

Microsoft Malware Prediction

*Project report submitted to Visvesvaraya National
Institute of Technology, Nagpur in partial fulfillment of
the requirements for the award of the degree*

Bachelor of Technology In Electronics and Communication Engineering

by

Nayan Nimbokar (BT17ECE052)

Prateek Jain (BT17ECE057)

Prateek R Bhansali (BT17ECE058)

under the guidance of

Dr. Joydeep Sengupta



Department of Electronics and Communication Engineering

Visvesvaraya National Institute of Technology

Nagpur 440 010 (India) 2020-21


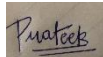
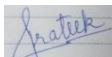
Department of Electronics and Communication Engineering

Visvesvaraya National Institute of Technology, Nagpur



Declaration

We, **Nayan Nimbokar, Prateek Jain and Prateek Bhansali**, hereby declare that this project work titled “**Microsoft Malware Prediction**” is carried out by us in the **Department of Electronics and Communication Engineering of Visvesvaraya National Institute of Technology, Nagpur**. The work is original and has not been submitted earlier whole or in part for the award of any degree/diploma at this or any other Institution

Sr.No.	Enrollment No	Name	Signature
1	BT17ECE052	Nayan Nimbokar	
2	BT17ECE057	Prateek Jain	
3	BT17ECE058	Prateek R Bhansali	

Certificate

This is to certify that the project titled “**Microsoft Malware Prediction**”, submitted by **Nayan Nimbokar, Prateek Jain and Prateek Bhansali** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Electronics and Communication Engineering, VNIT Nagpur**. The work is comprehensive, complete and fit for final evaluation.

Dr. A.G.Keskar

Professor and Head of Department

Electronics and Communication

Engineering VNIT, Nagpur



Dr. Joydeep Sengupta

Professor

Electronics and Communication

Engineering VNIT, Nagpur

Visvesvaraya National Institute of Technology

Nagpur

Non-Plagiarism Certificate

This is to certify that the B. Tech project titled “**Microsoft Malware Prediction**” submitted by us has been checked for plagiarism using TURNITIN software and the overlap is found within prescribed limits. The complete report is attached herewith. The summary of the report is given below.

Microsoft Malware Prediction

ORIGINALITY REPORT

10%

SIMILARITY INDEX

4%

INTERNET SOURCES

4%

PUBLICATIONS

7%

STUDENT PAPERS

Student Name: Nayan Nimbokar

Roll No.: BT17ECE052

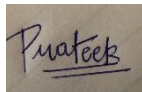
ID NO.: 21366



Student Name: Prateek Jain

Roll No.: BT17ECE057

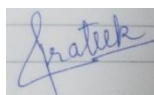
ID NO.: 21325



Student Name: Prateek R Bhansali

Roll No.: BT17ECE058

ID NO.: 21019



Acknowledgement

We'd like to share our appreciation and deep respect for our mentor, Dr. Joydeep Sengupta, for his unwavering assistance. His vast knowledge and expertise came in handy during the course of this project and the writing of this thesis. It was an honor to work and research under his guidance. We would like to use this time to express our gratitude to all of the Electronics and Communication Engineering department's faculty members for their valuable contributions and guidance.

Nayan Nimbokar (BT17ECE052)

Prateek Jain (BT17ECE057)

Prateek Bhansali (BT17ECE058)

INDEX

Declaration & Certificate

Plagiarism Certificate

Abstract.....i

List of Figuresii

List of Tables.....iii

Nomenclature.....iv

Chapters

1. INTRODUCTION1

1.1 Overview1

1.2 Problem Statement1

1.3 Objective2

1.4 Selection of Dataset3

2. EXPLORATORY DATA ANALYSIS4

2.1 Dataset Description4

2.2 Balance of Dataset4

2.3 Data Type5

2.4 Missing Values in Dataset5

2.5 Cardinality in Dataset8

3. Data Preprocessing10

3.1 What is Data Preprocessing ?11

3.1.1 Splitting features into binary, numeric and categorical.....11

3.1.2 Using Reduce Memory Function.....13

3.1.3	Deleting Columns with too many missing values	14
3.1.4	Removing columns with high cardinality	14
3.1.5	Filling missing values.....	14
3.1.6	Feature Engineering	15
3.1.7	One Hot Encoding	17
3.2	Resultant data after cleaning and preprocessing.....	17
3.3	Size and Runtime	19
4.	Machine Learning Techniques & Python Libraries.....	20
4.1	Light GBM	20
4.1.1	GOSS.....	20
4.1.2	Exclusive Feature Bundling Technique for Light GBM	21
4.1.3	Architecture of Light GBM Model	21
4.1.4	Parameter Tuning	21
4.2	Xgboost	22
4.2.1	Model Features	22
4.2.2	System Features	22
4.3	Decision Tree Classifier	23
4.4	Gradient Boosting classifier	23
4.5	Logistic Regression.....	23
4.6	Adaboost Classifier	23
4.7	Support Vector Machine (SVM).....	23
4.8	K-Neighbors Classifiers	24
4.9	Multi-Layer Perceptron Classifier.....	24
4.10	Random Forest Classifier.....	24

4.11 Linear Discriminant Analysis	25
4.12 Python Libraries	25
4.12.1 Import numpy as np.....	25
4.12.2 Import pandas as pd	25
4.12.3 From tqdm import tqdm.....	25
4.12.4 Import dask.dataframe as dd.....	26
4.12.5 Import seaborn as sns.....	26
4.12.6 Import matplotlib.pyplot as plt	26
5. Result of Light GBM Model	27
5.1 Model Selection and Data Building.....	27
5.2 Parameter tuning	28
5.3 Training	29
5.4 Model parameters	29
5.4.1 Cross Validation Score	29
5.4.2 AUC score	30
5.5 Prediction	31
5.6 Feature Importance.....	32
6. Results of Model comparison	34
6.1 Selecting Model	34
6.2 Model preparation.....	35
6.3 Parameters for evaluation	36
6.3.1 Cross Validation Score.....	36
6.3.2 Auc Score	38
6.3.3 Confusion Matrix	40

7. CONCLUSION	45
7.1 Conclusion	45
7.2 Future Work	46
Reference	47
APPENDIX A: PLAGIARISM REPORT	48

Abstract

The internet has been widely used in this new, technical era. As a result, the risk of cybercriminals launching malware attacks has risen. The number of malware samples captured and analysed by antivirus firms has increased exponentially in recent years. Malware is used in these attacks, which have caused billions of dollars in financial harm. As a result, malware intrusion detection has become a critical component of the fight against cybercrime. Machine Learning has grown in importance in the area of malware detection in recent years, and is the first step in eliminating malware from compromised computers. We use machine learning algorithms in this study to forecast the malware detection of computers based on their characteristics. We used different machine learning algorithms like gradient boosting, ada boost, neural network, Light GBM, Decision tree learning many more. We have collected the dataset which is available over the internet provided by microsoft on kaggle that has training set and testing set and then preprocessed the dataset and applied feature engineering over the dataset and trained it on different algorithms. And then we conclude that, LightGBM was discovered to be the best model with an AUC Score of **0.6652** after eleven separate tests given below in chapter 6. So Light GBM provides the better results along with great efficiency and low runtime. This shows that Light GBM is the best gradient boosting algorithm in predicting the malware detection problem.

