A PROJET REPORT

ON

MALWARE ATTACKS DETECTION SYSTEM

Submitted to

KIIT Deemed to be University

In Partial Fulfillment of the Requirement for the Award

BACHELOR’S DEGREE IN COMPUTER SCIENCE & COMMUNICATION ENGINEERING

By
SAGNIK MUKHERJEE - 1729215
TOULIK DAS - 1729231
RUPAM PATRA - 1729237

UNDER THE GUIDANCE OF
PROF: RAJAT BEHERA

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA – 751024
APRIL 2020

KIIT Deemed to be University

School of Computer Engineering

Bhubaneswar, ODISHA 751024

CERTIFICATE

This is to certify that the project entitled

"MALWARE ATTACKS DETECTION SYSTEM"

Submitted By

SAGNIK MUKHERJEE - 1729215

TOULIK DAS - 1729231

RUPAM PATRA - 1729237

is a record of Bonafide work carried out by them, in the partial fulfillment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering OR Computer Science and Communication Engineering) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2019-2020, under our guidance.

Date: 9/ 04/ 2020

Prof. Rajat Behera
Project Guide

ACKNOWLEDGEMENTS

We are profoundly grateful to Prof. RAJAT BEHERA for his expert guidance and encouragement throughout to see that this project rights its target since its commencement to its completion. The work is a team effort minus which the completion of this project was not possible.

Sagnik Mukherjee
Rupam Patra
Toulik Das

ABSTRACT

Malware detection is an important factor in the security of the computer systems.Anti-malware Software Industries regularly receive large number of suspected malware files to be examined. However, currently utilized signature-based methods cannot provide accurate detection of zero-day attacks and polymorphic viruses. The dark world hackers are using them to lure into systems through the points mentioned in the vulnerability databases. Hence, it is highly necessary to predict the malware at an early stage to avoid further loss. That's why Machine Learning based malware prediction arises.

The objective of the project work predicts a windows system's probability of getting attacked by various families malware on the base level in the time of manufacturing of the System based on different properties of the Operating System and the machine. That helps billions of machines from damage before it happens. Two of the main automation approaches used by large number of researchers are malware classification and clustering, where similar malware samples are grouped into malware families. Grouping malware into families allows malware analysts to examine fewer representative samples from each family, hence streamlining the malware defense process.

In this project, we focus on two aspects of the automated malware defense. For the First part of Our work , determine the best feature extraction, feature representation, and classification methods that result in the best accuracy when used on the top of Google Colab and Jupyter Notebook. Specifically Decision Trees, Gradient Boosting, Adaptive Boosting, Gaussian Naive Bayes and Random Forest classifiers were evaluated.

In the Second Part of the project we Implemented we used Artificial Neural Networks to predict the attacks. We experimented with adding different numbers of system features and hidden neural layers to predict the accuracy of the model. Finally we used the gradient boosting framework to train the Model.We evaluate these approaches on windows defender's data set which contains 80Milion systems data and attacks scenario with 83 different hardware and software features for each machine. The evaluation shows that our representation results in a statistically significant open set recognition performance improvement when compared to a state of the art approach on the data Set. Further goal is to work on optimization algorithms considering the accuracy factor and adding more System features and advanced approaches to protect systems more effectively.

Keywords -- *Malware Prediction, Decision Trees, Naive Bayes, Random Forest, Artificial Neural Network, Gradient boosting framework, K-fold cross validation, Decision Science.*

CONTENTS

INTRODUCTION

With the fast development of the technology, cyber threats also increase because of malware. Malware is defined as “Malware or malicious software is a program that affects a computer system without the user’s permission and with an aim to cause harms to the system or steal private information from the system”. So, in recent years, the malware industry has become a large and well-organized market.While the diversity of malware is increasing, anti-virus scanners cannot fulfill the needs of protection, resulting in millions of hosts being attacked. According to , Juniper Research the cost of data breaches to increase to $2.1 trillion globally by 2019.
One of the major challenges that anti-malware software faces today are the vast amounts of data which needs to be evaluated for potential malicious intent. Thousands of new malwares are emerging every day and the existing malwares are evolving in their structure which is becoming difficult to detect. According to the latest internet threat from Symantec, a whopping 317 million new types of malwares were discovered. A first step in effectively analyzing and classifying such a large number of files is to group them and identify their respective families. Current static and dynamic methods do not provide efficient detection, especially when dealing with zero-day attacks. For this reason, machine learning-based techniques can be used. The goal of this project is to develop a automated model to predict the malware attacks probability on windows based machines by different malicious programs based on hardware specifications of the machine along with software conditions. Using that model Industries can develop more secure systems. That helps billions of machines from damage before it happens.

1.1 Motivation :

Our motive is to classify different malicious programs behaviour according to the systems specification and further predict malware attacks probability on the machine depends on various features of the hardware and software. Much research has been done in the field but accuracy of the model is not that higher to use efficiently. So in this project we try to improve those models along with that we add some updated features and models to predict the malicious attacks.

1.2 RELATED WORKS :

Nowadays malware detection using machine learning methods is considered important for every user and network. Lots of studies have been done in this area, to come with different accuracy for different methods. Various algorithms and classification techniques have been used earlier to enhance this field and a few are illustrated here:

Tian, R et al have proposed extraction method based on PE headers, DLLs and API functions which uses Naïve Bayes, J48 Decision Tree, Support Vector Machine (SVM) methods. The highest accuracy was achieved by J48 Decision Tree algorithm.However, the paper is based on malware detection not properly about malware attack prediction.

In Dragos Gavrilut paper the motive is to make a malware detection system using many modified perceptron machine learning algorithms. For different-different approaches, the accuracy of 69.90% is obtained.

In the research done by Tang, feature extraction is done on the malware data set which contains portable executable files. Malware analysis is done with the data set which gave the accuracy of 97.5 and false-positive rate of 0.03.However it’s based on malware analysis.

Along with that many data science challenges thrown by Industries like kaspaskay,Microsoft among data science communities to solve such problems. But there too accuracy of the build models is not more than approximately 60-68%.

Comparison table for some of the related Projects

SI No.	Citation	Feature Extraction methods, Classifiers and Algorithms Used	Accuracy percentage & Aim of the Research
1	Science Direct,Volume 77 [August 2018]	Early Stage Malware Prediction using Recurrent Neural Network.	Detect malwares based on malicious programs features Not on Machines Specification. Training Data set valid for Windows 7 machine. Accuracy : Trojans with 94% , 89% accuracy for the 6 different variants tested.
2	IEEE[2018], 12th International Conference on Malicious and Unwanted Software(2017)	Predicting signatures of future malware variants	It's also dependent on malware features. Accuracy : Average accuracy for different malwares prediction is approximate 89%.
3	IEEE[2019] , Transactions on Sustainable Computing(2018)	Robust Malware Detection for Internet of (Battlefield) Things Devices Using Deep Eigenspace Learning	deep learning based method to detect Internet Of Battlefield Things (IoBT) malware via the device's Operational Code (Op-code) sequence.
4	Microsoft malware prediction challenge by Georgia Tech & Northeastern University(2019),Vladislav Bogorod.	LightGBM. Baseline Model Using Sparse Matrix.	Detect malware in base level using Machines Specification. Accuracy : 70% (Top Ranks)
5	Microsoft malware prediction challenge by Georgia Tech & Northeastern University(2019), Andrew Lukyanenko.	LGBM & Feature Engineering	Detect malware in base level using Machines Specification. Accuracy: 68-70%(Top Ranks)

6	Microsoft malware prediction challenge by Georgia Tech & Northeastern University(2019), Chris Deotte.	Time Split validation & Adversarial EDA	Detect malware in base level using Machines Specification. Accuracy : 68%(Top Ranks)
7	Microsoft malware prediction challenge by Georgia Tech & Northeastern University(2019), NFFM Baseline.	K-fold with adam optimizer	Detect malware in base level using Machines Specification. Accuracy : 69%(Top Ranks)

2. Objective :

The objective of the project work predicts a windows system's probability of getting attacked by various families malware on the base level in the time of manufacturing of the System based on different properties of the Operating System and the machine. That helps billions of machines from damage before it happens. Two of the main automation approaches used by large number of researchers are malware classification and clustering, where similar malware samples are grouped into malware families. Grouping malware into families allows malware analysts to examine fewer representative samples from each family, hence streamlining the malware defense process.

In this project, we focus on two aspects of the automated malware defense. For the First part of Our work , determine the best feature extraction, feature representation, and classification methods that result in the best accuracy when used on the top of Google Colab and Jupyter Notebook. Specifically Decision Trees, Gradient Boosting, Adaptive Boosting, Gaussian Naive Bayes and Random Forest classifiers were evaluated.

In the Second Part of the project we Implemented we used Artificial Neural Networks to predict the attacks. We experimented with adding different numbers of system features and hidden neural layers to predict the accuracy of the model. Finally we used the gradient boosting framework to train the Model.We evaluate these approaches on windows defender's data set which contains 80Milion systems data and attacks scenario with 83 different hardware and software features for each machine.

3. Definitions and Overview :

Machine learning based malware defense systems that help automate and alleviate the burdens of human analysts need to fulfill certain requirements. First, the feature extraction process needs to retain more useful information about the malware. Second, the feature extraction process should be scalable to cope up with the influx of malware releases. Finally, the classification and clustering systems should be capable of operating in an open set scenario.

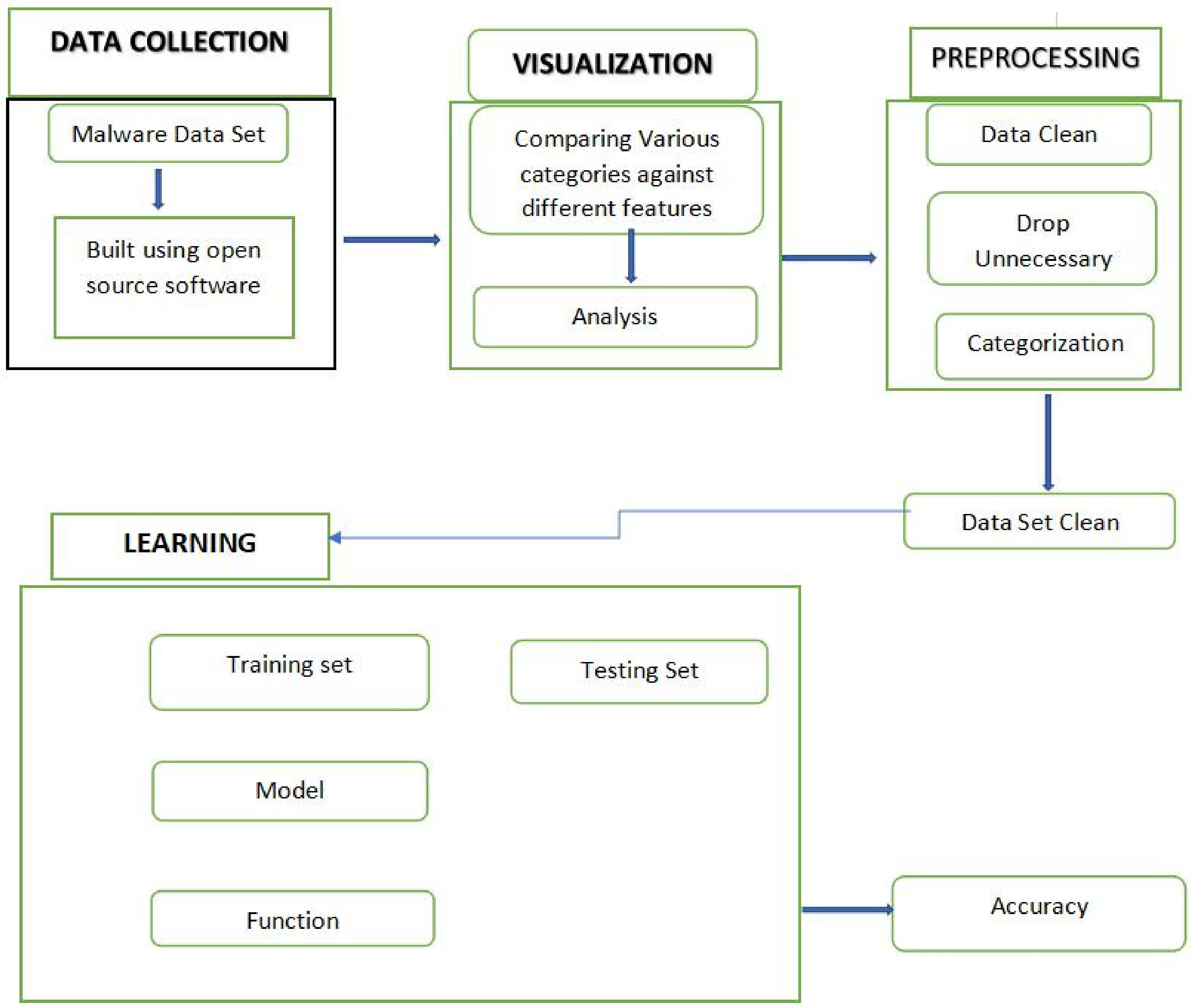


Figure 1 : Proposed Architecture For This Model

3.1 System Dependencies for malware attacks :

Malware attacks are depends on many Software and hardware specification of the target System. Eg. Outdated Systems or using antivirus which is not much updated with new malicious programs can be attacked easily. In Our data Set we consider almost all constraints by which windows security systems face malware attacks for last few years.

