the highest accuracy of 99.57% from Random forests, and XGBoost both followed by Artificial Neural Networks with an accuracy of 99.13%. We successfully deployed the application using streamlit and hosted the application on the internet using NGROK.

Dataset: Virus Share (134th archive)
Dependencies: Keras, Pandas, OS, Numpy, Sklearn, XGboost, Tensorflow, Matplotlib, Seaborn, Streamlit and NGROK.
Hardware: Kaggle CPU and GPU to train the model

References

1. T. Chen and C. Guestrin, “XGBoost,” Aug. 2016, doi: 10.1145/2939672.2939785.
    a. https://stackoverflow.com/questions/38640109/logistic-regression-python-solvers- defintions
    b. https://www.streamlit.io/

2. Firdausi, C. lim, A. Erwin and A. S. Nugroho, "Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection," 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies, Jakarta, 2010, pp. 201-203, doi: 10.1109/ACT.2010.33.

3. Walker and S. Sengupta, "Insights into Malware Detection via Behavioral Frequency Analysis Using Machine Learning," MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM), Norfolk, VA, USA, 2019, pp. 1-6, doi: 10.1109/MILCOM47813.2019.9021034.

4. Sharma S., Rama Krishna C., Sahay S.K. (2019) Detection of Advanced Malware by Machine Learning Techniques. In: Ray K., Sharma T., Rawat S., Saini R., Bandyopadhyay A. (eds) Soft Computing: Theories and Applications. Advances in Intelligent Systems and Computing, vol 742. Springer, Singapore. https://doi.org/10.1007/978-981-13-0589-4_31

5. www.kaggle.com

6. www.ieeexplore.ieee.org