List Of FIGURES

• Fig 1.1: Total Malware Infection Growth rate	2
• Fig 2.1: Count of HasDetection.....	4
• Fig 2.2: Column count based on data types.....	5
• Fig. 2.3 Missing values plot.....	8
• Fig 2.4: Cardinality plot.....	9
• Fig 3.1: Binary features.....	12
• Fig 3.2: Numerical features.....	12
• Fig 3.3: Categorical features.....	12
• Fig 3.4: Reduce memory function.....	13
• Fig 3.5: Column more than 60% null values.....	14
• Fig 3.6: Columns with high cardinality.....	14
• Fig 3.7: Total runtime for data processing notebook	19
• Fig 4.1 Leaf-Wise Tree Growth	21
• Fig 5.1: Hyperparameters	28
• Fig 5.2: Model training	29
• Fig 5.3: Confusion matrix	30
• Fig 5.4: ROC curve	31
• Fig 5.5: Probability density curve	32
• Fig 5.6: Probability distribution	32
• Fig 5.7: Feature importance	33
• Fig 6.1: Train-Test Split	35
• Fig 6.2: Training Data preparation	35
• Fig 6.3: Cross validation score	37
• Fig 6.4: ROC curve study	38
• Fig 6.5: ROC Curve analysis	39
• Fig 6.6: Confusion Matrix of Different Models	44

List of Tables

• Table 2.1: Missing Values.....	6
• Table 3.1: Resultant training data	18
• Table 3.2: Resultant testing data	18
• Table 3.3: Total memory utilization.....	19
• Table 5.1: Cross validation report	30
• Table 6.1: Comparison of confusion matrix parameters of Different Models.....	39
• Table 6.2: Confusion Matrix.....	41

Nomenclature

Sr. NO	Abbreviation	Description
1.	OS	Operating system
2.	NaN	Not a Number
3.	LGBM	Light Gradient Boosting Machine
4.	Goss	Gradient-based One Side Sampling technique
5.	GPU	Graphics Processing Unit
6.	XGBoost	eXtreme Gradient Boosting
7.	SVM	Support Vector Machine
8.	KNN	K-Nearest Neighbors
9.	MLP	Multi-Layer Perceptron
10.	LDA	Linear Discriminant Analysis
11.	AUC	Area under the ROC curve
12.	ROC	Receiver operating characteristic curve
13.	TP	True Positive
14.	TN	True Negative
15.	FP	False Positive
16.	FN	False Negative

CHAPTER 1

INTRODUCTION

1.1 Overview

Malware is malicious software that is having some codes that are built by cyberattackers. These software are harmful for a system as they try to make some changes in the operating system that can result into data breach. A malware attack is a common cyberattack where malware executes unauthorized actions on the victim's system. It is normally delivered over an email or any social media platforms with a link or file that gets automatically executed once clicked. Malware attacks are also done by cyberattackers to steal confidential data like financial details for their benefit. The victims can be entrepreneurs, enterprises, large corporations, or even governments. The common types of malware are worms, Trojan, Trojan Horse, Ransomware, spyware, etc. Malware is a collective name of all these.

1.2 Problem Statement

Microsoft Malware prediction is a project in which we predict the probability of a system getting attacked by malware before malware hits the system. From the last recent years, it is seen that there is exponential growth in the numbers of malware captured. Over the last decade, there has been an 87% increase in malware infections, and at present, over 92% of all malware is delivered by email. Nowadays cybercrime is one of the biggest threats to humankind, as it causes personal and financial damage to both individuals and organizations. Data taken from the internet shows the increase in malware from 2009 to 2018.

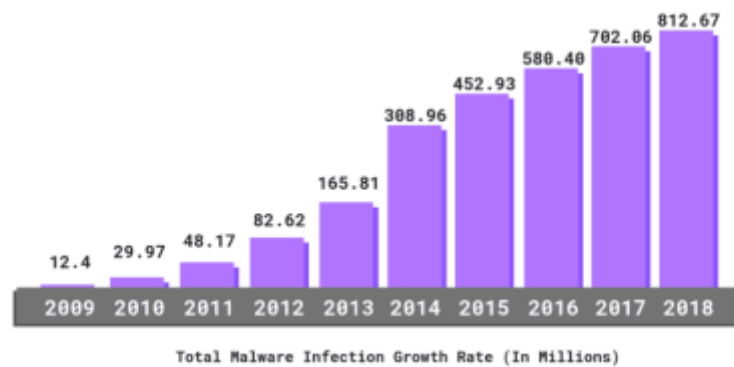


Fig 1.1: Total Malware Infection Growth rate

The WannaCry ransomware attack is the largest malware attack in history. The WannaCry ransomware attack was a global cyberattack launched in May 2017 by the WannaCry ransomware cryptoworm, which encrypts data and demands Bitcoin ransom payments from computers running the Microsoft Windows operating system. The incident was reported to have harmed over 200,000 computers in 150 nations, with estimated losses ranging from hundreds of millions to billions of dollars. Many attacks, like these, may occur, and many individuals or organisations may become victims. Malware is a well-organized and also well established industry dedicated to evading conventional security measures. Following the discovery of these issues, Microsoft took the issue seriously and invested heavily in improving security.

1.3 OBJECTIVE

Our objective is to identify the models that provide the highest accuracy by predicting the probability of a malware getting hit to a machine. We have done this by considering different properties of the machine. Microsoft has provided the dataset consisting 9 million different machine identifiers on kaggle to encourage open-source progress on effective techniques for predicting malware occurrences and challenge the community to protect more than one billion machines from damage before it happens.

1.4 Selection of Dataset

For Malware prediction, we need a big dataset so that we can train our algorithm and predict the probability with more accuracy and protect more than one billion machines from damage before it happens. Therefore after more research, we got the dataset and we decided to choose the kaggle dataset on Microsoft Malware Detection. As the kaggle dataset is highly organized and has all the required data like sample submission, training, and testing, the dataset is more reliable also, and also the dataset is suitable to study and preprocess easily. Many kaggle competitors used the dataset in the past and under the discussion section, many useful tips about preprocessing the dataset are given. Therefore we decided to use kaggle as our primary source of data.

CHAPTER 2

EXPLORATORY DATA ANALYSIS

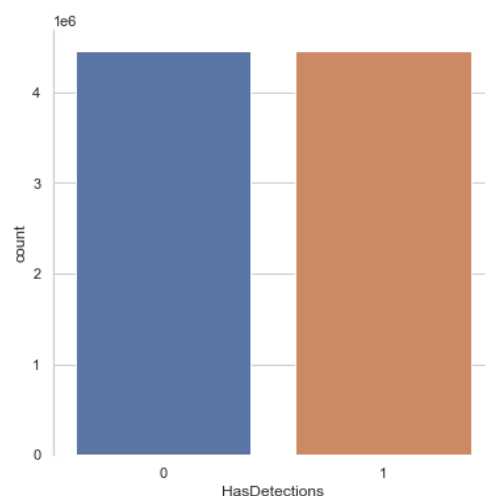
2.1 DATASET DESCRIPTION

The dataset which we used in this project is the dataset provided by Microsoft for Malware Prediction on Kaggle. The aim of the project is to determine the likelihood of a Windows machine being corrupted by different forms of malware based on the machine's various properties, so that infected computers can be quickly identified and restored. The dataset is large consisting of 9 million machine identifiers for training('train.csv') and 8 million for testing('test.csv'). The dataset contains total 82 features representing characteristics of a machine out of which 23 are numerically encoded to maintain the privacy.

2.2 BALANCE OF DATASET

First, we must decide if the dataset is unbalanced. Imbalanced data describes a condition in which the groups are not fairly represented. Since, they are sensitive to the proportions of the various classes, certain machine learning classifiers, such as Random Forest, are unable to handle imbalanced datasets. Such classifiers may have a bias for the class with the most observations. These qualified classifiers are often incorrect and may make incorrect predictions.

Fig 2.1: Count of HasDetection



2.3 DATA TYPE

A data type is a property of data that informs the compiler or interpreter how the programmer intends to use the information. A data type is a collection of values from which an expression (variable, function, and so on) can derive its values. All of the data provided by Microsoft is in int 64 format, but when we examined the dataset, we discovered that some of it is of the numerical, binary, and categorical types. So that after changing the data type, we can reduce the size of the dataset and use it more effectively.

