Machine Learning Project: Malware Detection using Machine Learning

Malware is a worldwide pandemic. It is designed to damage computer systems without the knowledge of the owner using the system. Software's from reputable vendors also contain malicious code that affects the system or leaks information's to remote servers. Malware includes computer viruses, spyware, dishonest ad-ware, rootkits, Trojans, etc.
Malicious software disrupts IT and computer processes and in extreme cases can delete, steal or complete breakdown of the corporate network or the loss of business-critical data.

Following can be the impacts of malware on business:

- Attack sites and disable services
- Identity Theft/Identity Spoofing
- Affect network performance or complete breakdown of the corporate network
- Steal sensitive information
- Control over the applications running in your systems
- Hardware Failure

According to security firm Kaspersky Lab reports, cybercriminals do not steal just data. They also stole up to $1 billion from 100 different financial institutions across the U.S., Germany, Russia, Ukraine, and China over the past two years.
According to Coveware Q2 Ransomware Marketplace Report "Ransomware Attacks Costs Nearly Triple in 2019 to over $36K per Attack".
According to the FBI's Internet Crime Complaint Center, ransomware--malicious programs that infect a computer or network and hold data hostage until a ransom is paid--has cost companies $18 million in the past 15 months.

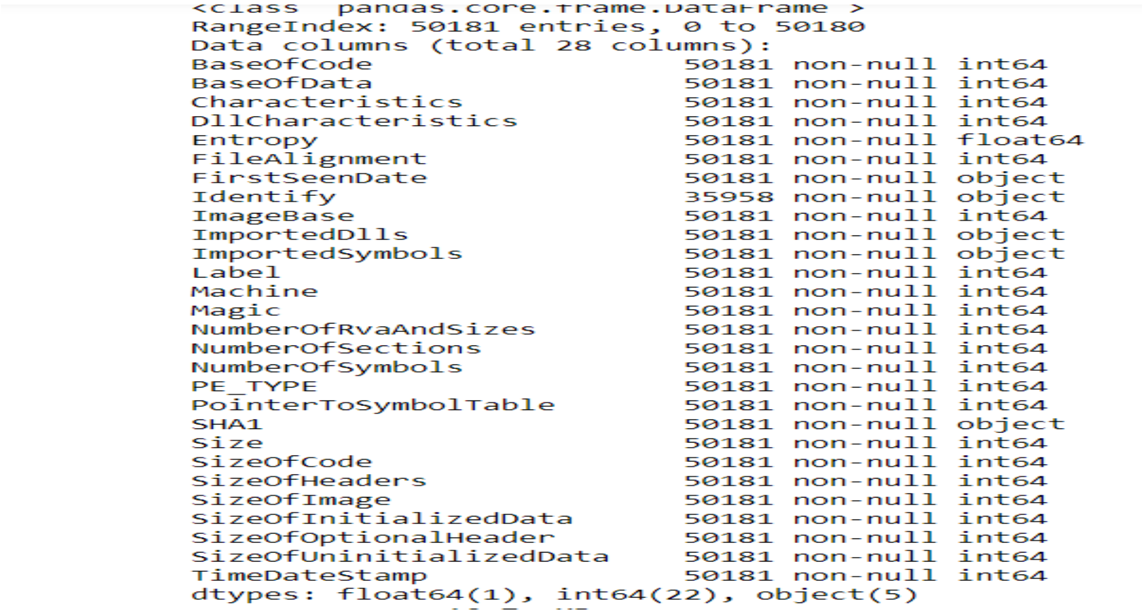
Problem Statement:

In the current world scenario, the internet has become a huge part of everyone's lives, with millions and millions of data packets being downloaded and uploaded each day. With such a huge transfer rate, it becomes very easy for some to sneak in just one unit of data, capable of massive destruction of an application, a whole system, or the entire network. One may never find out if the file one is about to download contains malware or not. With one download containing so many files, it is nearly impossible to physically find out if a file contains a malware or not. Thankfully, this job has been made easier by Machine Learning algorithms.