The sampling methodology used to create this data set was designed to meet certain business constraints, both in regards to user privacy as well as the time period during which the machine was running. Malware detection is inherently a time-series problem, but it is made complicated by the introduction of new machines, machines that come online and offline, machines that receive patches, machines that receive new operating systems, etc. While the data set provided here has been roughly split by time, the complications and sampling requirements mentioned above may mean you may see imperfect agreement between your cross validation, public, and private scores! Additionally, this data set is not representative of Microsoft customers' machines in the wild; it has been sampled to include a much larger proportion of malware machines.

3.2 Need for machine learning :

As stated before, malware detectors that are based on signatures can perform well on previously-known malware, that was already discovered by some anti-virus vendors. However, it is unable to detect polymorphic malware, that has an ability to change its signatures, as well as new malware, for which signatures have not been created yet. In turn, the accuracy of heuristics-based detectors is not always sufficient for adequate detection, resulting in a lot of false-positives and false-negatives.

Need for the new detection methods is dictated by the high spreading rate of polymorphic viruses. One of the solutions to this problem is reliance on the heuristics-based analysis in combination with machine learning methods that offer a higher efficiency during detection. heuristics-based analysis in combination with machine learning methods that offer a higher efficiency during detection.

3.3 Hardware & Software Interface Used :

Software :

1. Python 3.7
2. Jupyter Note Book
3. Tensorflow 2.0
4. Keras 2.2.5
5. Spyder IDE
6. Python Data Visualization tools
7. Scikit-learn 0.22.2

Hardware:

1. GPU(Nvidia RTX)
2. Intel i5 7th Gen Processor
3. RAM 16GB

4. Problem Statement :

Prediction of a windows system's probability of getting attacked by various families malware on the base level in the time of manufacturing of the System based on different properties of the Operating System and the machine. Based on windows defender's data set which contains 80Milion systems data and attacks scenario with 83 different hardware and software features for each machine.

5. Data Set :

5.1 Over view of the Data Set :

The telemetry data containing different properties and the machine infections was generated by combining heartbeat and threat reports collected by Microsoft's endpoint protection solution, Windows Defender. The data set contains 80lacks rows and 83 columns.

Each row in this data set corresponds to a machine, uniquely identified by a MachineIdentifier. HasDetections is the ground truth and indicates that Malware was detected on the machine. Using the information and labels in train.csv, you must predict the value for HasDetections for each machine in test.csv.

5.2 Important Features of the Data Set :

The data set containing many features among them some important features shown below :

HasDetections - is the ground truth and indicates that Malware was detected on the machine
MachineIdentifier - Individual machine ID
HasTpm - True if machine has tpm
CountryIdentifier - ID for the country the machine is located in
CityIdentifier - ID for the city the machine is located in
GeoNameIdentifier - ID for the geographic region a machine is located in
LocaleEnglishNameIdentifier - English name of Locale ID of the current user
Platform - Calculates platform name (of OS related properties and processor property)
Processor - This is the process architecture of the installed operating system
OsVer - Version of the current operating system
OsBuild - Build of the current operating system
OsSuite - Product suite mask for the current operating system.

OsPlatformSubRelease - Returns the OS Platform sub-release (Windows Vista, Windows 7, Windows 8, TH1, TH2)

OsBuildLab - Build lab that generated the current OS. Example: 9600.17630.amd64fre.winblue_r7.150109-2022

IsProtected - This is a calculated field derived from the Spynet Report's AV Products field. Returns: a. TRUE if there is at least one active and up-to-date antivirus product running on this machine. b. FALSE if there is no active AV product on this machine, or if the AV is active, but is not receiving the latest updates. c. null if there are no Anti Virus Products in the report. Returns: Whether a machine is protected.

AutoSampleOptIn - This is the SubmitSamplesConsent value passed in from the service, available on CAMP 9+

SMode - This field is set to true when the device is known to be in 'S Mode', as in, Windows 10 S mode, where only Microsoft Store apps can be installed
Firewall - This attribute is true (1) for Windows 8.1 and above if windows firewall is enabled, as reported by the service.

Census_MDC2FormFactor - A grouping based on a combination of Device Census level hardware characteristics. The logic used to define Form Factor is rooted in business and industry standards and aligns with how people think about their device. (Examples: Smartphone, Small Tablet, All in One, Convertible...)

Census_DeviceFamily - AKA DeviceClass. Indicates the type of device that an edition of the OS is intended for. Example values: Windows.Desktop, Windows.Mobile, and iOS.Phone

SkuEdition - The goal of this feature is to use the Product Type defined in the MSDN to map to a 'SKU-Edition' name that is useful in population reporting.

Census_PrimaryDiskTotalCapacity - Amount of disk space on primary disk of the machine in MB

Census_PrimaryDiskTypeName - Friendly name of Primary Disk Type - HDD or SSD

Census_SystemVolumeTotalCapacity - The size of the partition that the System volume is installed on in MB

Census_HasOpticalDiskDrive - True indicates that the machine has an optical disk drive (CD/DVD)

Census_TotalPhysicalRAM - Retrieves the physical RAM in MB

Census_IsVirtualDevice - Identifies a Virtual Machine (machine learning model)

Census_IsTouchEnabled - Is this a touch device ?

Census_IsPenCapable - Is the device capable of pen input ?

Census_OSWUAutoUpdateOptionsName - Friendly name of the WindowsUpdate auto-update settings on the machine.

Census_IsPortableOperatingSystem - Indicates whether OS is booted up and running via Windows-To-Go on a USB stick.

Census_GenuineStateName - Friendly name of OSGenuineStateID. 0 = Genuine

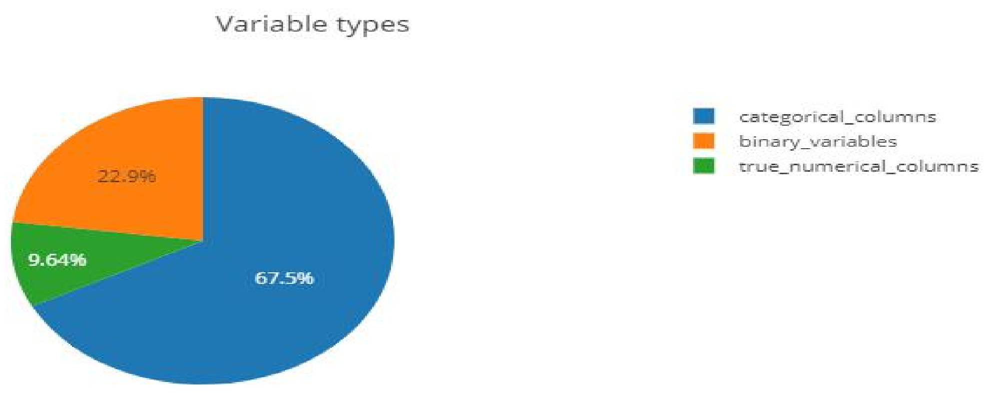
Census_ActivationChannel - Retail license key or Volume license key for a machine.

5.3 Graphical Representation :

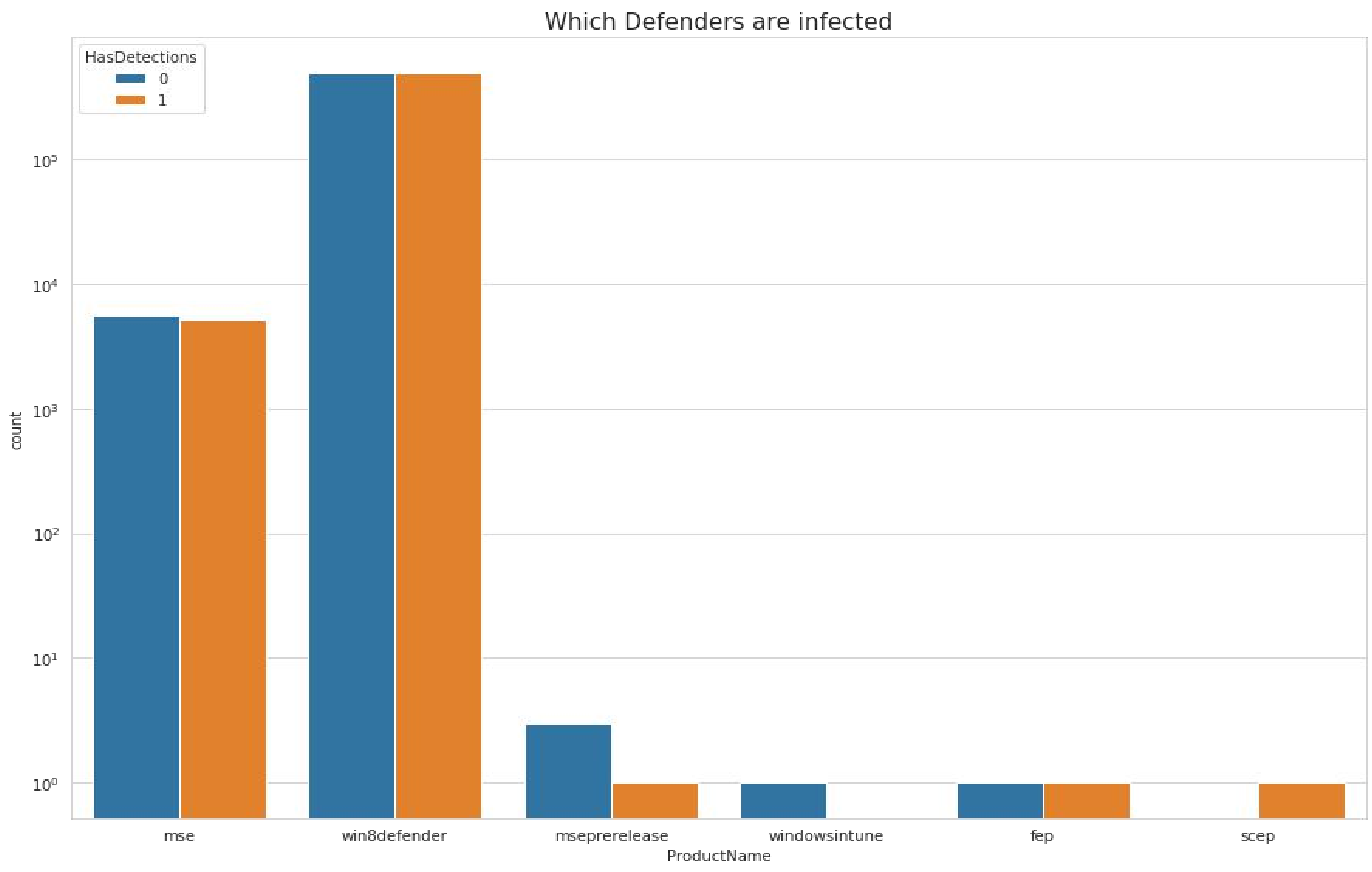
In the visualization part, many key points have been analyzed which helped in finding some relations between various features.This visualization work is done using a python library called seaborn which helps in plotting clean and understandable graphs. For the visualization, each feature is removed to understand the importance that it has in the original data set. Some Of the Important data for the Model building is represented below :

Variable Types :

Different Types of variable in the data set shown here.

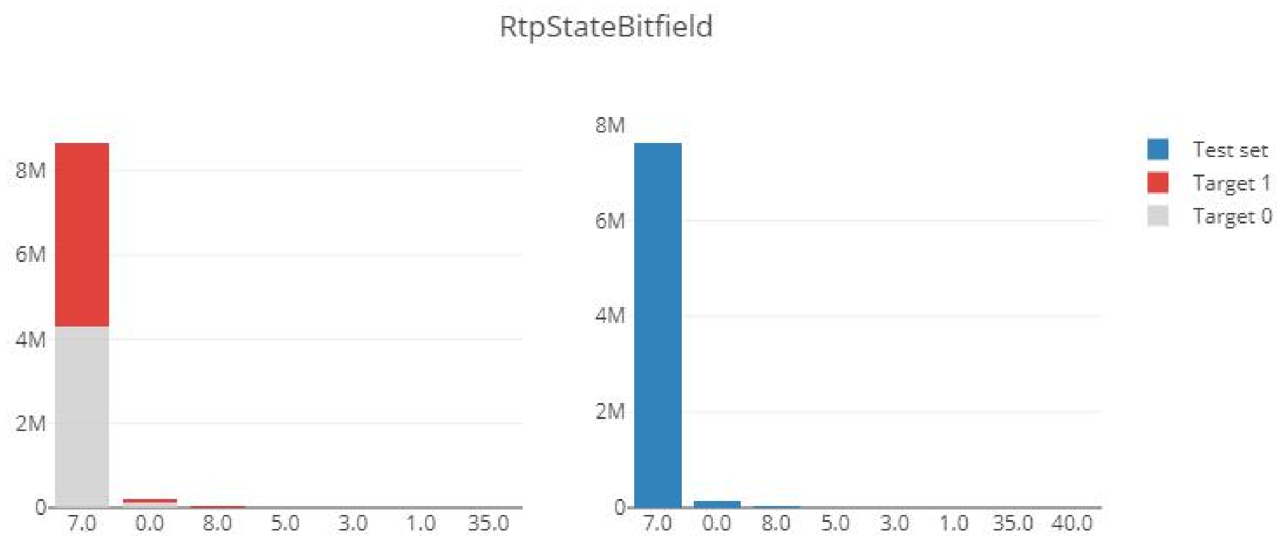


Infected Defenders :