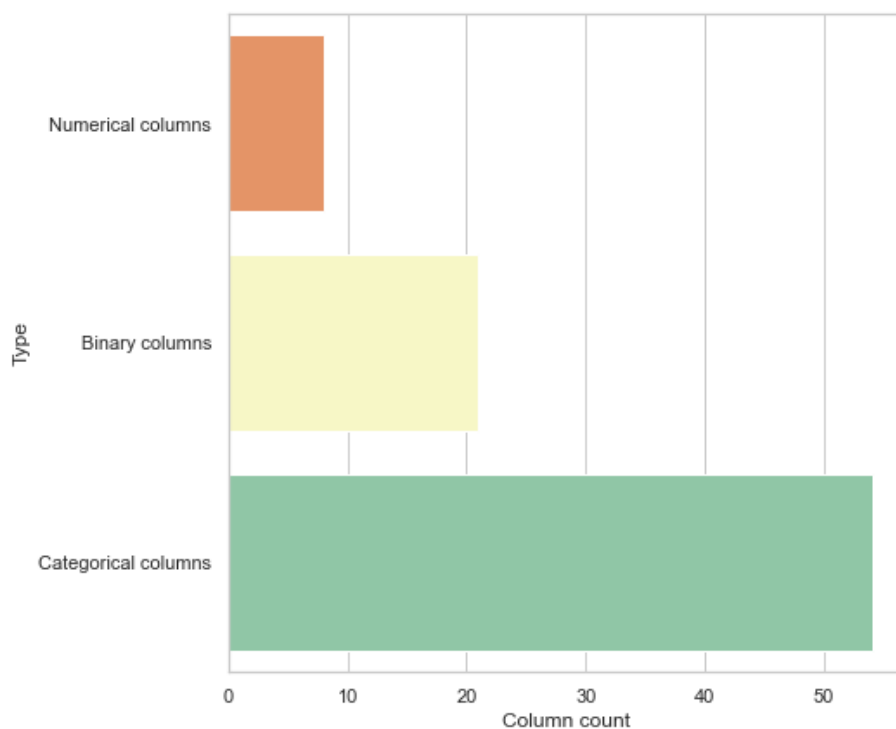


Fig 2.2: Column count based on data types

2.4 MISSING VALUES IN DATASET

Values that are not present or missing in the dataset are referred to as NaN values. Features with a high percentage of missing values give no meaningful information for research or training.

Table 2.1: Missing Values

Feature	Missing values	%
PuaMode	8919174	100.00
Census_ProcessorClass	8884852	99.60
DefaultBrowsersIdentifier	8488045	95.10
Census_IsFlightingInternal	7408759	83.00
Census_InternalBatteryType	6338429	71.00
Census_ThresholdOptIn	5667325	63.50
Census_IsWIMBootEnabled	5659703	63.40
SmartScreen	3177011	35.60
OrganizationIdentifier	2751518	30.80
SMode	537759	6.00
CityIdentifier	325409	3.60
Wdft_RegionIdentifier	303451	3.40
Wdft_IsGamer	303451	3.40
Census_InternalBatteryNumberOfCharges	268755	3.00
Census_FirmwareManufacturerIdentifier	183257	2.10
Census_IsFlightsDisabled	160523	1.80
Census_FirmwareVersionIdentifier	160133	1.80
Census_OEMModelIdentifier	102233	1.10
Census_OEMNameIdentifier	95478	1.10
Firewall	91350	1.00
Census_TotalPhysicalRAM	80533	0.90
Census_IsAlwaysOnAlwaysConnectedCapable	71343	0.80

Census_OSInstallLanguageIdentifier	60084	0.70
leVerIdentifier	58894	0.70
Census_PrimaryDiskTotalCapacity	53016	0.60
Census_SystemVolumeTotalCapacity	53002	0.60
Census_InternalPrimaryDiagonalDisplaySizeInInches	47134	0.50
Census_InternalPrimaryDisplayResolutionHorizontal	46986	0.50
Census_InternalPrimaryDisplayResolutionVertical	46986	0.50
Census_ProcessorModelIdentifier	41343	0.50
AVProductsEnabled	36221	0.40
AVProductsInstalled	36221	0.40
AVProductStatesIdentifier	36221	0.40
IsProtected	36044	0.40
RtpStateBitfield	32318	0.40
Census_IsVirtualDevice	15953	0.20
Census_PrimaryDiskTypeName	12844	0.10
UacLuaenable	10838	0.10
Census_ChassisTypeName	623	0.00
GeoNameIdentifier	213	0.00
Census_PowerPlatformRoleName	55	0.00
OsBuildLab	21	0.00



Fig. 2.3 Missing values plot

There are 44 columns that have missing values shown in figure 2.3 above.

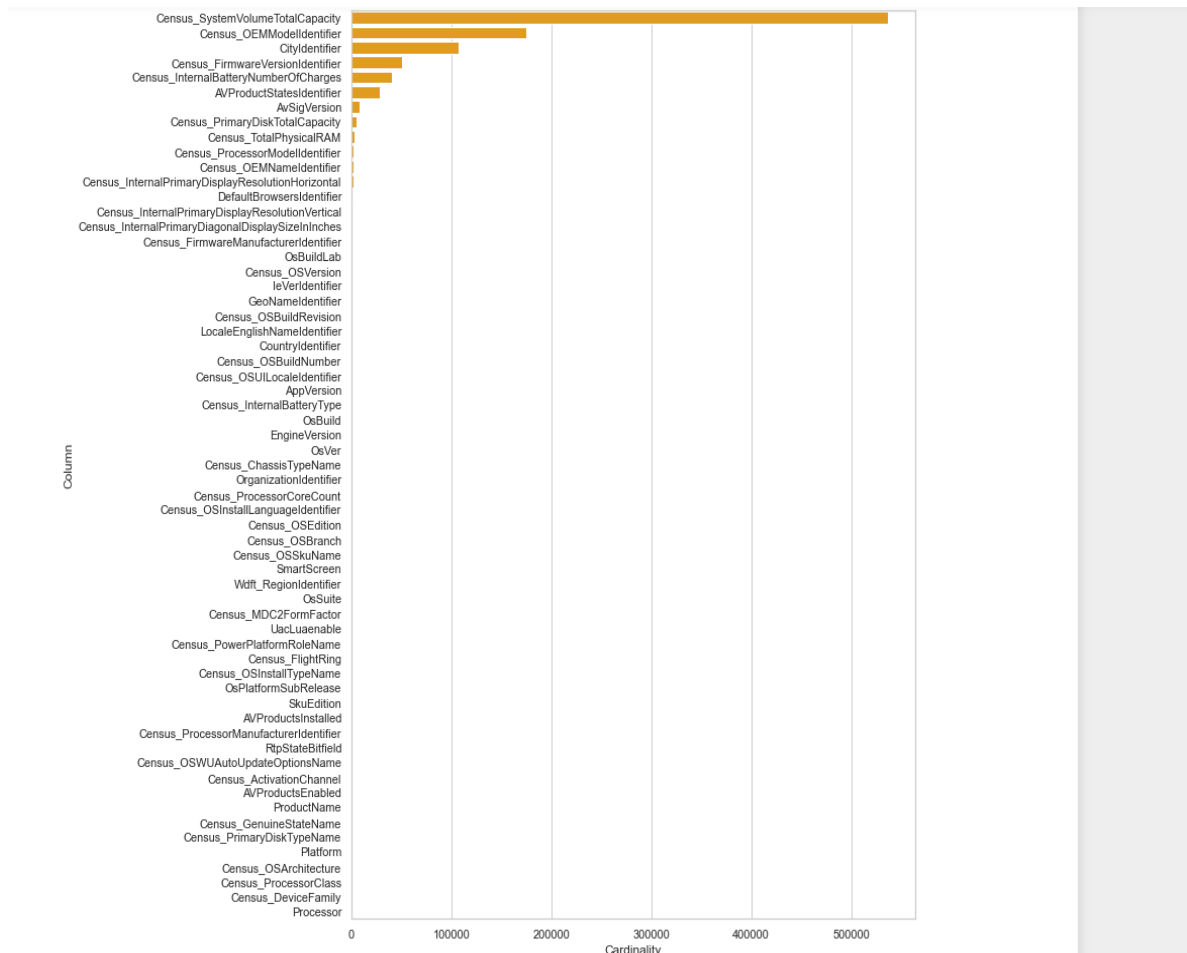
We don't have too many missing values for most of the features, but we do have seven features with more than 50% missing values, which probably would be a good idea to remove them.

2.5 CARDINALITY IN DATASET

The term "cardinality" refers to a high degree of variance in data entries. There are some columns with very high variation in their values. They provide some

unique values related to that example. As a result, it is ineffective in the creation of an efficient model.

Fig 2.4: Cardinality plot



CHAPTER 3

DATA PREPROCESSING

Data in machine learning is referred to as the examples or cases from the domain that specifies the given information and used to solve the given problem. Tabular data consists of rows and columns. Columns specify the characteristic or features of the dataset given. While rows consist of different examples or cases information according to the respective column. Every combination of row and column defines or guides us to a new datapoint. To access the datapoint we need to specify the row number and column number. In supervised learning, data consists of the domain information along with the correct output information that we need to predict. So after learning from the input dataset the machine learning model becomes capable of predicting values that are not provided also.