Dataset:

The dataset which can be used for the model is the Brazillian Malware Dataset.
link: https://github.com/fabriciojoc/brazilian-malware-dataset

The dataset is a mix of numerical and textual data, having around 50000 entries. A summary of the data is given below:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50181 entries, 0 to 50180
Data columns (total 28 columns):
BaseOfCode                     50181 non-null int64
BaseOfData                     50181 non-null int64
Characteristics                50181 non-null int64
DllCharacteristics             50181 non-null int64
Entropy                        50181 non-null float64
FileAlignment                  50181 non-null int64
FirstSeenDate                  50181 non-null object
Identify                       35958 non-null object
ImageBase                      50181 non-null int64
ImportedDlls                   50181 non-null object
ImportedSymbols                50181 non-null object
Label                          50181 non-null int64
Machine                        50181 non-null int64
Magic                          50181 non-null int64
NumberOfRvaAndSizes            50181 non-null int64
NumberOfSections               50181 non-null int64
NumberOfSymbols                50181 non-null int64
PE_TYPE                        50181 non-null int64
PointerToSymbolTable           50181 non-null int64
SHA1                           50181 non-null object
Size                           50181 non-null int64
SizeOfCode                     50181 non-null int64
SizeOfHeaders                  50181 non-null int64
SizeOfImage                    50181 non-null int64
SizeOfInitializedData          50181 non-null int64
SizeOfOptionalHeader           50181 non-null int64
SizeOfUninitializedData        50181 non-null int64
TimeDateStamp                  50181 non-null int64
dtypes: float64(1), int64(22), object(5)
```

A first-look analysis:

- Seeing the number of entries in each column, we can say that the data is not clean. We have some columns missing large amounts of data (Identify). All of the NaNs will have to dealt with.
- As seen the last line of the image, we have 1 float value column, 22 integer value columns, and 5 ‘object’ types, implying they are strings. Work is required on the 5 columns.
- Each column represents a feature of the data file, the meaning of each feature not all quite clear.
- Dive in a little deeper, and we find that some of the columns have a redundant value, i.e. the entire column has the same data entry for all the rows.

The following is a link to further understand the features: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format

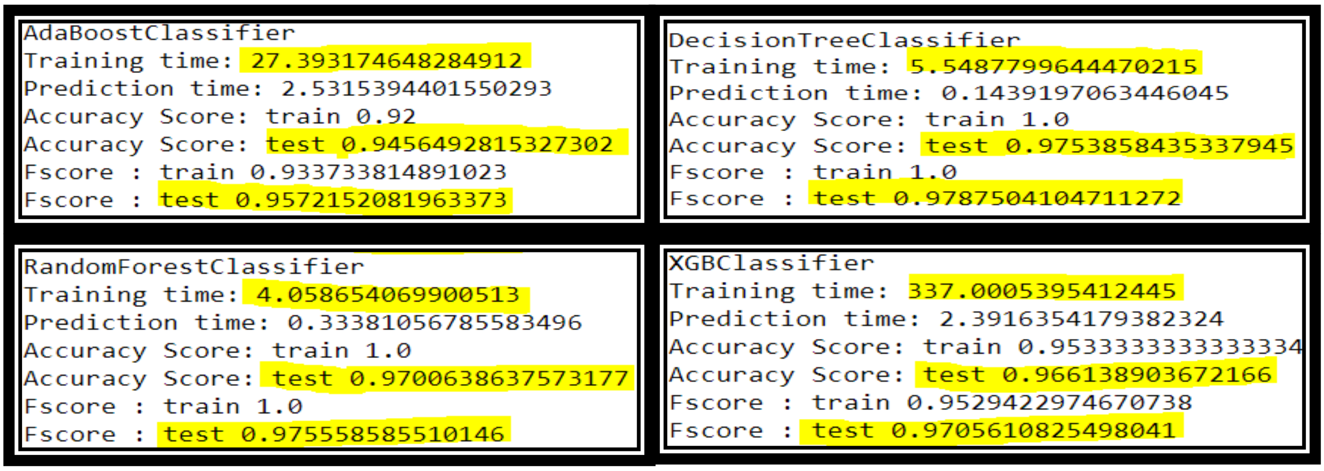
Solution:

Every time the user downloads something, before running/opening/executing it, the user can pass it through a trained ML model, which will predict if the data contains malware or not. If it does, the user can delete the file right away, avoiding any damage to the system. If it does not have any malware, the user can safely open/run the file.

Historical model for comparison(benchmark):

It has been found through a paper (included in the repository) that the Random Forest Classifier has been expected to be the best model, with around 97% accuracy. However, it was not compared to the XGBoost Classifier.

I decided to put this to the test, and had pre-processed the data a bit, and ran 4 models to compare, and the following is the output:



```
AdaBoostClassifier
Training time: 27.393174648284912
Prediction time: 2.5315394401550293
Accuracy Score: train 0.92
Accuracy Score: test 0.9456492815327302
Fscore : train 0.933733814891023
Fscore : test 0.9572152081963373
```