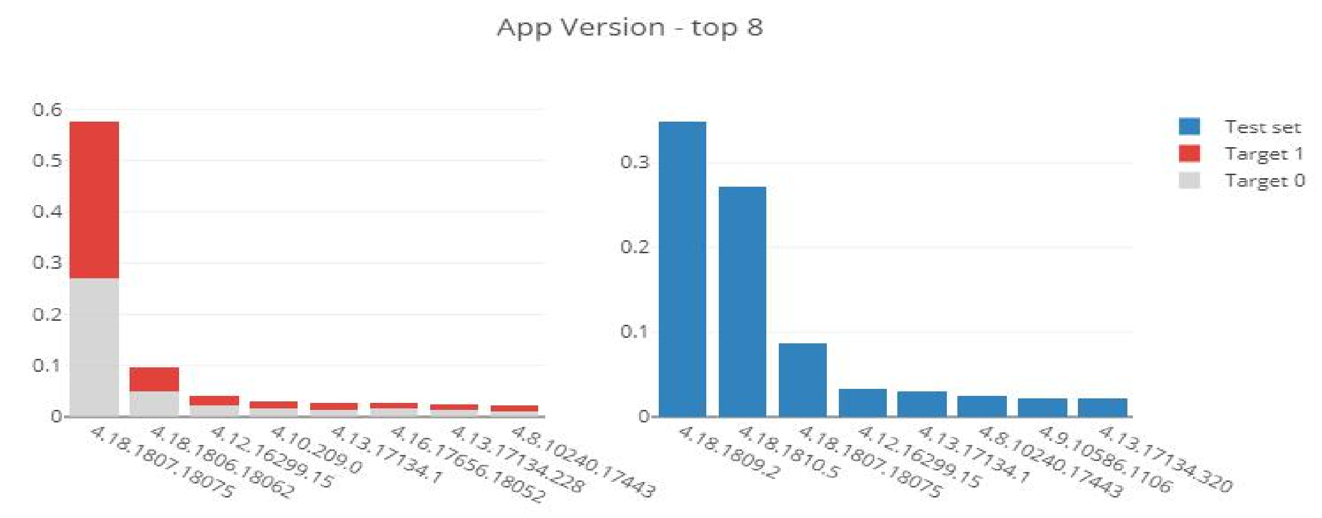
<u>Rtp State Bitfield</u>

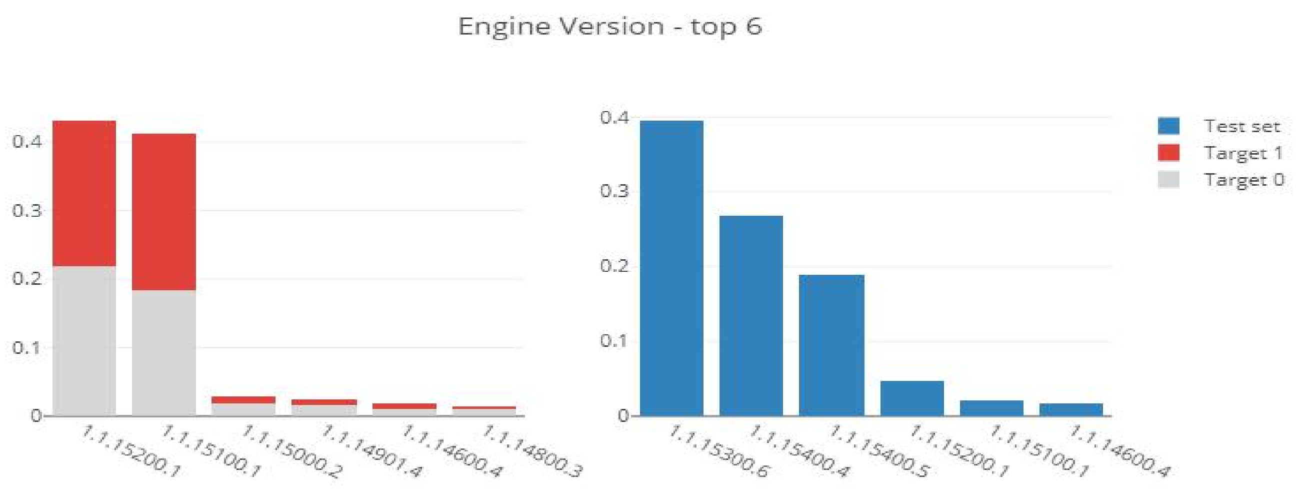
We don't have any infos on this one. Rtp means Real-time transport protocol, it is a network protocol for delivering audio and video over IP networks.



Engine and App version

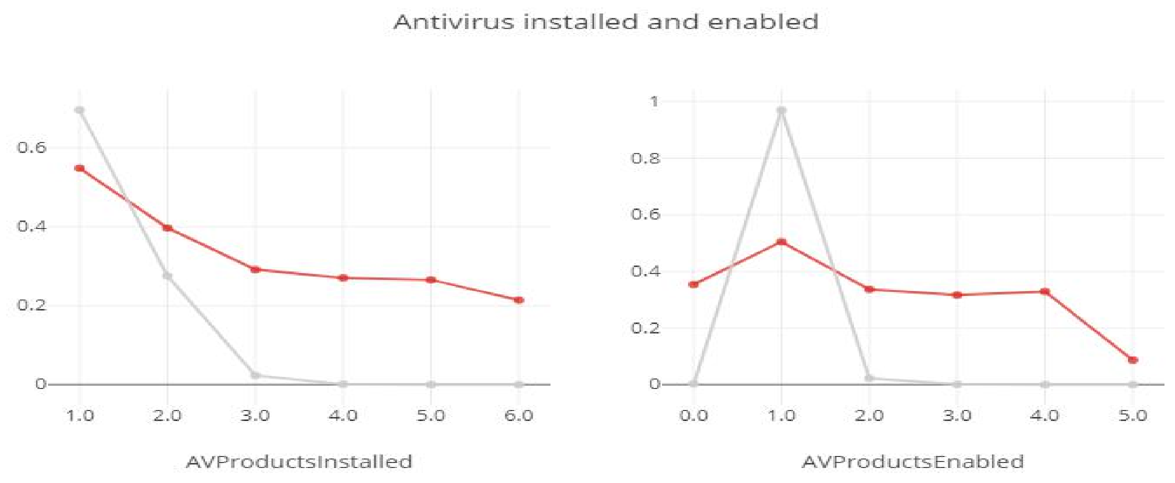
In the next plots the left barplot is the train data set and the right is the test data set. The left plot (train data) is divided in positive and negative target. We can see that the most common versions are slightly different for train and test data sets.





Antivirus installed and Enabled

These two numerical columns doesn't have a description yet, but I think it's the number of installed and enabled AV software's.

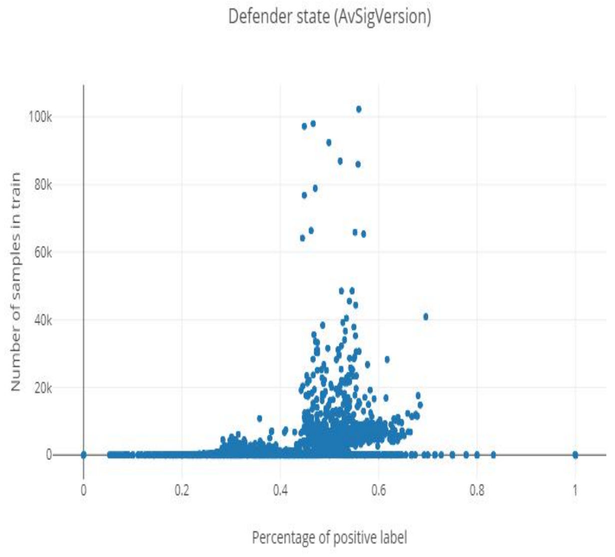


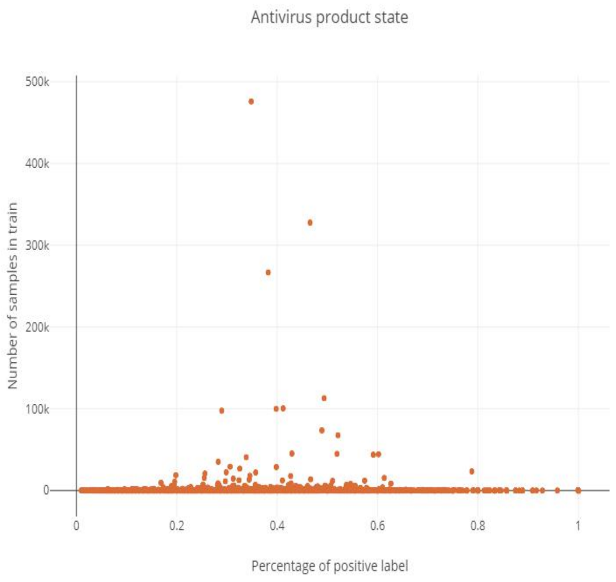
AvSigVersion

This is the most important feature in the current kernels using GBM and is described as the current defender state information.

AV Product State

ID for the specific configuration of a user's antivirus software. The configuration number 53447 has almost 6M machines with 55% detection rate, while the others 28.969 configurations have less than half million machines and very different detection ratios. In the next plot I'll remove configuration 53447 for better visualization:



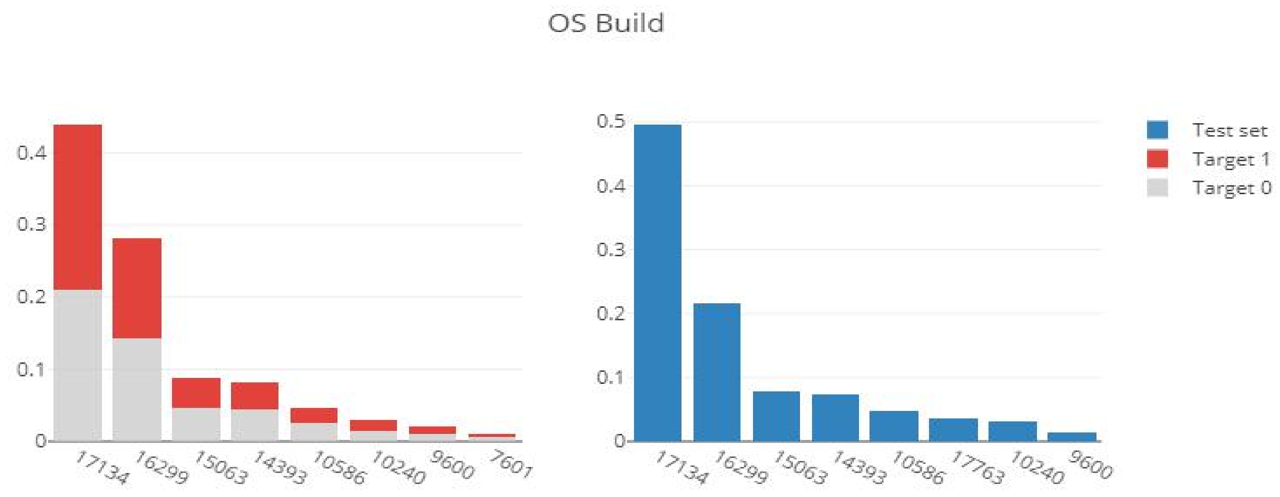


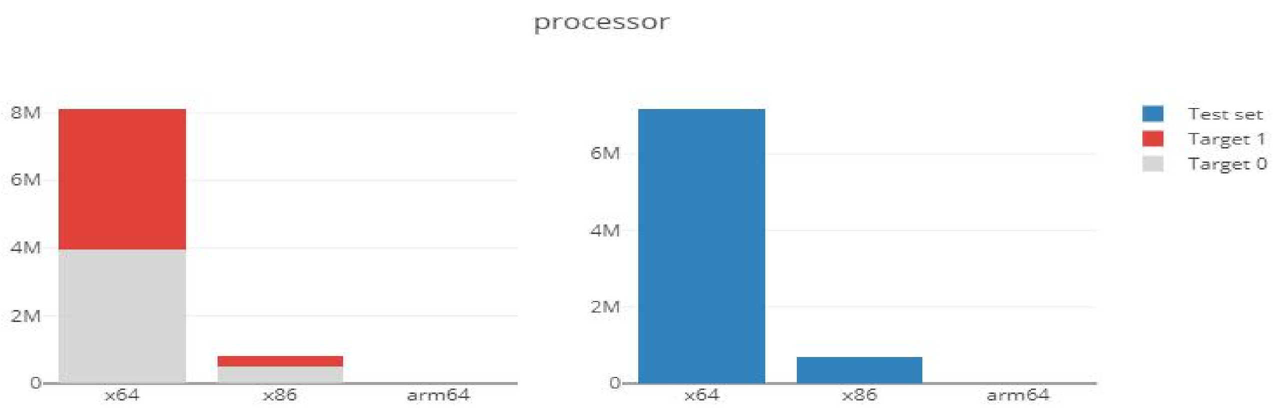
Processor

It seems that x86 architecture is less susceptible to malwares or maybe the users are more careful:

OS Build

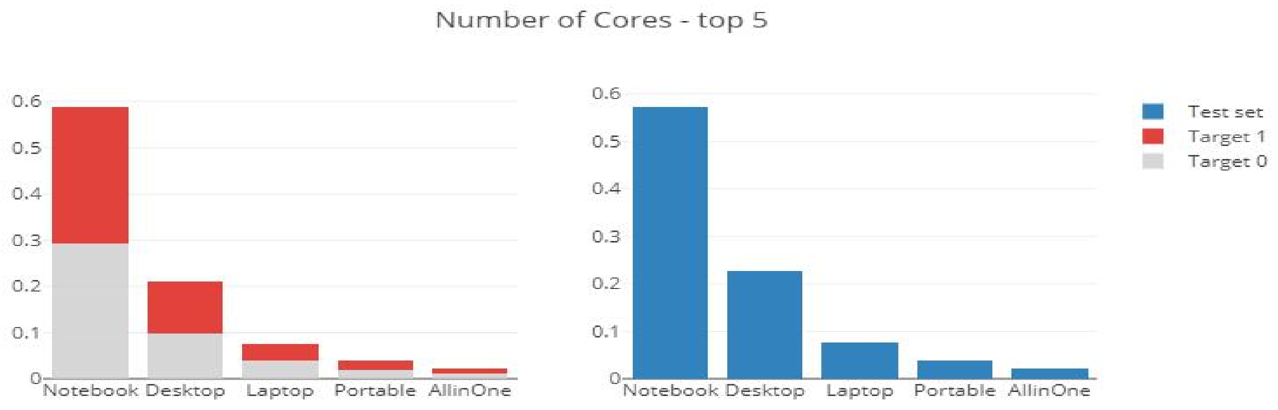
Build of the current operating system.