Row: A single case or information regarding a domain

Column: A property of the example represented in that row

Dataset is the collection of data objects. Data objects can be called as points, records, vectors, patterns, entities or cases. These data objects define the data information. Data objects describe the feature information of a dataset. Features are the characteristics of the source. Features are also called variables or dimensions.

The dataset provided by Microsoft is very large and divided into various features. These features consist of various characteristics of different machines. Each row provides some information about various features of that machine. HasDetections is considered as the true values given by the data provider. There are many missing values also that are present in the dataset. Using this information we need to predict how much the test machine is vulnerable to a malware hit. The detailed information about the dataset is given earlier.

3.1 What is Data Preprocessing?

The data that is collected from the domain or data provider is considered as **raw data**. The raw data is the data that is not processed in any way. It is the data that we get from the source. It is the most important thing to process this data. There are three main reasons why we have to preprocess the data.

- 1) Numbers are expected in machine learning algorithm and also it has some requirements
- 2) To optimize the data so as to minimize runtime.
- 3) The quality of data and useful information derived from it improves the model performance

For that, here are the data cleaning and preprocessing steps that we followed.

- Dividing features into binary columns, true numerical columns and categorical columns
- Using reduce memory function
- Columns with too many missing values are deleted
- Columns with high cardinality are deleted
- Filling missing values
- Feature engineering
- One hot encoding

Now lets see each data cleaning and preprocessing step in detail.

3.1.1 Splitting features into binary, numeric and categorical

There are 83 different features present in the dataset including unique machine identifiers and the output column as HasDetection. Each feature contributes towards the information. There are columns with numeric values and character strings also. And some are having combined data i.e. both numeric and string values. So to increase the model efficiency it is more important to split the columns into various categories.

Binary columns:

The columns having only 2 types of values are categorised as binary columns. So there are only two types of unique values that exist in that column information.

```
['IsBeta', 'IsSxsPassiveMode', 'HasTpm', 'IsProtected', 'AutoSampleOptIn', 'PuaMode', 'SMode', 'Firewall', 'Census_HasOpticalDiskDrive', 'Census_IsPortableOperatingSystem', 'Census_IsFlightingInternal', 'Census_IsFlightsDisabled', 'Census_ThresholdOptIn', 'Census_IsSecureBootEnabled', 'Census_IsWIMBootEnabled', 'Census_IsVirtualDevice', 'Census_IsTouchEnabled', 'Census_IsPenCapable', 'Census_IsAlwaysOnAlwaysConnectedCapable', 'Wdft_IsGamer', 'HasDetections']
```

Fig 3.1: Binary features

True numeric features:

The columns having only numerical entries are considered as numerical columns. The information that those columns are only integers or floating point numbers. But there are more than two unique values in them.

```
['Census_ProcessorCoreCount', 'Census_PrimaryDiskTotalCapacity', 'Census_SystemVolumeTotalCapacity', 'Census_TotalPhysicalRAM', 'Census_InternalPrimaryDiagonalDisplaySizeInches', 'Census_InternalPrimaryDisplayResolutionHorizontal', 'Census_InternalPrimaryDisplayResolutionVertical', 'Census_InternalBatteryNumberOfCharges']
```

Fig 3.2: Numerical features

Categorical features:

Every categorical data attribute represents discrete values that fall into a particular finite set of categories or groups. Groups or labels are attributes or variables that are to be predicted by a model in the form of attributes or variables (popularly known as response variables). Discrete values may be represented using text, numeric values, or even unstructured data like images.

Fig 3.3:
Categorical
features

```
['MachineIdentifier', 'ProductName', 'EngineVersion', 'AppVersion', 'AvSigVersion', 'RtpStateBitfield', 'DefaultBrowsersIdentifier', 'AVProductStatesIdentifier', 'AVProductsInstalled', 'AVProductsEnabled', 'CountryIdentifier', 'CityIdentifier', 'OrganizationIdentifier', 'GeoNameIdentifier', 'LocaleEnglishNameIdentifier', 'Platform', 'Processor', 'OsVersion', 'OsBuild', 'OsSuite', 'OsPlatformSubRelease', 'OsBuildLab', 'SkuEdition', 'IeVerIdentifier', 'SmartScreen', 'UacLuaenable', 'Census_MDC2FormFactor', 'Census_DeviceFamily', 'Census_OEMNameIdentifier', 'Census_OEMModelIdentifier', 'Census_ProcessorManufacturerIdentifier', 'Census_ProcessorModelIdentifier', 'Census_ProcessorClass', 'Census_PrimaryDiskTypeName', 'Census_ChassisTypeName', 'Census_PowerPlatformRoleName', 'Census_InternalBatteryType', 'Census_OSVersion', 'Census_OSArchitecture', 'Census_OSBranch', 'Census_OSBuildNumber', 'Census_OSBuildRevision', 'Census_OSEdition', 'Census_OSSkuName', 'Census_OSInstallTypeName', 'Census_OSInstallLanguageIdentifier', 'Census_OSUILocaleIdentifier', 'Census_OSWUAUUpdateOptionsName', 'Census_GenuineStateName', 'Census_ActivationChannel', 'Census_FlightRing', 'Census_FirmwareManufacturerIdentifier', 'Census_FirmwareVersionIdentifier', 'Wdft_RegionIdentifier']
```


3.1.2 Using reduce memory function

Reduce memory function is a function that is used to assign data types for each column according to the highest entry present in that column. The columns of our dataset are divided into numerical and categorical columns explained earlier in fig 2.2. This function helped a lot in cutting down the memory usage. For example, the columns IsBeta, HasTpm are binary columns consisting only of 0s and 1s. So for this minimum data type that can be used is int8. So like that this function helps in reducing memory.

The function uses the following approach.

1. Iterate through each column.
2. Check to see if the column is numeric.
3. Determine if an integer can be used to represent the column.
4. Determine the minimum and maximum values.
5. Choose the smallest data type that will match the set of values and use it.

After using this function on the dataset there is a massive cut down in memory usage. The statistics are shown below.

```
train
Memory usage of dataframe is 4764.59 MB
Memory usage after optimization is: 1475.04 MB
Decreased by 69.0%
CPU times: user 4min 11s, sys: 30.8 s, total: 4min 42s
Wall time: 3min 9s

test
Memory usage of dataframe is 4194.09 MB
Memory usage after optimization is: 1306.89 MB
Decreased by 68.8%
CPU times: user 3min 46s, sys: 23.9 s, total: 4min 10s
Wall time: 2min 48s
```

Fig 3.4: Reduce memory function

3.1.3 Deleting columns with too many missing values

When the percent of missing data is low, it's the most useful. If the percentage of missing data is too high, the findings will be devoid of natural variance, which will make it difficult to build an efficient model. The other choice is to delete information. To minimise bias when dealing with data that is missing at random, similar data may be removed. The table 2.1, shows the missing values for every column. So we can see there are 7 columns that are having missing values greater than 60%. So these columns contribute very little towards the information to train the model. We can remove them from our data set.

```
Columns with more than 60% null values: ['PuaMode', 'Census_ProcessorClass', 'Census_InternalBatteryType', 'Census_IsFlightingInternal', 'Census_ThresholdOptIn', 'Census_IsWIMBootEnabled']
```

Fig 3.5: Column more than 60% null values

3.1.4 Removing columns with high cardinality

Cardinality refers to high variation of data entries. There are columns with very high variation in their values see fig 2.4. They provide some unique values related to that example. So it doesn't help much in building an efficient model. So we can delete these columns in order to get better performance.

```
Columns with high cardinality: ['AvSigVersion', 'DefaultBrowsersIdentifier', 'AVProductStatesIdentifier', 'CityIdentifier', 'OsBuildLab', 'Census_OEMNameIdentifier', 'Census_OEMModelIdentifier', 'Census_ProcessorModelIdentifier', 'Census_FirmwareManufacturerIdentifier', 'Census_FirmwareVersionIdentifier']
```

Fig 3.6: Columns with high cardinality

3.1.5 Filling missing values

There are several columns with some missing values in them. In python, missing values are represented as NaN values. So these columns are filled with

different values according to the values that are present in that column. For example, if all the values present in a column are all positive then missing values can be filled with -1. Like that we can differentiate between the values that are not given.