```
DecisionTreeClassifier
Training time: 5.5487799644470215
Prediction time: 0.1439197063446045
Accuracy Score: train 1.0
Accuracy Score: test 0.9753858435337945
Fscore : train 1.0
Fscore : test 0.9787504104711272
```

```
RandomForestClassifier
Training time: 4.058654069900513
Prediction time: 0.33381056785583496
Accuracy Score: train 1.0
Accuracy Score: test 0.9700638637573177
Fscore : train 1.0
Fscore : test 0.975558585510146
```

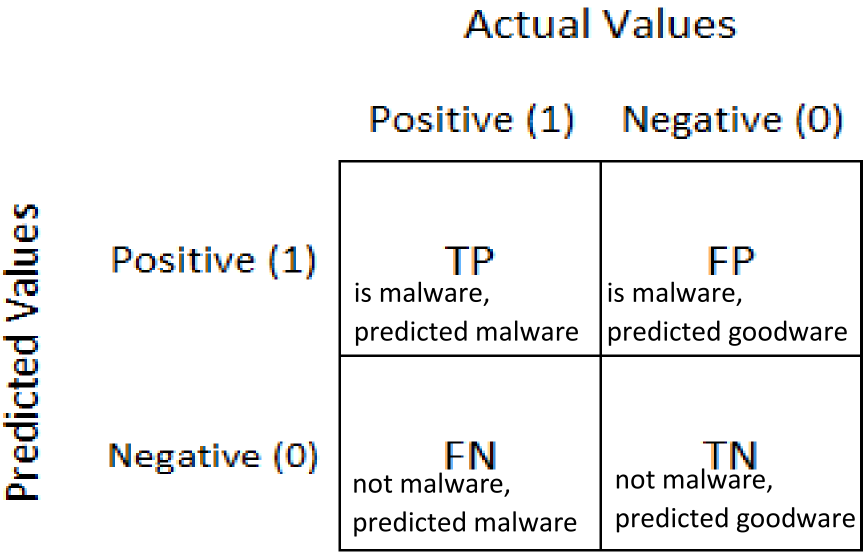
```
XGBClassifier
Training time: 337.0005395412445
Prediction time: 2.3916354179382324
Accuracy Score: train 0.9533333333333334
Accuracy Score: test 0.966138903672166
Fscore : train 0.9529422974670738
Fscore : test 0.9705610825498041
```

On my comparison, I found that though it takes a lot of training time(comparatively), the XGBoost Classifier might prove to be better than the Random Forest Classifier. The accuracy and F-score are quite close for the models, both currently unoptimized.

Hence, the XGB Classifier is definitely worth checking out, after tuning its hyperparameters

Testing Metrics:

Although accuracy may be a good measure, we might want to observe the confusion matrix:



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP is malware, predicted malware	FP is malware, predicted goodware
	Negative (0)	FN not malware, predicted malware	TN not malware, predicted goodware

In our case, accuracy will not be the best metric. Though we can afford having False Negatives (hey, the model's playing it safe), we cannot have any False Positives at all. Hence, a good testing metric would be the Recall value of the model, over its precision. We can check the f-beta score of the model, beta being set to a value close to 1. The testing metric would be the f-beta score.

Project Design:

Like every machine learning project, we start with:

1. Data Pre-processing:
   We already know where the data is. After downloading, an analysis is needed on the dataset, to find out columns having too many Nulls, columns having redundant values, columns not making any difference to the analysis, etc. Also, we need to perform separate pre-processing for the string type columns, (one hot encoding, etc).
   After normalising, we'll have a dataset ready to feed to our models, after being split into training, validation and testing data. The split should be **20% testing data, 10% validation data and 70% training data.**
2. Model Training:
   Since we've already fixated on the model, we are going to implement the XGB Classifier and fit it to the training data. We will evaluate the model using the validation set.
3. Model Tuning:
   Obviously, we cannot expect the model to be the best right at the start. Some hyperparameter tuning will definitely be required (We may use the XGB's hyperparameter tuner to find the best possible estimator). We will train the new optimised model once again, and see how it works.
4. Model Prediction:
   Once the model is ready, it can make predictions on any data provided to it.
5. Model Deployment:
   If working well, the model could be deployed on an AWS API Gateway API, fed with a Lambda function. We can use a localhost webpage to make calls to the API, leading to the model predicting whether the input file has malware or not.

Future Idea:

We could create something like a Google Chrome extension, where the file's download link could be pasted, and a local server downloads the file itself, and checks it through the model. The extension's colour could show the level of danger in the model.