CPU Cores and Memory

These two features can also be treated as categoricals. Most machines (around 95%) are using 2, 4 or 8 cores:

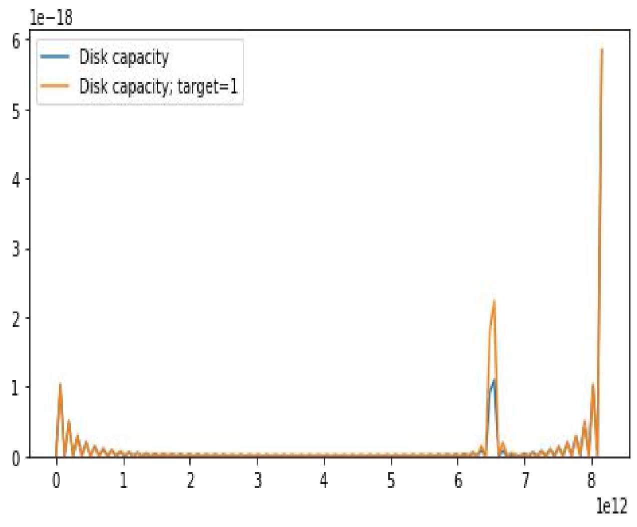


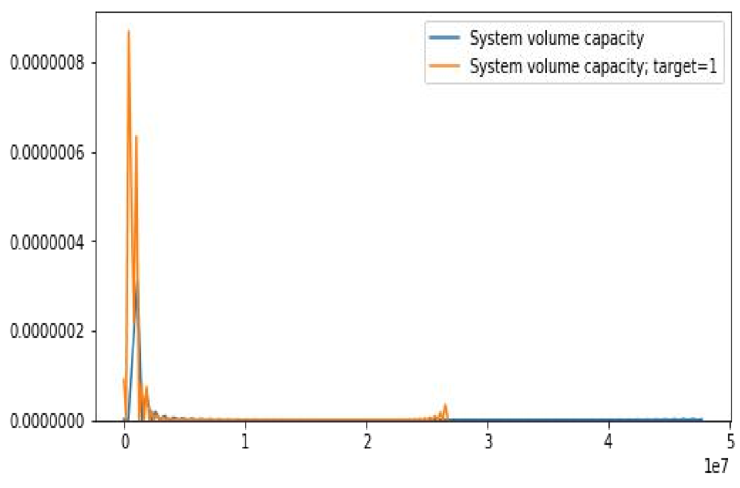
Disk capacity

Primary disk capacity in MB.

System volume capacity

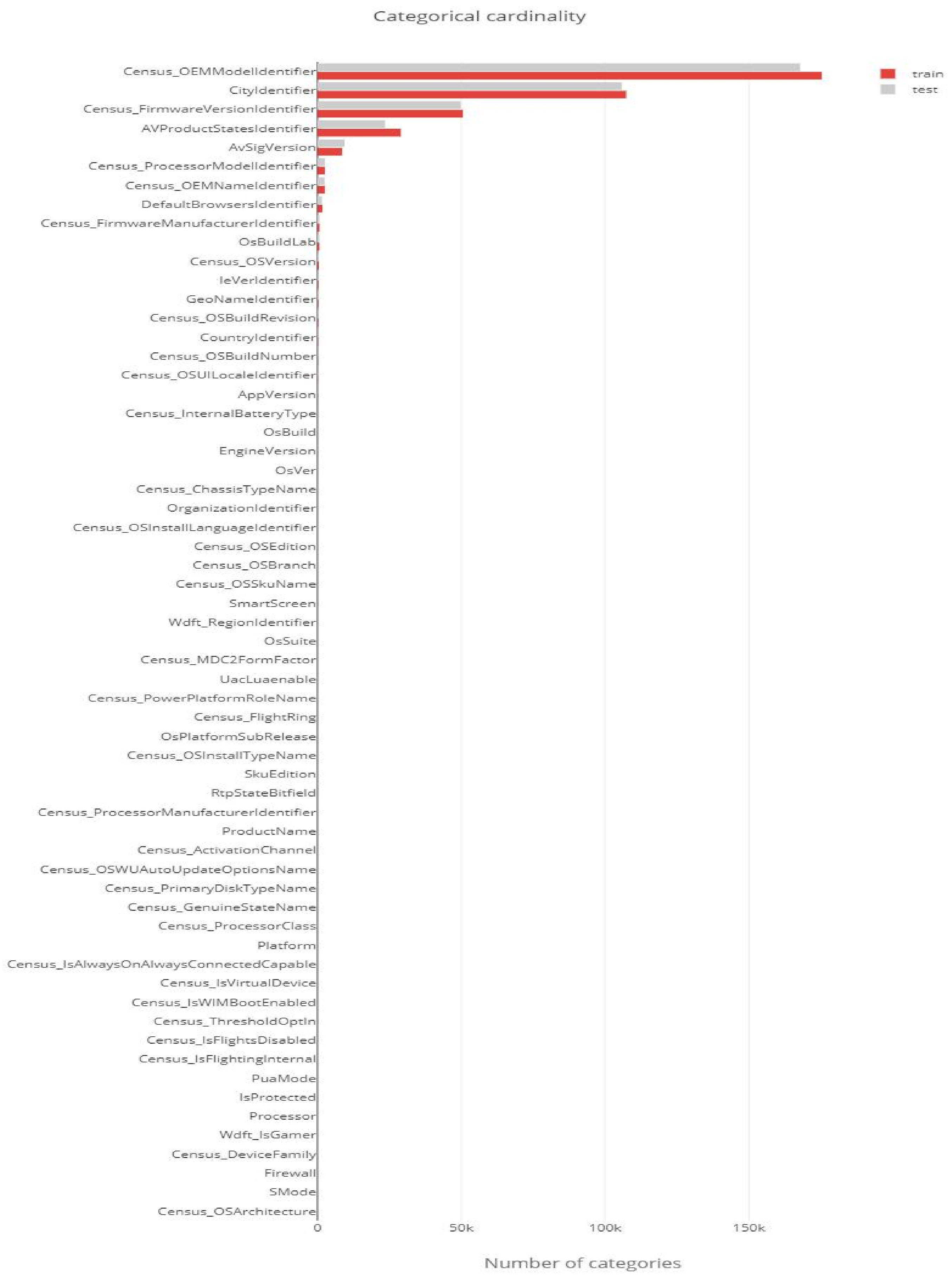
Capacity (in MB) for the volume where the system is installed.





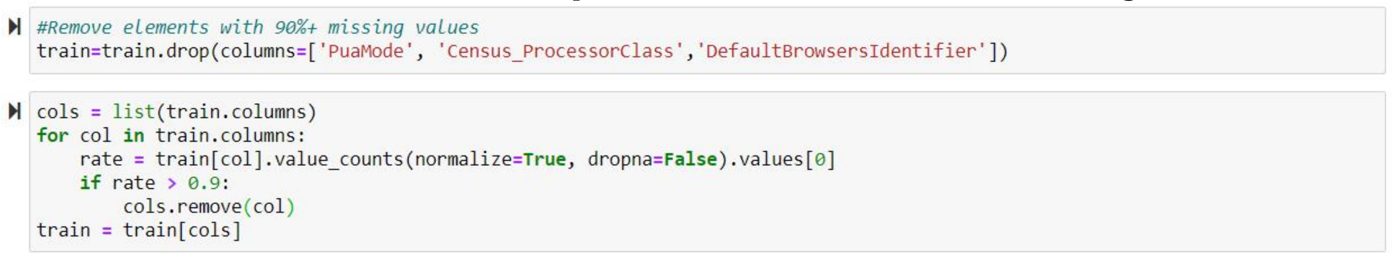
Categorical Cardinality :

Most features are categorical in this data set, so in this section we'll have a look at the number of categories that features has in the train and
tested.



5.4 Data Cleaning(Pre Processing) :

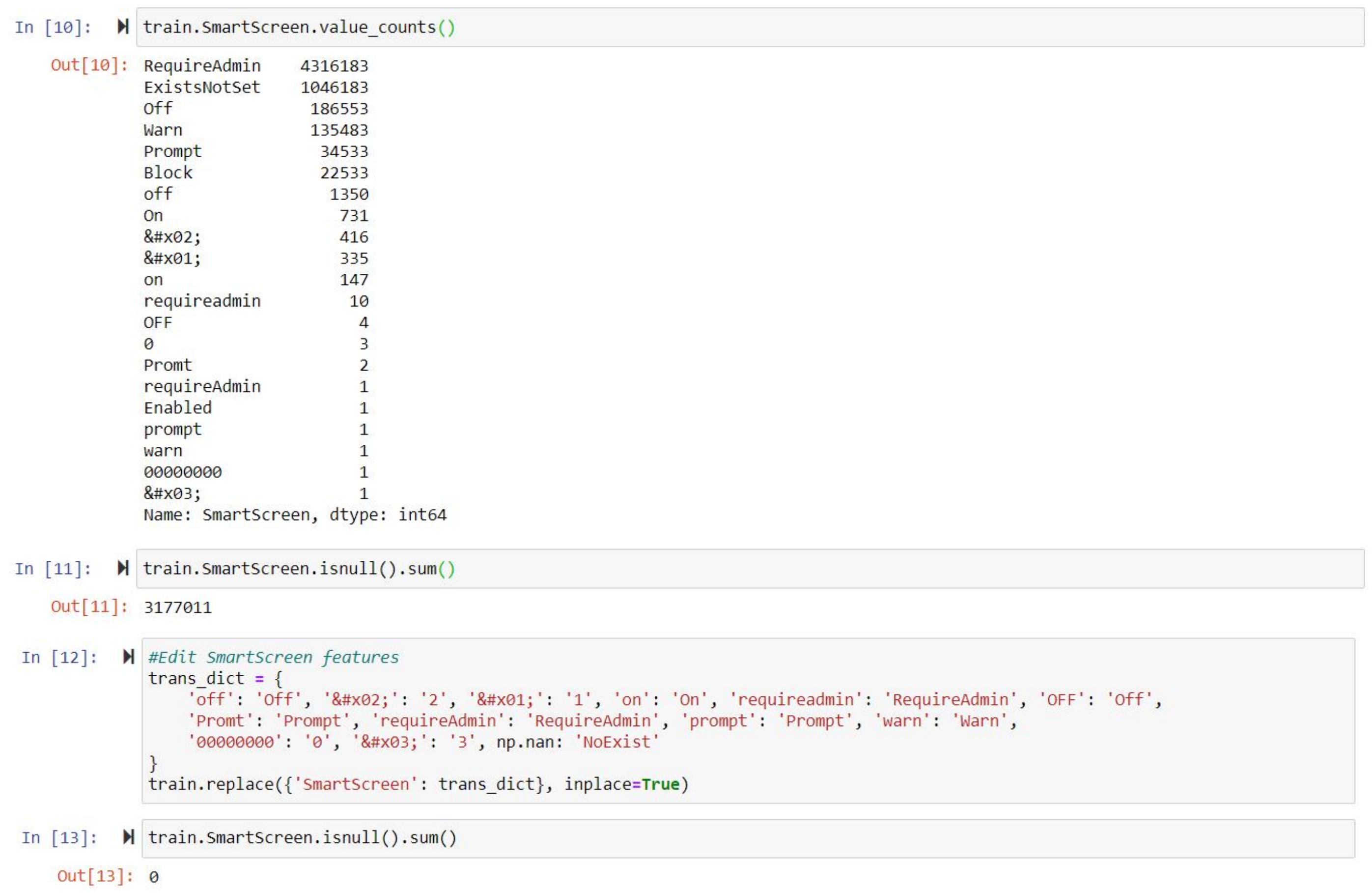
Some of the features in data Set have 99% Missing values. We get that by data Visualization.We removed those features. Then we also dorp those features which have 90% missing values.



```
#Remove elements with 90%+ missing values
train=train.drop(columns=['PuaMode', 'Census_ProcessorClass','DefaultBrowsersIdentifier'])
```