List of columns with -1 value filled:

```
['Census_ProcessorCoreCount', 'Census_PrimaryDiskTotalCapacity',  
'Census_SystemVolumeTotalCapacity', 'Census_TotalPhysicalRAM',  
'Census_InternalPrimaryDiagonalDisplaySizeInInches',  
'Census_InternalPrimaryDisplayResolutionHorizontal',  
'Census_InternalPrimaryDisplayResolutionVertical',  
'Census_InternalBatteryNumberOfCharges', 'RtpStateBitfield',  
'AVProductsInstalled', 'AVProductsEnabled', 'CountryIdentifier',  
'OrganizationIdentifier', 'GeoNameIdentifier', 'LocaleEnglishNameIdentifier',  
'OsBuild', 'OsSuite', 'IeVerIdentifier', 'UnaLienable',  
'Census_ProcessorManufacturerIdentifier', 'Census_OSBuildNumber',  
'Census_OSBuildRevision', 'Census_OSInstallLanguageIdentifier',  
'Census_OSUILocaleIdentifier', 'Wdft_RegionIdentifier']
```

List of columns with 0 value filled:

```
['IsBeta', 'IsSxsPassiveMode', 'AutoSampleOptIn', 'SMode',  
'Census_HasOpticalDiskDrive', 'Census_IsPortableOperatingSystem',  
'Census_IsFlightsDisabled', 'Census_IsSecureBootEnabled',  
'Census_IsVirtualDevice', 'Census_IsTouchEnabled', 'Census_IsPenCapable',  
'Census_IsAlwaysOnAlwaysConnectedCapable', 'Wdft_IsGamer',  
'HasDetections']
```

List of columns with 1 value filled:

```
['HasTpm', 'IsProtected', 'Firewall']
```

3.1.6 Feature engineering

Not all features are created equal. It all comes down to picking a small subset of features from a huge pool. The attributes that better illustrate the relationship between an independent variable and the target variable are

chosen. Certain features are more critical than others when it comes to the model's accuracy. It differs from dimensionality reduction in that dimensionality reduction uses existing attributes to combine them, while feature selection uses certain characteristics to add or remove them.

In our dataset there are various features that can be added to a single feature to reduce data redundancy. We can have the information of two or more features combined so that the resultant feature provides the information present in individual features.

There are following steps that are taken for feature engineering

- total disk capacity remaining
- $\text{primary_drive_c_ratio} = \frac{\text{Census_SystemVolumeTotalCapacity}}{\text{Census_PrimaryDiskTotalCapacity}}$
- $\text{non_primary_drive_MB} = \text{Census_PrimaryDiskTotalCapacity} - \text{Census_SystemVolumeTotalCapacity}$
- $\text{aspect_ratio} = \frac{\text{Census_InternalPrimaryDisplayResolutionHorizontal}}{\text{Census_InternalPrimaryDisplayResolutionVertical}}$
- $\text{monitor_dms} = \frac{\text{Census_InternalPrimaryDisplayResolutionHorizontal} * \text{Census_InternalPrimaryDisplayResolutionVertical}}{\text{Census_InternalPrimaryDiagonalDisplaySizeInInches}^{**2}} / (\text{aspect_ratio}^{**2} + 1)$
- $\text{ram_per_processor} = \frac{\text{Census_TotalPhysicalRAM}}{\text{Census_ProcessorCoreCount}}$
- $\text{ProcessorCoreCount_DisplaySizeInInches} = \frac{\text{Census_ProcessorCoreCount} * \text{Census_InternalPrimaryDiagonalDisplaySizeInInches}}{\text{Census_InternalPrimaryDiagonalDisplaySizeInInches}}$
- $\text{SmartScreen_AVProductsInstalled} = \text{SmartScreen} + \text{AVProductsInstalled}$

So by combining these columns new features are created and the remaining features are hence dropped.

Dropped features:

```
['Census_PrimaryDiskTotalCapacity','Census_SystemVolumeTotalCapacity',  
'Census_TotalPhysicalRAM','Census_ProcessorCoreCount',  
'Census_InternalPrimaryDisplayResolutionHorizontal',  
'Census_InternalPrimaryDisplayResolutionVertical',  
'Census_InternalPrimaryDiagonalDisplaySizeInInches']
```

3.1.7 One hot encoding

One kind of hot encoding is the conversion of categorical variables into a format that can be fed into machine learning algorithms to improve prediction accuracy. Categorical data consists of variable having labels and these labels are often have limited number. Some algorithms can work directly with categorical data. But there are other algorithms that are unable to work directly on categorical data. They demand numeric variables for input and output. This implies that categorical data must be transformed into numerical data. The integer encoding is insufficient for categorical variables with no such ordinal relationship. In such case, integers can be encoded with one-hot encoding. For each unique integer value, the encoded values is replaced by a new binary value.

Not all machine learning algorithms are capable of handling all types of features (explicitly the categorical features). As a result, features are encoded to ensure that processing is smooth. Both training and testing sets are encoded.

1. Label encoding: Label Encoding converts values into numbers between 0 and $n-1$, where n denotes the number of different labels.
2. Frequency encoding: The frequency of feature values is used to encode them. Numbers between 0 and m are transformed values, where m is the number of values with a frequency greater than or equal to 2.

3.2 Resultant data after cleaning and preprocessing:

After performing all the data cleaning and preprocessing steps the final train and test data specification are as follows.

For training dataset: 8921483 entries

dtypes:	columns(69 total)
category	29
float16	17
float32	4
float64	2
int16	5
int32	1
int8	11

Table 3.1: Resultant training data

memory usage: 3.87 GB

For testing dataset : 7853253 entries

dtypes:	columns(69 total)
category	29
float16	18
float32	5
float64	1
int16	5
int32	1
int8	10

Table 3.2: Resultant testing data

memory usage: 3.33 GB

3.3 Size and Runtime:

Initially the training dataset is of 4.08 gb and the testing dataset is 3.54 gb. After applying feature engineering and preprocessing on the dataset ,the training dataset is reduced to 3.87 gb and the testing dataset is reduced to 3.33 gb. In feature engineering and data preprocessing we not only deleted the feature which is not useful but also added missing values and some features , so that we get some insights from the dataset.

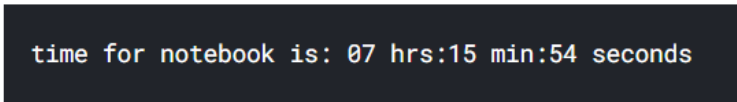
	Before preprocessing	After preprocessing
Training dataset	4.08 GB	3.87 GB
Testing dataset	3.54 GB	3.33 GB
Total	7.62 GB	7.2 GB

Table 3.3: Total memory utilization

Total size=Training dataset+Testing dataset = 7.2 GB

A program's runtime is the amount of time it spends running. It starts when a programme is started (or executed) and ends when the programme is completed. We used Kaggle server for data preprocessing as it provides 16GB RAM with 4 core processors and also it is easy to explore and create models. Kaggle's servers have a massive amount of computing capacity. By visiting their website, you will have access to cutting-edge machine learning and data analysis software that has been pre-installed and tested for compatibility.

The total runtime for the preprocessing is shown below:



```
time for notebook is: 07 hrs:15 min:54 seconds
```

Fig 3.7: Total runtime for data processing notebook

CHAPTER 4

MACHINE LEARNING TECHNIQUES & PYTHON LIBRARIES

4.1 Light GBM

Light GBM algorithm

1. It is a machine learning algorithm that uses a tree-based gradient boosting framework.
2. It is efficient, better accuracy, high speed training, and lower memory usage.
3. Large scale data can be handled in this algorithm.
4. It supports GPU, distributed, and parallel learning.

This algorithm uses two novel techniques :

- GOSS
- EFB Technique

Above techniques combined together attain the histogram-based algorithm which was being used in most of the decision tree algorithms before which makes the algorithm work efficiently.

4.1.1 GOSS:

We know that at various stages of a particular code have different parts in the reckoning of information gain. Larger the gradient of a trained model, the more it contributes towards information gain. This technique keeps all these large gradient occurrences that are larger than a threshold that is predefined and it drops those small gradient occurrences so that the accuracy of data output estimation is maintained. When the information data is large , Gradient algorithms can give rise to a better estimation of gain.

4.1.2 Exclusive Feature Bundling Technique for LightGBM:

Due to the fact that high-dimensional data is typically sparse, we may build a close loss-less loom to minimize most of the quality features. Many features are

mutually exclusive, meaning non zero values can't be taken at same time. The unique features can be securely combined into one feature (called an Exclusive Feature Bundle).