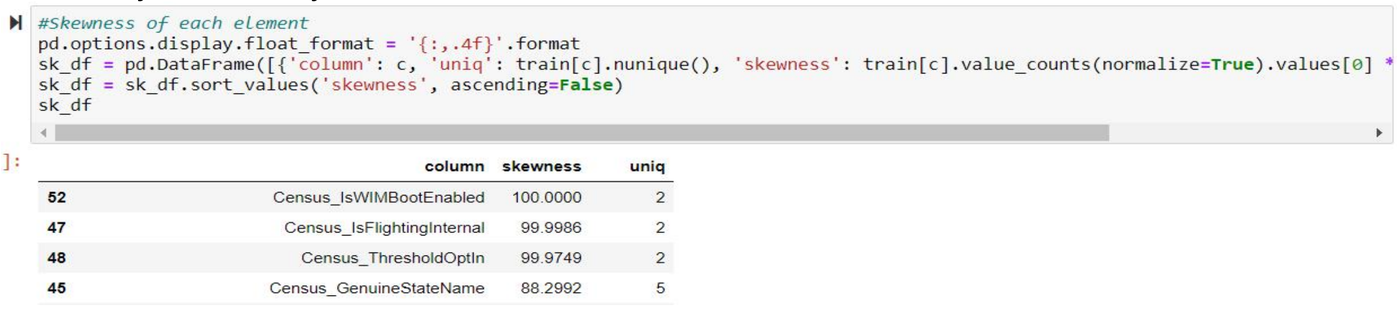
```
cols = list(train.columns)
for col in train.columns:
    rate = train[col].value_counts(normalize=True, dropna=False).values[0]
    if rate > 0.9:
        cols.remove(col)
train = train[cols]
```

Smart Screen is one of the most feature in this prediction. We get Null values in that along with some bugs mix with the different Smart screen type names. We replace Null values and fix the name errors.



```
In [10]:  train.SmartScreen.value_counts()

Out[10]: RequireAdmin    4316183
         ExistsNotSet    1046183
         Off              186553
         Warn             135483
         Prompt            34533
         Block             22533
         off                1350
         On                  731
         &#x02;              416
         &#x01;              335
         on                  147
         requireadmin         10
         OFF                   4
         0                     3
         Promt                 2
         requireAdmin          1
         Enabled               1
         prompt                1
         warn                  1
         00000000              1
         &#x03;                1
         Name: SmartScreen, dtype: int64

In [11]:  train.SmartScreen.isnull().sum()

Out[11]: 3177011

In [12]:  #Edit SmartScreen features
          trans_dict = {
              'off': 'Off', '&#x02;': '2', '&#x01;': '1', 'on': 'On', 'requireadmin': 'RequireAdmin', 'OFF': 'Off',
              'Promt': 'Prompt', 'requireAdmin': 'RequireAdmin', 'prompt': 'Prompt', 'warn': 'Warn',
              '00000000': '0', '&#x03;': '3', np.nan: 'NoExist'
          }
          train.replace({'SmartScreen': trans_dict}, inplace=True)

In [13]:  train.SmartScreen.isnull().sum()

Out[13]: 0
```

In same way we cleaned some of the other important features too. Next we checked skewness of the features and removed those which have skewness greater than 99% and convert all categorical features by One Hot key encoder



```
#Skewness of each element
pd.options.display.float_format = '{:,.4f}'.format
sk_df = pd.DataFrame([{'column': c, 'uniq': train[c].nunique(), 'skewness': train[c].value_counts(normalize=True).values[0] *
sk_df = sk_df.sort_values('skewness', ascending=False)
sk_df
```

]:

	column	skewness	uniq
52	Census_IsWIMBootEnabled	100.0000	2
47	Census_IsFlightingInternal	99.9986	2
48	Census_ThresholdOptIn	99.9749	2
45	Census_GenuineStateName	88.2992	5

6. **Project Planning** :

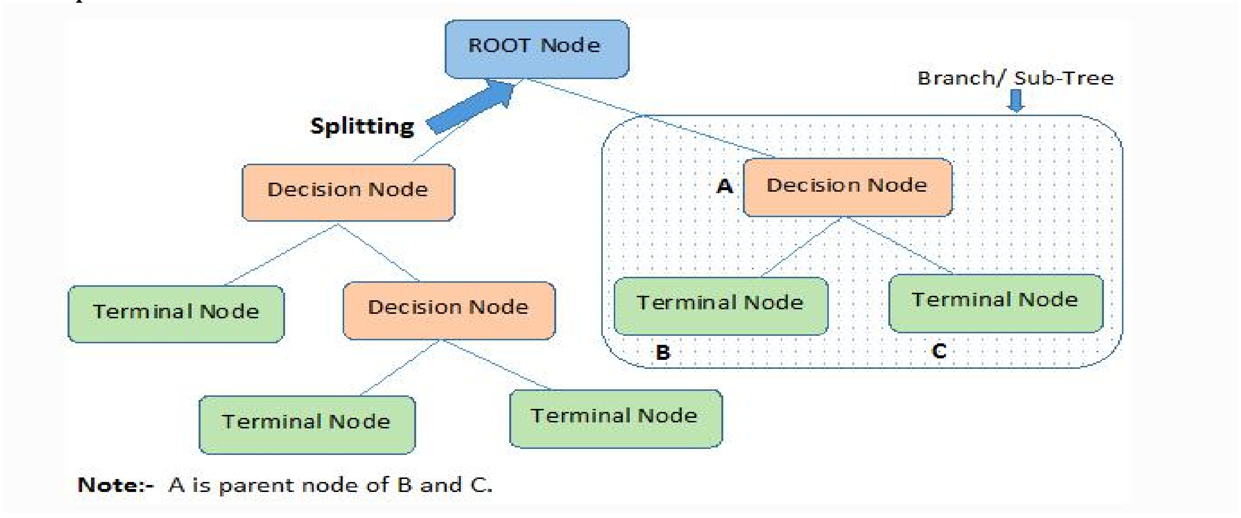
In this project, we focus on two aspects of the automated malware defense. For the First part of Our work , determine the best feature extraction, feature representation, and classification methods that result in the best accuracy when used on the top of Google Colab and Jupyter Notebook. Specifically Decision Trees, Gradient Boosting, Adaptive Boosting, Gaussian Naive Bayes and Random Forest classifiers were evaluated.

In the Second Part of the project we Implemented we used Artificial Neural Networks to predict the attacks. We experimented with adding different numbers of system features and hidden neural layers to predict the accuracy of the model. Finally we used the gradient boosting framework to train the Model.

7. **Implantation Of Machine Learning Models** :

7.1 Decision Tree Algorithm :

As it implies from the name, decision trees are data structures that have a structure of the tree. The training data set is used for the creation of the tree, that is subsequently used for making predictions on the test data. In this algorithm, the goal is to achieve the most accurate result with the least number of the decisions that must be made. Decision trees can be used for both classification and regression problems.



From the decision tree we can calculate the entropy in the following way and from that we can calculate the information gain.

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)

= Entropy (0.36, 0.64)

= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

$$E(T,X) = \sum_{c \in X} P(c)E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

$$Information\ Gain\ =\ Entropy(before) - \sum_{j=1}^{K} Entropy(j,\ after)$$

7.2 Gaussian Naive Bayes Algorithm :

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

Abstractly, naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $X = (x_1, x_2, \dots\dots, x_n)$ representing some *n* features (independent variables), it assigns to this instance probabilities $p(C_k \mid x_1, x_2, \dots\dots, x_n)$.

for each of K possible outcomes or *classes* C_k.

$$p(C_k \mid x) = \frac{p(C_k)\,p(x \mid C_k)}{p(x)} \qquad\qquad posterior = \frac{prior * likelihood}{evidence}$$

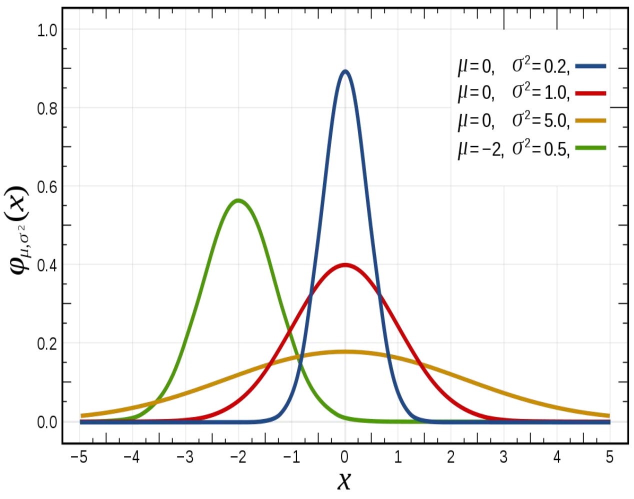
the conditional distribution over the class variable C is :

$$P(C_k \mid x_1, x_2, \dots, x_n) = \frac{1}{Z}\, p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k) \qquad where,\ Z = p(x) = \sum_k p(C_k)\, p(x \mid C_k)$$

is a scaling factor dependent only on $(x_1, \dots\dots, x_n)$, that is, a constant if the values of the feature variables are known.

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution.Gaussian distribution shown in the Graph.

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\alpha_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\alpha_k^2}}$$



7.3 Random Forest Algorithm :

It is a tree-based technique that uses a high number of decision trees built out of randomly selected sets of features. Contrary to the simple decision tree, it is highly uninterpretable but its generally good performance makes it a popular algorithm.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, \dots, x_n$ with responses $Y = y_1, \dots, y_n$, bagging repeatedly (*B* times) selects a random sample with replacement of the training set and fits trees to these samples: For $b = 1, \dots, B$:

1. Sample, with replacement, *n* training examples from *X*, *Y*; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x':

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}.$$

7.4 Gradient Boosting Algorithm :

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

We want our predictions, such that our loss function (MSE) is minimum. By using **gradient descent** and updating our predictions based on a learning rate, we can find the values where MSE is minimum.

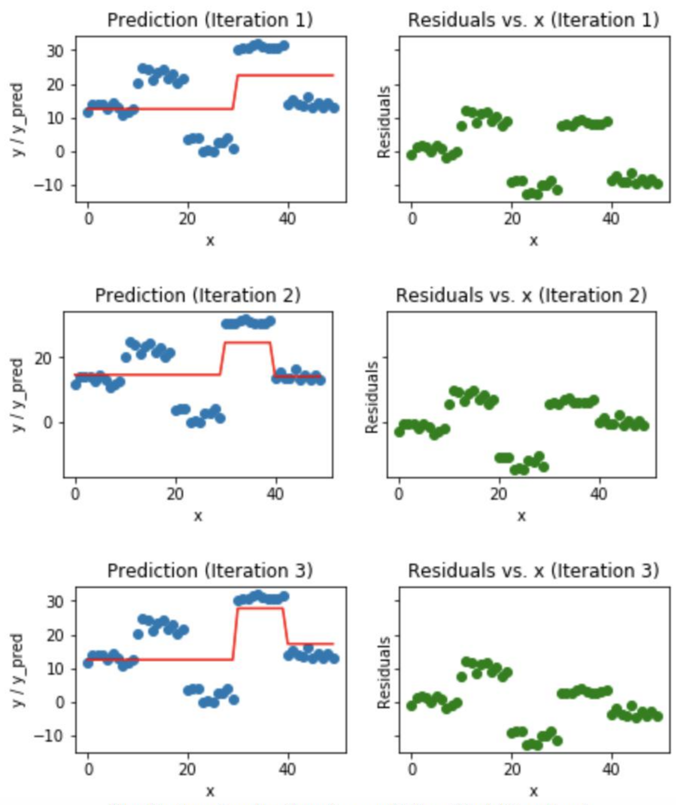
Loss = MSE = $\sum(y_i - y_i^p)^2$

Where, y_i = ith target value, y_i^p = ith prediction, $L(y_i, y_i^p)$ is loss function.

After applying Gradient Boosting,

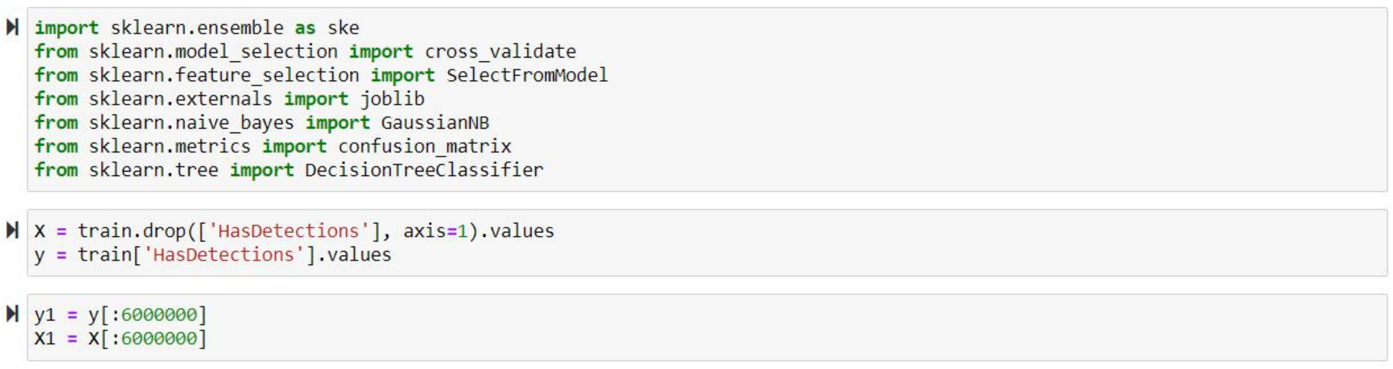
$y_i^p = y_i^p + \alpha * \delta\sum(y_i - y_i^p)^2 / \delta y_i^p$ which becomes, $y_i^p = y_i^p - \alpha * 2 * \sum(y_i - y_i^p)$

Where, α is learning rate and $\sum(y_i - y_p)$ is sum of residuals.