4.1.3 Architecture of Light GBM Model:

Unlike other boosting algorithms that expand trees level-by-level, Light-GBM breaks the tree leaf-by-leaf. It grows the leaf with the greatest delta loss. The leaf algorithm loss is less than the level algorithm since the leaf is set. In small datasets, growth in leaf algorithms can enhance the model's complication and give rise to overfitting.

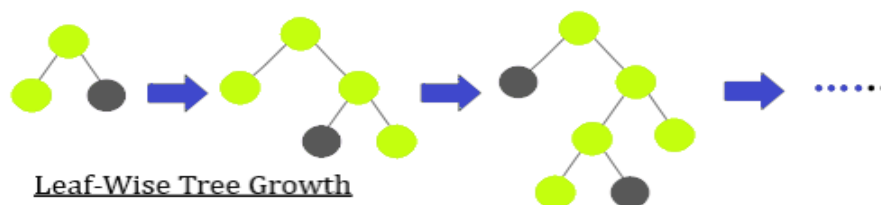


Fig 4.1: Leaf-Wise Tree Growth

4.1.4 Parameter Tuning:

- Categorical feature: This parameter determines the categorical feature that will be used in the training model.
- The number of iterations to be done is defined by num iterations. 100 is the default rating.
- The number of leaves in a tree is specified by num leaves. It should be less than max depth squared.
- min data in the bin determines the smallest volume of space that can be stored in a single bin.
- Max depth: It limits the width of the forest. 20 is the default value. It works well for managing to overfit.
- It determines the mission we want to complete, which is either train or prediction.

- It determines the fraction of functionality that should be thought in every iteration. One is the default value.
- Bagging Function: It determines the fraction of data that will be used for each iteration.

4.2 Xgboost

Gradient Boosted and decision trees are introduced in XGBoost. C++ has been used to construct this library. It is a set of software libraries that was developed with the purpose of improving model speed and performance. In recent times, it has prevailed in the area of applied machine learning. Many Kaggle competitions are driven by XGBoost design. Decision trees are produced concurrently in this algorithm. In XGBoost, weights are quite relevant. Every one of the independent variables is assigned weights, which are given to the decision tree, which speculate outcomes. The weight of variables that the decision tree estimated incorrectly get raised, and the variables are given into the next decision tree. After that, the separate classifiers are merged to have a more efficient reliable model. It could be used to resolve issues that include user-defined regression, classification, grading, and prediction. There are little frills in the library since it is laser-focused on computing speed and model efficiency.

4.2.1 Model Features

Gradient boosting is provided in three different ways:

- Gradient Boosting
- Stochastic Gradient Boosting
- Regularized Gradient Boosting

4.2.2 System Features

This library contains the features to be used in a variety of programming environments: Tree raising parallelization, distributed computing for training very big models, algorithm of cache optimization and configuration of data.

4.3 Decision Tree Classifier

Decision Trees are a non-parametric algorithm. It is a supervised learning process used for classification and regression. The model aims to learn some basic judgement rules by training the data and then based on that it tries to forecast the output value. It work on divide and conquer algorithm, that is, it breaks down the dataset into smaller subsets and tries to develop a decision tree. Decision trees algorithm can handle both categorical and numerical variables.

4.4 Gradient Boosting classifier

Gradient Boosting constructs an additive model in a stage-by-stage manner, allowing for the optimization of every differential loss function. The negative gradient of the binomial or multinomial deviance loss function is used to match n classes_ regression trees in each point. A special case of binary classification is where only one regression tree is induced.

4.5 Logistic Regression

It is used to determine the probability of a categorical dependent variable. This variable is binary variable which is given as 1 (yes, performance, etc.) or 0 (no) (no, failure, etc.). Meaning, as a function of X , this model predicts $P(Y=1)$.

Logistic regression is the go-to linear classification algorithm. It's easy to use, learn, and get excellent results on a wide range of challenges, even though the method's standards of data aren't fulfilled.

4.6 Adaboost Classifier

It starts by fixing classifier in given dataset, which then fixes duplication of classifier in the given dataset only. It adjusts weights of wrongly categorized occurrence which concentrate largely on troublesome situations.

4.7 Support Vector Machine (SVM)

It is a supervised and linear Machine Learning algorithm that is most widely used for solving classification problems. The aim of SVM is to find the best line that divides the two types of data points. SVM produces a line that neatly divides the two groups. You might wonder how clean it is. There are several different ways

to draw a line that divides the two groups, but in SVM, the margins and support vectors decide it. As seen in the illustration above, the margin is the space between the two dashed green lines. The greater the margin, the higher the separation between groups. The data points that each of the green lines goes along are the support vectors. These points are known as support vectors because they help to define the margins and hence the classifier. These help vectors are actually the data points that are nearest to the boundary of any of the groups that have a chance of belonging to one of them. The SVM then creates a hyperplane with the highest margin, which in this situation is the black bold line that divides the two groups and is at the optimal distance between them. The line is replaced by a hyperplane that divides multidimensional spaces where there are more than two features and multiple dimensions.

4.8 K-Neighbors Classifier

K nearest neighbors is a straightforward algorithm that stores all possible cases and categorizes new ones using a similarity metric (e.g., distance functions). KNN has been used in pattern analysis and mathematical estimation. On top of Python, the sklearn library provides an abstraction layer. It is therefore necessary to construct an instance of `KNeighborsClassifier` in order to use the KNN algorithm. The `KNeighborsClassifier` searches for the 5 closest neighbors by default. To determine the distance between adjacent points, we must specifically instruct the classifier to use Euclidean distance.

4.9 Multi-Layer Perceptron Classifier

The classification task is performed by `MLPClassifier` using an underlying Neural Network. Backpropagation is a supervised learning method used by MLP for teaching. MLP is distinguished from a linear perceptron by its many layers and non-linear activation. It can tell the difference between data that isn't linearly separable.

4.10 Random Forest Classifier

A random forest is a meta estimator that uses averaging to increase predictive precision and control over-fitting by fitting a range of decision tree classifiers on different sub-samples of the dataset. For getting an accurate and attested

prediction, it makes and merges different decision trees. When increasing the leaves, the random forest brings more randomness to the model.

4.11 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA), also known as Normal Discriminant Analysis (NDA), or Discriminant Function Analysis (DFA), is a dimensionality reduction methodology widely utilized for supervised classification problems. It's used to represent group distinctions, such as dividing two or three classes. LDA would act as a classifier, then minimize the dataset's dimensionality using a neural network to perform the classification task; the outcomes of both methods will then be compared.

4.12 Python Libraries

There are very vast libraries in python languages. Some of the important libraries which we used in our complete dataset and model are listed below.

4.12.1 Import numpy as np

NumPy is a free and open-source Python library for numerical computations. A multi-dimensional sequence and matrix data structures are used in NumPy. It can perform a variety of mathematical operations on arrays, including trigonometric, statistical, and algebraic routines. NumPy is a Numeric and Numarray extension.

4.12.2 Import pandas as pd

Import pandas as pd is a popular Python-based data analysis toolkit that can be imported. It includes a variety of tools, from interpreting various file formats to transforming an entire data table into a NumPy matrix list. As a result, pandas is a reliable partner in data science and machine learning. 1-D arrays are referred to as a series. 2-D arrays are referred to as Dataframe.

4.12.3 From tqdm import tqdm

TQDM is a status bar library that works well for Jupyter's Python notebooks and nested loops. Tdqm employs keen algorithms to estimate residual time and avoid excessive iteration displays, resulting in minimal workload in most situations.

4.12.4 Import `dask.dataframe` as `dd`

A Dask DataFrame is a huge parallel DataFrame that is divided along the index into several smaller Pandas DataFrames. Pandas DataFrames can be stored on disc for out-of-memory processing on a single computer or through a cluster of computers. Many Pandas DataFrame operations are triggered by a single Dask DataFrame operation. When Pandas fails due to data size or computing speed, Dask DataFrame is used. Manipulation of huge datasets, even though they are too large to fit in memory. Using several cores to speed up long computations. Normal Pandas operations like groupby, join, and time series computations are used to perform distributed computing on massive datasets.