Visualization of gradient boosting predictions (First 4 iterations)

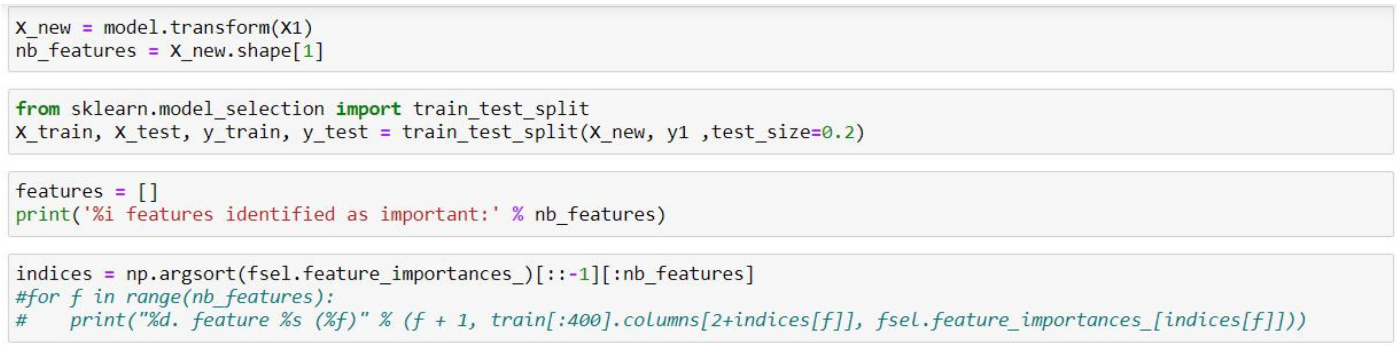
We train our data set using the classifiers mentioned above :



```
import sklearn.ensemble as ske
from sklearn.model_selection import cross_validate
from sklearn.feature_selection import SelectFromModel
from sklearn.externals import joblib
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
```

```
X = train.drop(['HasDetections'], axis=1).values
y = train['HasDetections'].values
```

```
y1 = y[:6000000]
X1 = X[:6000000]
```

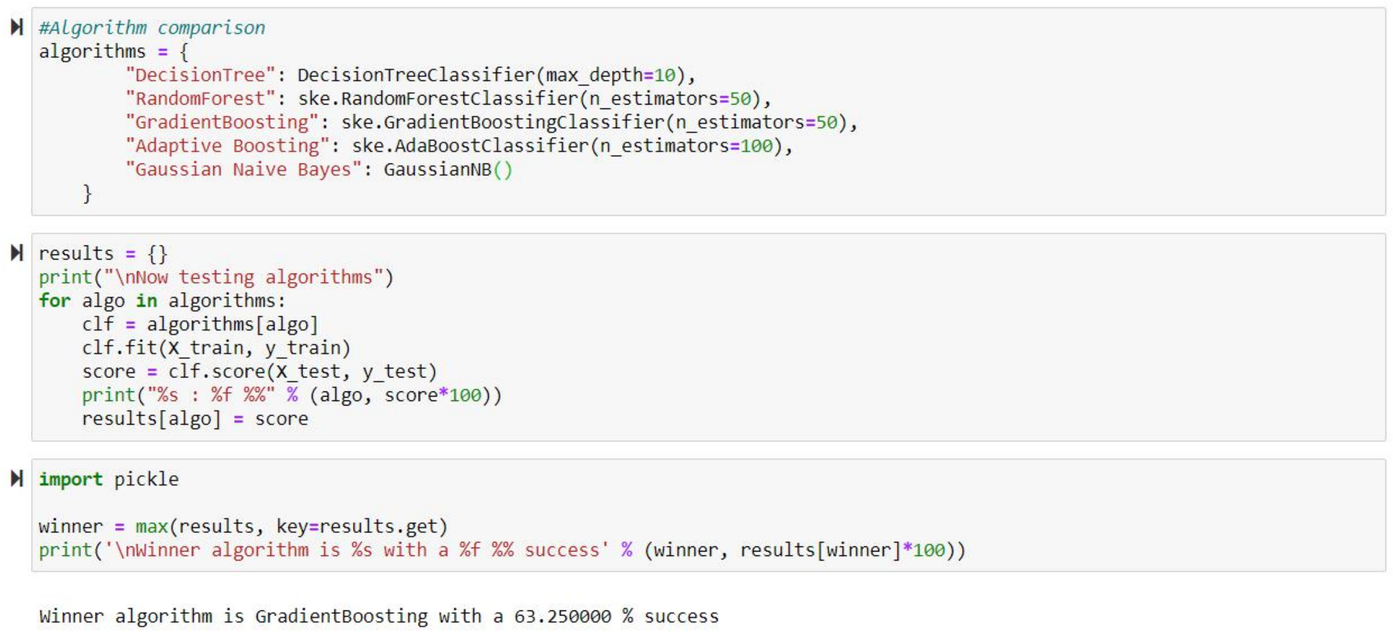


```
X_new = model.transform(X1)
nb_features = X_new.shape[1]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_new, y1 ,test_size=0.2)
```

```
features = []
print('%i features identified as important:' % nb_features)
```

```
indices = np.argsort(fsel.feature_importances_)[::-1][:nb_features]
#for f in range(nb_features):
#    print("%d. feature %s (%f)" % (f + 1, train[:400].columns[2+indices[f]], fsel.feature_importances_[indices[f]]))
```



```
#Algorithm comparison
algorithms = {
        "DecisionTree": DecisionTreeClassifier(max_depth=10),
        "RandomForest": ske.RandomForestClassifier(n_estimators=50),
        "GradientBoosting": ske.GradientBoostingClassifier(n_estimators=50),
        "Adaptive Boosting": ske.AdaBoostClassifier(n_estimators=100),
        "Gaussian Naive Bayes": GaussianNB()
    }
```

```
results = {}
print("\nNow testing algorithms")
for algo in algorithms:
    clf = algorithms[algo]
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    print("%s : %f %%" % (algo, score*100))
    results[algo] = score
```

```
import pickle

winner = max(results, key=results.get)
print('\nWinner algorithm is %s with a %f %% success' % (winner, results[winner]*100))
```

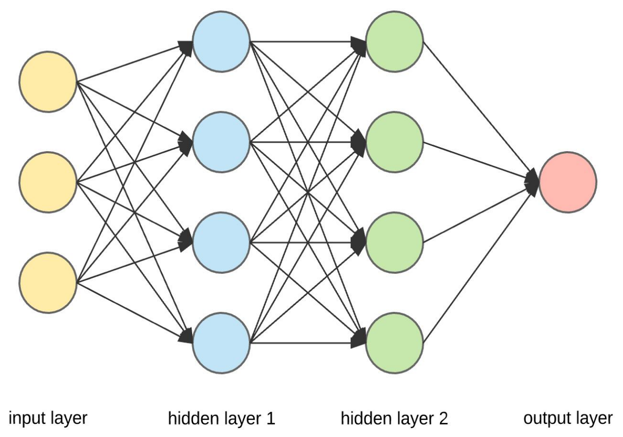
```
Winner algorithm is GradientBoosting with a 63.250000 % success
```

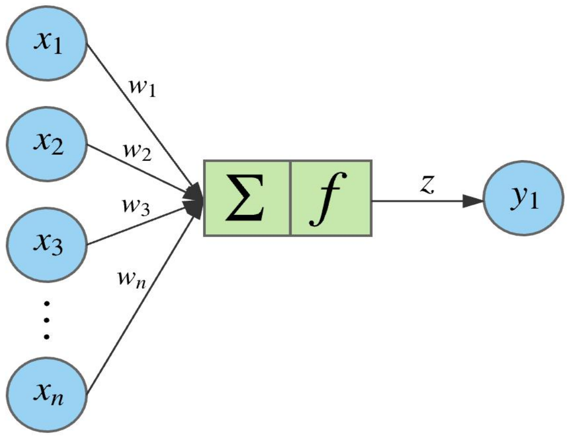
Using these Algorithms we got 60-63% accuracy. Among the algorithms

8. Implantation Of Neural Network Model :

Artificial neural networks are a family of models effective at solving problems of function approximation, pattern recognition, classification, and clustering.

ANNs are composed of artificial neurons which retain the biological concept of neurons, which receive input, combine the input with their internal state (activation) and an optional threshold using an activation function, and produce output using an output function.





The equation for a given node looks as follows. The weighted sum of its inputs passed through a non-linear activation function. It can be represented as a vector dot product, where n is the number of inputs for the node.

$$z = f(x \cdot w) = f\left(\sum_{i=1}^{n} x_i w_i\right)$$

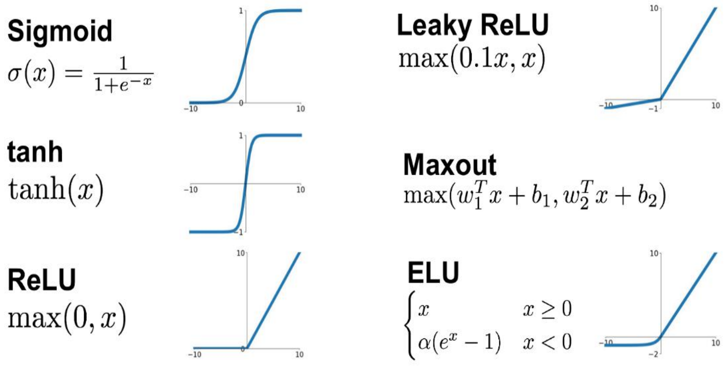
$$x \in d_{1\times n},\ w \in d_{n\times 1},\ z \in d_{1\times 1}$$

For completeness, the above equation looks as follows with the bias included.

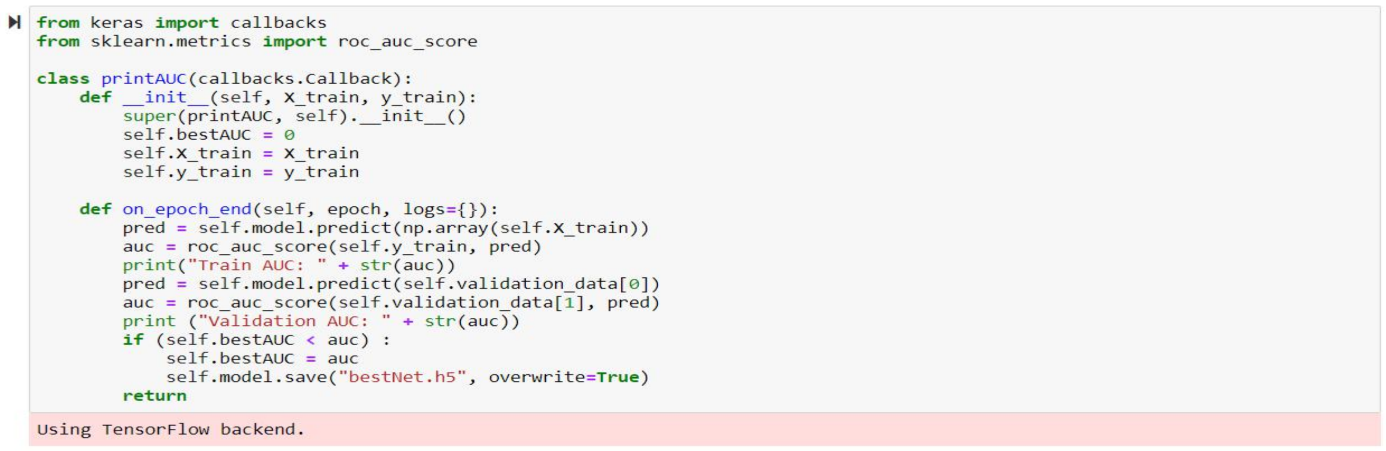
$$z = f(b + x \cdot w) = f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

$$x \in d_{1\times n},\ w \in d_{n\times 1},\ b \in d_{1\times 1},\ z \in d_{1\times 1}$$

In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs. A standard integrated circuit can be seen as a digital network of activation functions that can be "ON" or "OFF", depending on input. There are different types of activation functions are there among them we use 'sigmoid' and 'relu' function.



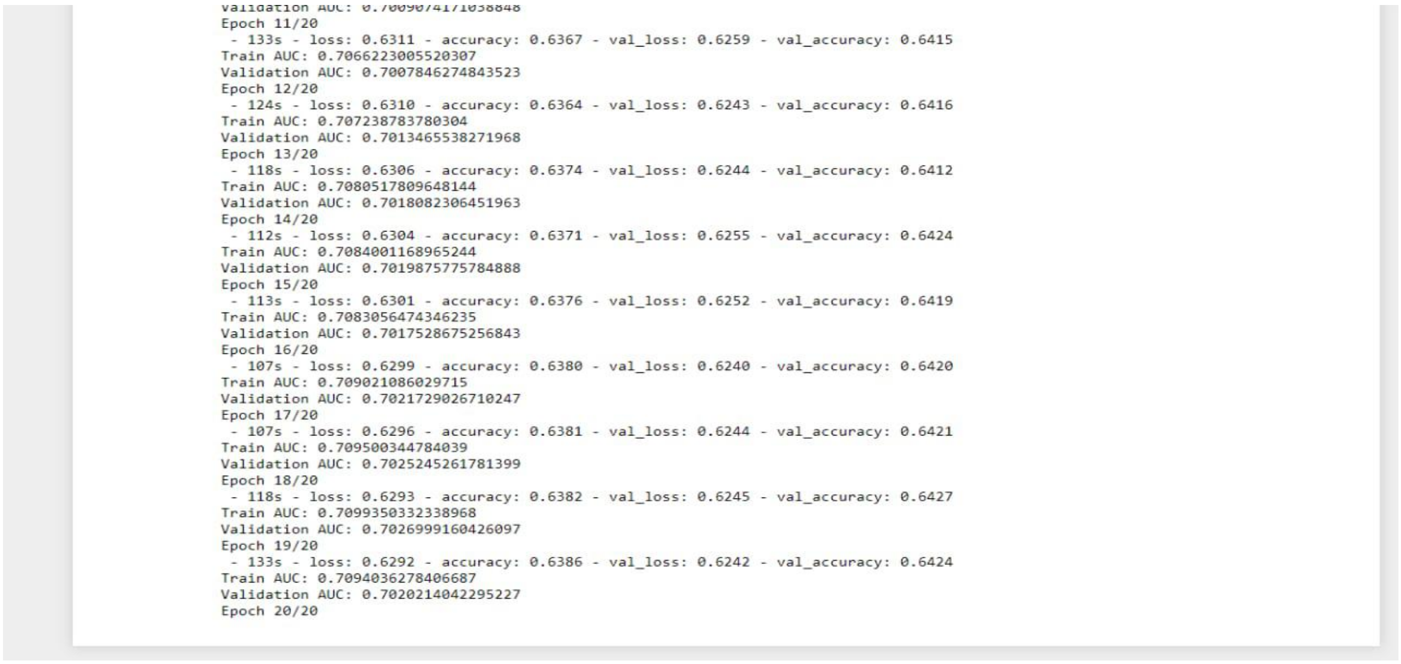
We train our model using ANN in following way :



```
from keras import callbacks
from sklearn.metrics import roc_auc_score

class printAUC(callbacks.Callback):
    def __init__(self, X_train, y_train):
        super(printAUC, self).__init__()
        self.bestAUC = 0
        self.X_train = X_train
        self.y_train = y_train

    def on_epoch_end(self, epoch, logs={}):
        pred = self.model.predict(np.array(self.X_train))
        auc = roc_auc_score(self.y_train, pred)
        print("Train AUC: " + str(auc))
        pred = self.model.predict(self.validation_data[0])
        auc = roc_auc_score(self.validation_data[1], pred)
        print ("Validation AUC: " + str(auc))
        if (self.bestAUC < auc) :
            self.bestAUC = auc
            self.model.save("bestNet.h5", overwrite=True)
        return
```

```
Using TensorFlow backend.
```

```
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization, Activation
from keras.callbacks import LearningRateScheduler
from keras.optimizers import Adam

#SPLIT TRAIN AND VALIDATION SET
X_train, X_val, Y_train, Y_val = train_test_split(df_train[cols], df_train['HasDetections'], test_size = 0.5)

# BUILD MODEL
model = Sequential()
model.add(Dense(100,input_dim=len(cols)))
model.add(Dropout(0.4))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(100))
model.add(Dropout(0.4))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(100))
model.add(Dropout(0.4))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=Adam(lr=0.01), loss="binary_crossentropy", metrics=["accuracy"])
annealer = LearningRateScheduler(lambda x: 1e-2 * 0.95 ** x)

# TRAIN MODEL
model.fit(X_train,Y_train, batch_size=32, epochs = 20, callbacks=[annealer,
          printAUC(X_train, Y_train)], validation_data = (X_val,Y_val), verbose=2)
```



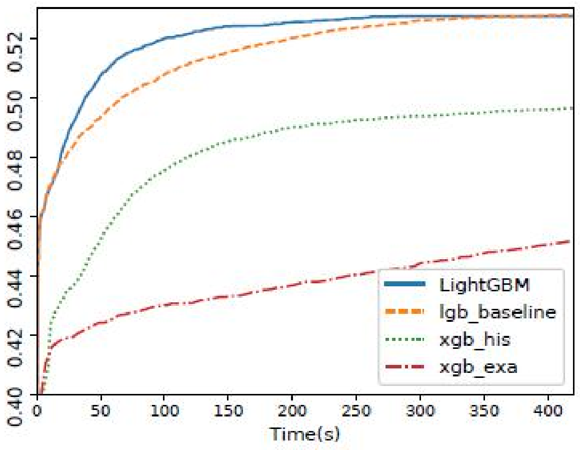
```
Validation AUC: 0.7009074171038848
Epoch 11/20
 - 133s - loss: 0.6311 - accuracy: 0.6367 - val_loss: 0.6259 - val_accuracy: 0.6415
Train AUC: 0.7066223005520307
Validation AUC: 0.7007846274843523
Epoch 12/20
 - 124s - loss: 0.6310 - accuracy: 0.6364 - val_loss: 0.6243 - val_accuracy: 0.6416
Train AUC: 0.707238783780304
Validation AUC: 0.7013465538271968
Epoch 13/20
 - 118s - loss: 0.6306 - accuracy: 0.6374 - val_loss: 0.6244 - val_accuracy: 0.6412
Train AUC: 0.7080517809648144
Validation AUC: 0.7018082306451963
Epoch 14/20
 - 112s - loss: 0.6304 - accuracy: 0.6371 - val_loss: 0.6255 - val_accuracy: 0.6424
Train AUC: 0.7084001168965244
Validation AUC: 0.7019875775784888
Epoch 15/20
 - 113s - loss: 0.6301 - accuracy: 0.6376 - val_loss: 0.6252 - val_accuracy: 0.6419
Train AUC: 0.7083056474346235
Validation AUC: 0.7017528675256843
Epoch 16/20
 - 107s - loss: 0.6299 - accuracy: 0.6380 - val_loss: 0.6240 - val_accuracy: 0.6420
Train AUC: 0.709021086029715
Validation AUC: 0.7021729026710247
Epoch 17/20
 - 107s - loss: 0.6296 - accuracy: 0.6381 - val_loss: 0.6244 - val_accuracy: 0.6421
Train AUC: 0.709500344784039
Validation AUC: 0.7025245261781399
Epoch 18/20
 - 118s - loss: 0.6293 - accuracy: 0.6382 - val_loss: 0.6245 - val_accuracy: 0.6427
Train AUC: 0.7099350332338968
Validation AUC: 0.7026999160426097
Epoch 19/20
 - 133s - loss: 0.6292 - accuracy: 0.6386 - val_loss: 0.6242 - val_accuracy: 0.6424
Train AUC: 0.7094036278406687
Validation AUC: 0.7020214042295227
Epoch 20/20
```

9. <u>Implementation Of Gradient Boosting Framework</u> :

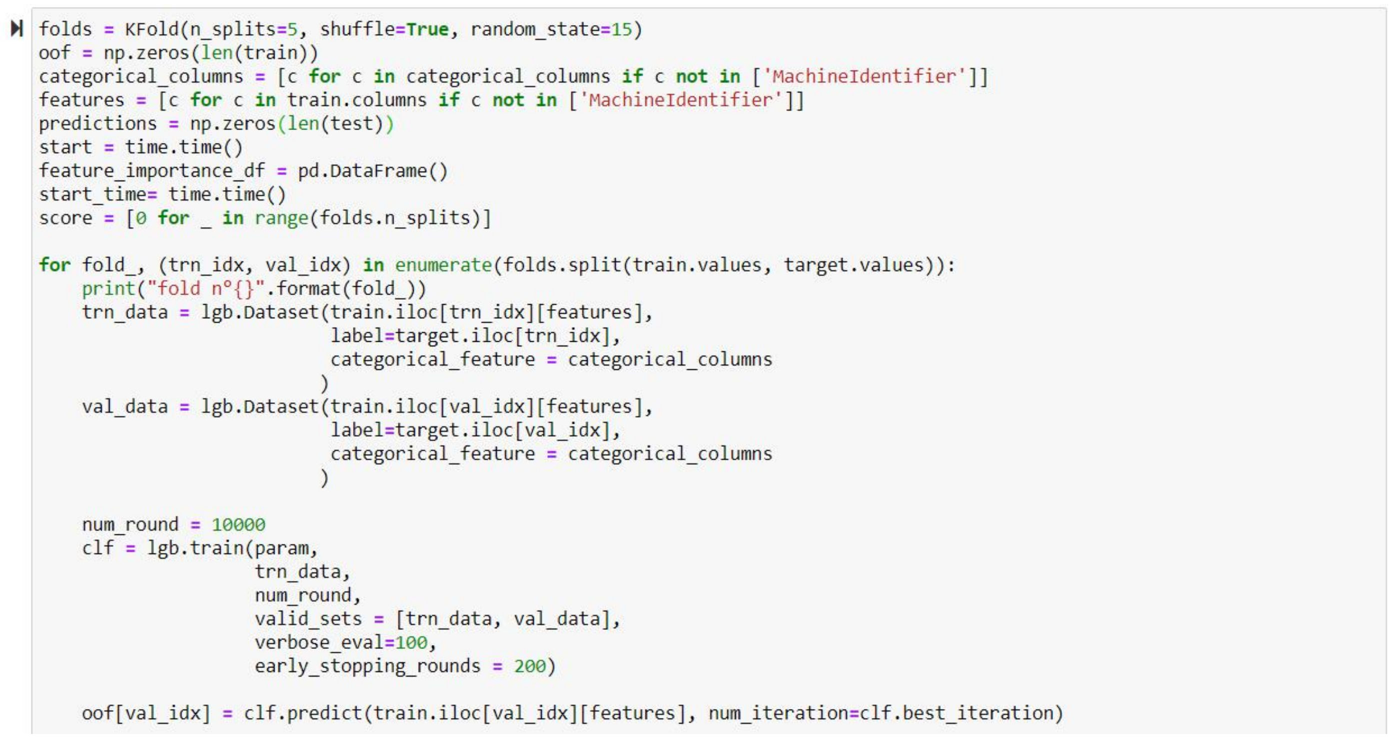
gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

A. Faster training speed and higher efficiency.
B. Lower memory usage.
C. Better accuracy.
D. Support of parallel and GPU learning.
E. Capable of handling large-scale data.

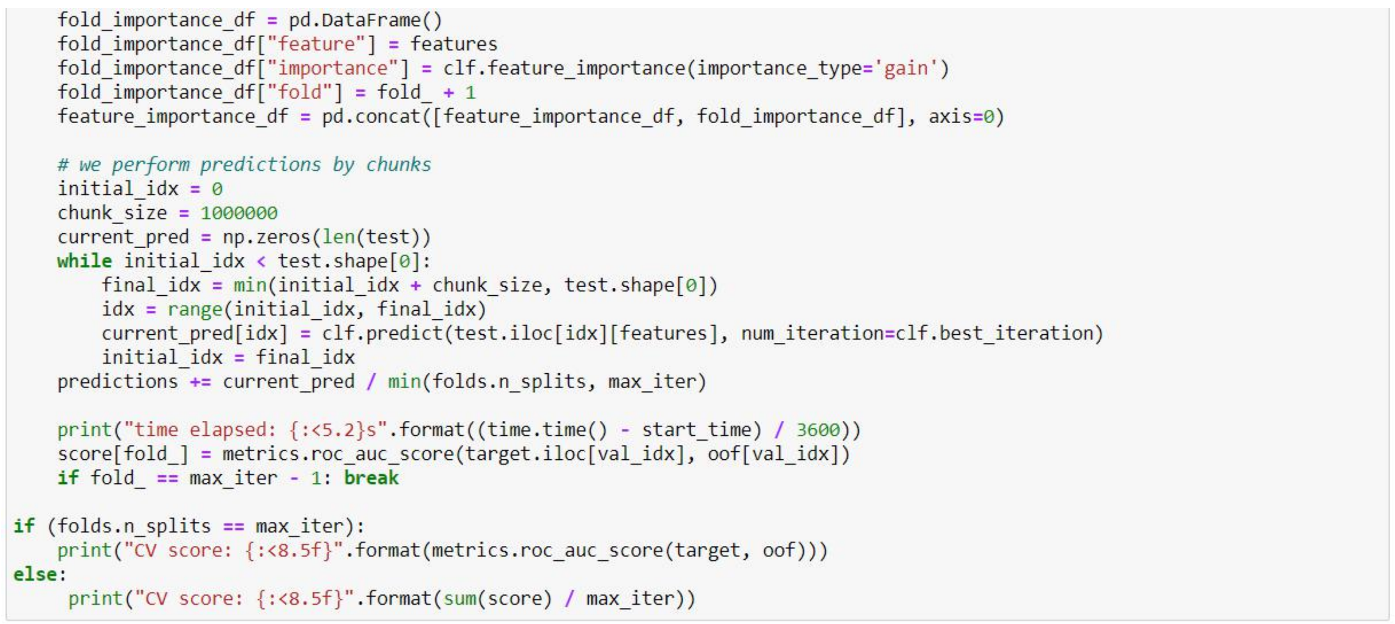
Comparison of Gradient Boosting Frameworks.We are using lightGBM framework here.