4.12.5 Import `seaborn` as `sns`

It is a library in python for making mathematical graphics. matplotlib is the keyword on which it is established on and tightly interacts with pandas data structures. It assists you in exploring and comprehending your results. Its plotting functions work for dataframes and arrays comprising whole databases, doing the required semantic mapping and statistical aggregation internally to generate detailed plots.

4.12.6 Import `matplotlib.pyplot` as `plt`

It is a library in python used to plot data in given dataset. This pyplot has a various function and it works mostly as a plot function which works in MATLAB. Every pyplot function alter diagram in a way thus making a new figure in a plot area and making lines in that area, enhancing the plot with labels, etc.

In this chapter, we learned about various machine learning algorithms, what they are, how they are used in models and the basic concept behind those algorithms. Studying these algorithms was very useful as after knowing this, we know how to apply these algorithms to our dataset. We also learned about different python libraries which came in very handy during writing code, for studying output and implementing the code.

CHAPTER 5

RESULTS of Light GBM Model

Due to the challenges of categorical features of high cardinality and insufficient hardware resources (CPU, RAM, GPU). A tree-based ensemble architecture, namely LightGBM, which was open-sourced by Microsoft in 2017, is the best ML model capable of handling missing values, handling categorical features with limited memory use, producing a model easily, and having high efficiency. LightGBM uses tree based learning algorithms and is a gradient boosting framework. It is intended to be distributed and effective, and it offers the following benefits: Increased training quality and pace. Reduce our memory use. Better precision. It is also implemented by different Kagglers for their final submission. So we have decided to work on it and implement this model for our dataset.

In this chapter, we are going to discuss the detailed analysis of how the Light GBM model is tuned and how it is implemented.

5.1 Model Selection and Data Building

LGBMClassifier is based on the scikit-learn framework. That's why we went ahead and used the classifier as our model right away. However, if we use the original lgbm, we may need to first construct LGB.datasets before passing it, which is a lot of work.

The input dataset used for training and testing is the data that is processed earlier. See 3.3. The size of the training dataset is 3.87 GB and the size of the testing dataset is 3.33 GB. So the total size of the input dataset becomes 7.2GB, which is a huge dataset to process on a computer having 16 RAM also. As we were not having the computers with the ram and processor required for this data. Therefore, we have decided to use online platforms like kaggle and google colab. Google colab provides 12 GB RAM for processing and Kaggle notebook gives 16 GB RAM. So we finally shifted on Kaggle notebooks for model training.

5.2 Parameter tuning

Hyperparameters are responsible for the model performance. They influence the model training process. So it is necessary to tune the hyperparameters prior to training of the model. For that task, Bayesian Hyperparameter Optimization was used with the hyperopt-library. A three-fold Cross Validation was used to return the best-fit model in this process.

Bayesian Hyperparameter Optimization can be stated as it creates an objective function probability model and uses it to choose the most promising hyperparameters to test in the real objective function. So by using the library function and also taking some hyperparameters from other kagglers who have already implemented the model we got the following parameters best fitting the model. Still there is a lot of work that can be done to optimize the parameters further.

```
[20]: best_hyp = {'boosting_type': 'gbdt',  
              'class_weight': None,  
              'colsample_bytree': 0.6110437067662637,  
              'learning_rate': 0.0106,  
              'min_child_samples': 295,  
              'num_leaves': 160,  
              'reg_alpha': 0.6321152748961743,  
              'reg_lambda': 0.6313659622714517,  
              'subsample_for_bin': 80000,  
              'subsample': 0.8202307264855064}
```

```
%%time  
# Re-create the best model and train on full training data  
model = lgb.LGBMClassifier(n_estimators=estimators, n_jobs = -1,  
                           objective = 'binary', random_state = 50, **best_hyp)
```

```
CPU times: user 36 µs, sys: 7 µs, total: 43 µs  
Wall time: 46.3 µs
```

Fig 5.1: Hyperparameters

5.3 Training

After trying many iterations on hyperparameter tuning. Finally the training dataset is applied to the tuned model for the training process. The variable 'model' contains the properly tuned Light GBM classifier see fig 5.1. As our dataset is considered to be large so it took a little longer time. We tried training this model on a Kaggle notebook with 16 GB RAM and 4 core processor. After facing so many errors we finally were able to train the model.

5.4 Model parameters

After training the model the parameters of the trained model need to be tested. So according to section 5.1, we have a validation set ready for testing it. This set is given input to the trained model and predictions were calculated. After predicting the output on validation set. The following parameters were studied.

```
In [23]: %%time
          model.fit(X_train, y_train)

CPU times: user 8h 35min 44s, sys: 44.2 s, total: 8h 36min 28s
Wall time: 2h 11min 21s

Out[23]: LGBMClassifier(colsample_bytree=0.6110437067662637, learning_rate=0.0106,
                        min_child_samples=295, n_estimators=12000, num_leaves=160,
                        objective='binary', random_state=50,
                        reg_alpha=0.6321152748961743, reg_lambda=0.6313659622714517,
                        subsample=0.8202307264855064, subsample_for_bin=80000)
```

Fig 5.2: Model training

5.4.1 Cross Validation Score

The classification report is generated using the library function. See figure 5.3, this report shows the precision, recall and f1 score along with the confusion matrix. All the three values are nearly the same so we can conclude from that the model that is trained is not biased. The number of false positives and the number of false negatives is nearly the same. So the model is somewhat equally likely to the predictions of 1s and 0s. The accuracy comes out to be 67% which has a scope of improvisation further by fine parameter tuning.

Table 5.1: Cross validation report

	precision	recall	f1-score
0	0.66	0.68	0.67
1	0.67	0.65	0.66
accuracy			0.67
Macro avg	0.67	0.67	0.67
Weighted avg	0.67	0.67	0.67

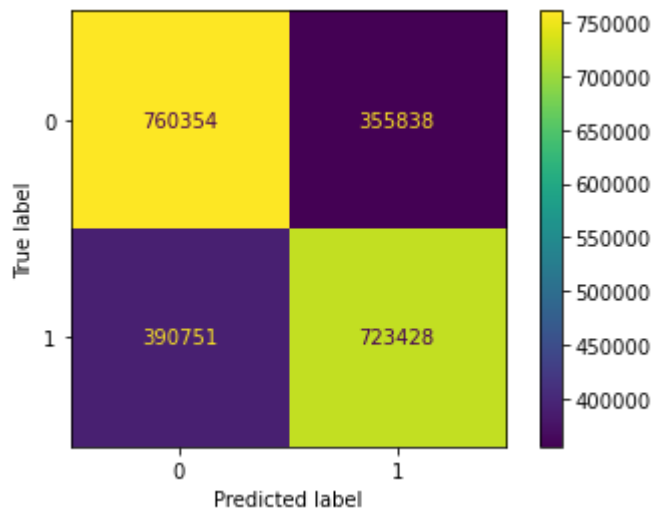


Fig 5.3: Confusion matrix

Validation sklearn mean auc score: 0.6652

5.4.2 AUC score

From the results of prediction output of the model the mean auc score comes out to be **0.6652**, which is considered as a good score. As the dataset provided is very large and there are a lot of variations in the data. So the score above 0.5 is considered to be good and we were able to achieve the score that is a little better than that also. See fig 5.4. A complete graph is shown with false positive rate positive rate and true positive rate as its axis. This graph shows the

ROC (Receiver Operating Characteristics) curve and the area under that curve is the auc score that we got, which is 0.6652.

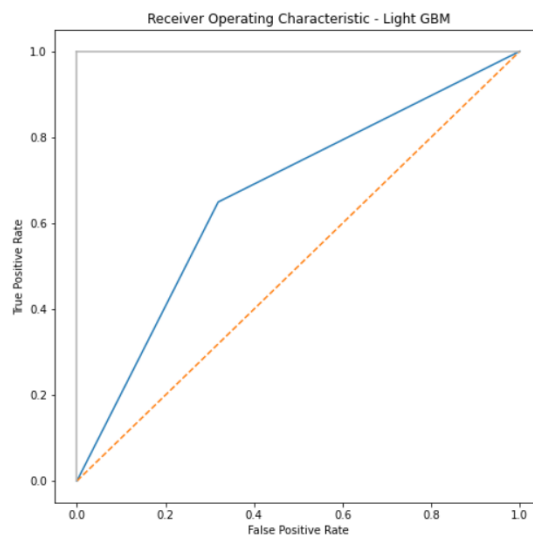


Fig 5.4: ROC curve

5.5 Prediction

Finally, after tuning the parameters of the model and testing it on the validation set. The Light GBM model with descent results is ready. So we have applied the testing data to the model and as the testing data was too big, many times we encountered memory errors. So then after finally debugging the errors the model started the prediction. But the runtime for that also took a little longer. Finally after completely processing the output that we got is shown in the plot 5.5.