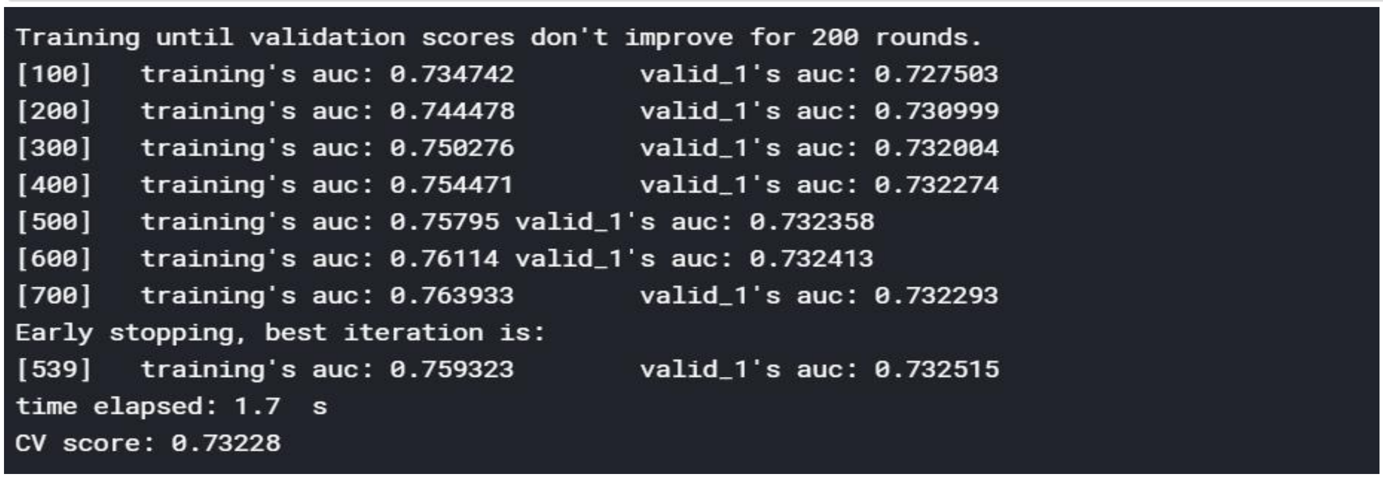
Screen shots of Implementation of the model is showing below :



```
folds = KFold(n_splits=5, shuffle=True, random_state=15)
oof = np.zeros(len(train))
categorical_columns = [c for c in categorical_columns if c not in ['MachineIdentifier']]
features = [c for c in train.columns if c not in ['MachineIdentifier']]
predictions = np.zeros(len(test))
start = time.time()
feature_importance_df = pd.DataFrame()
start_time= time.time()
score = [0 for _ in range(folds.n_splits)]

for fold_, (trn_idx, val_idx) in enumerate(folds.split(train.values, target.values)):
    print("fold n°{}".format(fold_))
    trn_data = lgb.Dataset(train.iloc[trn_idx][features],
                           label=target.iloc[trn_idx],
                           categorical_feature = categorical_columns
                          )
    val_data = lgb.Dataset(train.iloc[val_idx][features],
                           label=target.iloc[val_idx],
                           categorical_feature = categorical_columns
                          )

    num_round = 10000
    clf = lgb.train(param,
                    trn_data,
                    num_round,
                    valid_sets = [trn_data, val_data],
                    verbose_eval=100,
                    early_stopping_rounds = 200)

    oof[val_idx] = clf.predict(train.iloc[val_idx][features], num_iteration=clf.best_iteration)
```



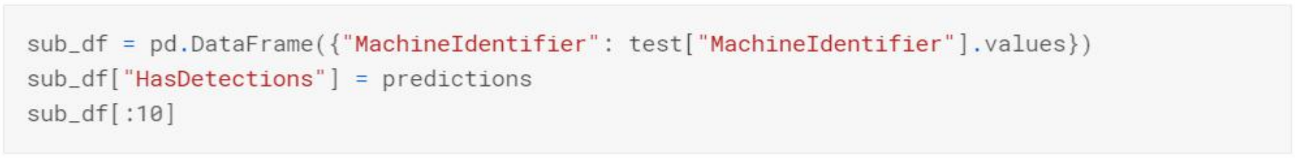
```
    fold_importance_df = pd.DataFrame()
    fold_importance_df["feature"] = features
    fold_importance_df["importance"] = clf.feature_importance(importance_type='gain')
    fold_importance_df["fold"] = fold_ + 1
    feature_importance_df = pd.concat([feature_importance_df, fold_importance_df], axis=0)

    # we perform predictions by chunks
    initial_idx = 0
    chunk_size = 1000000
    current_pred = np.zeros(len(test))
    while initial_idx < test.shape[0]:
        final_idx = min(initial_idx + chunk_size, test.shape[0])
        idx = range(initial_idx, final_idx)
        current_pred[idx] = clf.predict(test.iloc[idx][features], num_iteration=clf.best_iteration)
        initial_idx = final_idx
    predictions += current_pred / min(folds.n_splits, max_iter)

    print("time elapsed: {:<5.2}s".format((time.time() - start_time) / 3600))
    score[fold_] = metrics.roc_auc_score(target.iloc[val_idx], oof[val_idx])
    if fold_ == max_iter - 1: break

if (folds.n_splits == max_iter):
    print("CV score: {:<8.5f}".format(metrics.roc_auc_score(target, oof)))
else:
     print("CV score: {:<8.5f}".format(sum(score) / max_iter))
```



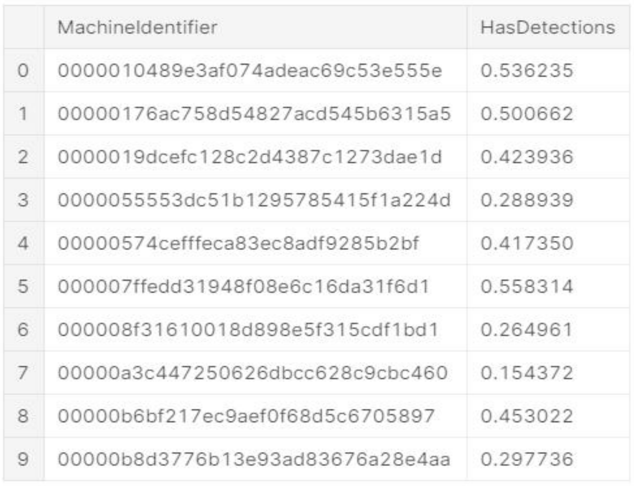
```
Training until validation scores don't improve for 200 rounds.
[100]   training's auc: 0.734742        valid_1's auc: 0.727503
[200]   training's auc: 0.744478        valid_1's auc: 0.730999
[300]   training's auc: 0.750276        valid_1's auc: 0.732004
[400]   training's auc: 0.754471        valid_1's auc: 0.732274
[500]   training's auc: 0.75795 valid_1's auc: 0.732358
[600]   training's auc: 0.76114 valid_1's auc: 0.732413
[700]   training's auc: 0.763933        valid_1's auc: 0.732293
Early stopping, best iteration is:
[539]   training's auc: 0.759323        valid_1's auc: 0.732515
time elapsed: 1.7   s
CV score: 0.73228
```

10. Comparison Of different methods based on accuracy:

So after Training all the models we get 60-63% accuracy in Machine learning based Algorithms. ANN's generates 70-73% accuracy and using Gradient boosting framework generates 73-75% accuracy. So we Finally looking at Gradient Boosting Framework to implement the Model.
Final Prediction Using the Models shown bellow:



```
sub_df = pd.DataFrame({"MachineIdentifier": test["MachineIdentifier"].values})
sub_df["HasDetections"] = predictions
sub_df[:10]
```



	MachineIdentifier	HasDetections
0	0000010489e3af074adeac69c53e555e	0.536235
1	00000176ac758d54827acd545b6315a5	0.500662
2	0000019dcefc128c2d4387c1273dae1d	0.423936
3	0000055553dc51b1295785415f1a224d	0.288939
4	00000574cefffeca83ec8adf9285b2bf	0.417350
5	000007ffedd31948f08e6c16da31f6d1	0.558314
6	000008f31610018d898e5f315cdf1bd1	0.264961
7	00000a3c447250626dbcc628c9cbc460	0.154372
8	00000b6bf217ec9aef0f68d5c6705897	0.453022
9	00000b8d3776b13e93ad83676a28e4aa	0.297736

11. Results and Discussions :

The project aims at malware attack detection on based level in time of manufacturing Using Machines features available. Not much research papers available on this area but there are many Challenges thrown by Industries for data science communities but there are not any such model available on this area which gives a 80%+ accuracy, most of them are near 70% accuracy so our aim is to do that. We train this model using different classifiers and get up to 75% accuracy.

11.1 Future Scope :

The features used in this work were extracted through static analysis. It has been shown that the use of more advanced binary packers by malware authors can make disassembling, which is required for extraction these static analysis features, difficult. One way to handle this is to use dynamic analysis to run the packed program and dump process memory image once the malware has unpacked itself. After that, the feature extraction approaches proposed in this work can be applied the process code segment dump.
In other hand we can more update the models. Try with more different Classifiers for getting the bet result. Or we can think about other ways to train the model using variable chunks of data and epochs.

12. Conclusions:

Anti-malware vendors receive a large number of suspected malware files daily. To cope with this influx of malware files, machine learning techniques, such as classification and clustering, are used to group similar malware into families. Grouping malware into families allows human analysts to examine representative samples from each group instead of examining all file. The effectiveness of the machine learning techniques heavily relies on how much discriminative information the features extracted from these files carry. Since it is difficult to collect training samples from each malware family, the classification and clustering systems are forced to operate in an open set scenario where they are faced with instances from never before seen families. In this dissertation, we worked on extracting useful malware features and proposed the malware classification and clustering systems capable of operating in an open set scenario.The research could be further extended with various data sets and various attributes for the ensemble of the machine learning algorithms

13. Appendix B :

The team comprised of three members and each one of them had contributed well on their part.The efficiency of the team is surely reflected in the work showcased above.

Sagnik Mukherjee(CSCE,1729215) : Worked On different Machine Learning Models. The start included study and construe of the renowned research papers and get the idea as to how to implement the project in terms of documentation , writing the research paper.Analysis of the research papers has been done and their outcomes are compared . Project planning and methodology lying under the work exhibited.

Toulik Das(CSCE,1729231) : Worked On different ANN Models and Gradient Boosting Frameworks. Had done research work based on the topics relevant to the project and found the papers holding much importance for the project.Equal contribution in writing the research paper which is an ongoing work as of now. Helped with the documentation part basis classifiers and their working.. Project planning and methodology lying under the work exhibited.

Rupam Patra(CSCE,1729215) : Worked On different Data Visualization Tools and Data Pre-processing stage.Had equal contribution in study of the research papers.Took the initiative to design the presentation for the same with the help of the second member Toulik Das.Worked out with the code to implement ANN model. Project planning and methodology lying under the work exhibited.

14. References:

1) Sukriti Bhattacharya, Hector D. Menendez, Earl T. Barr, and David Clark. Itect: Scalable information theoretic similarity for malware detection. CoRR, abs/1609.02404, 2016. URL: http://arxiv.org/abs/1609.02404, arXiv:1609.02404.

2) Z. Berkay Celik, Patrick McDaniel, and Rauf Izmailov. Feature cultivation in privileged information-augmented detection. In Proceedings of the 3rd ACM on Interna-tional Workshop on Security And Privacy Analytics, IWSPA '17, pages 73–0, New York, NY, USA, 2017. ACM. URL: http://doi.acm.org/10.1145/3041008.3041018, doi:10.1145/3041008.3041018.

3) J. Fan, C. Guan, K. Ren, Y. Cui, and C. Qiao. Spabox: Safeguarding privacy during deep packet inspection at a middlebox. IEEE/ACM Transactions on Networking, 25(6):3753–3766, Dec 2017. doi:10.1109/TNET.2017.2753044.

4)X. Hu, J. Jang, T. Wang, Z. Ashraf, M. P. Stoecklin, and D. Kirat. Scalable malware classification with multifaceted content features and threat intelligence. IBM Journal of Research and Development, 60(4):6:1–6:11, July 2016. doi:10.1147/JRD.2016.2559378.

5) Jin-Young Kim, Seok-Jun Bu, and Sung-Bae Cho. Malware detection using deep transferred generative adversarial networks. In Derong Liu, Shengli Xie, Yuanqing Li, Dongbin Zhao, and El-Sayed M. El-Alfy, editors, Neural Information Processing, pages 556–564, Cham, 2017. Springer International Publishing.