The output that we got is the list of probability of 1s, i.e. machine is getting hit by a malware, which is a number between 0 and 1 for each test machine identifier. This number shows how much a machine is likely to get infected by a malware attack. The histogram shown in figure 5.6. Shows the count of probabilities with different slabs of values between 0 and 1. The plot shown in figure 5.5 also shows the same with each probability on its x-axis.

Finally, combining the testing machine identifiers and probability outcomes a single data frame is created for submission. This submission file is saved in the local computer.

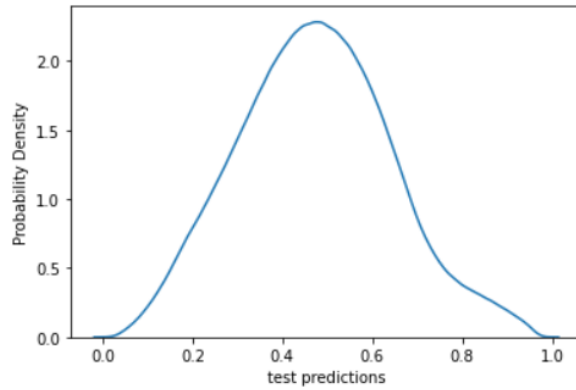


Fig 5.5: Probability density curve

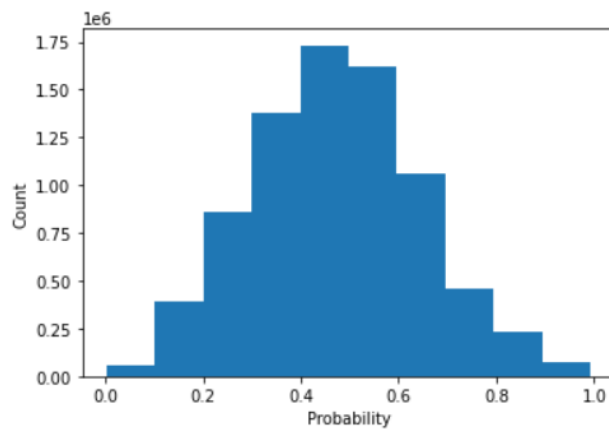


Fig 5.6. Probability distribution

5.6 Feature Importance

For further study and model improvisation, we have studied the feature importance of each feature. Feature importance is a measure of contribution of an individual feature for prediction of the output. So by using the library function we were able to get values that represent the feature importance. We have plotted the values see figure 5.7. This plot shows the feature importance of the Light GBM model, with feature importance on y-axis and its importance value on the x-axis.

From the values shown we can see that the feature 'non_primary_drive_MB' which is created by feature engineering is contributing the most for the predictions. Along with that there are several other features which have a huge influence on the results. On the other hand, features like 'IsBeta' and 'Census_Is_flightsDisabled' contribute less.

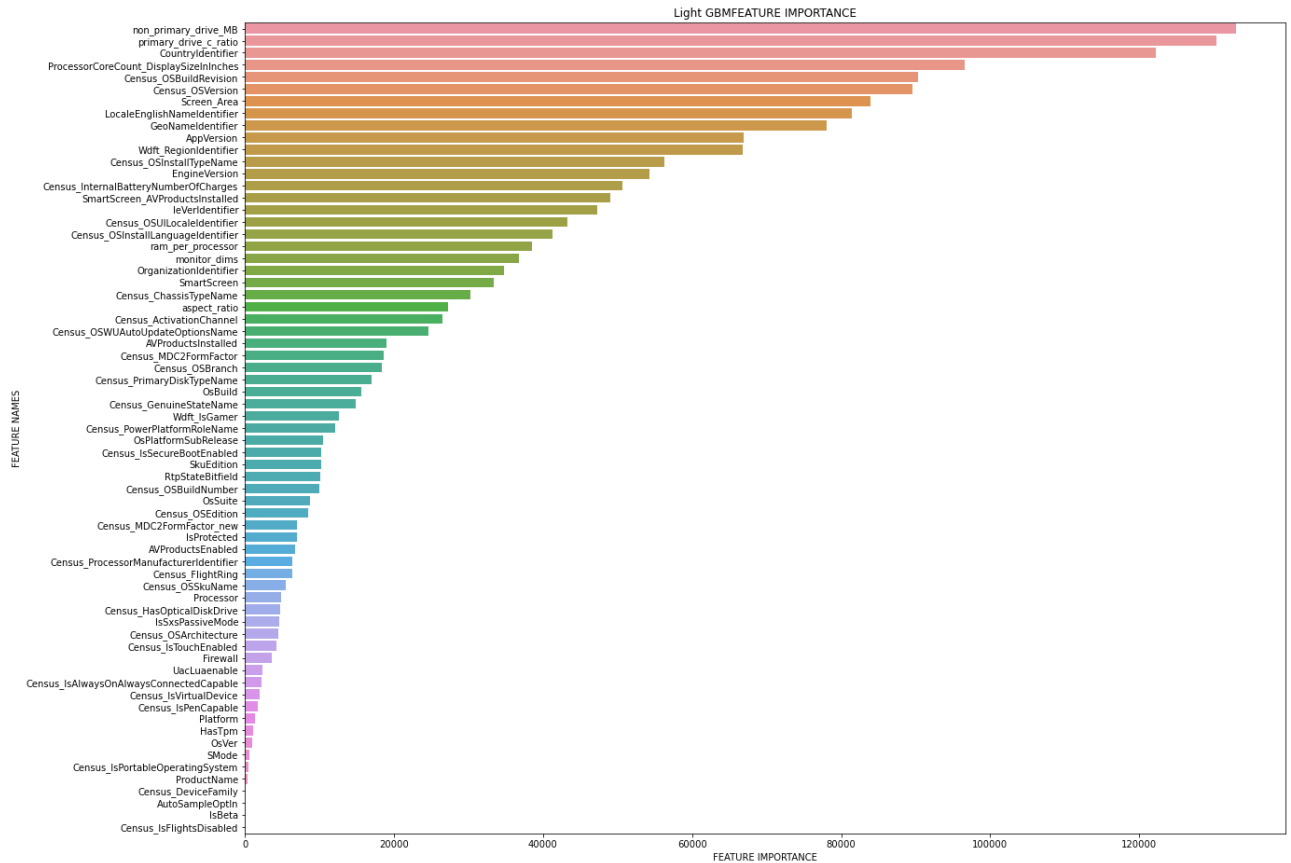


Fig 5.7: Feature importance

There are several other aspects to analyze the performance of a trained model. We have analyzed the above described features. The results that we got are properly stated in different different sections of this chapter. For further conclusion on the same refer to chapter 7.

CHAPTER 6

Results of model comparison

When we select different machine learning algorithms to carry out our training and testing, there is always a tradeoff. Machine learning algorithms are, without a doubt, always challenging. The key issues, or so-called "challenges". The memory consumption, the time it takes to run, and the accuracy it provides are all factors to consider.

There are a number of different machine learning algorithms that are used for various purposes. So it is necessary to compare the performance of these algorithms on the same parameters and same data. To find out which model performs better we need to compare them. So we have tried to analyze different algorithms based on various parameters. So as to decide which model provides better results on this dataset.

6.1 Selecting Data

In machine learning, data is one of the most important part. So for getting proper data, that is required for good machine learning prediction, data processing is most important.

In our dataset, there are nearly 9 million training machine identifiers and nearly 8 million test machine identifiers. So the dataset given consists of a billion data points. See Table 3.3 , The size of the training dataset is 3.87 GB and the size of the testing dataset is 3.33 GB. So the total size of the input dataset becomes 7.2GB, which is a huge dataset to process on a computer having 16 RAM also. Along with it training a single model with total data takes RUNTIME OF LIGHTGBM is 2Hr 20min 47sec. Therefore, we have decided to choose 100000 random rows from the training dataset. As we are just comparing the model performance and not predicting the output on testing data. We have not included the testing dataset in this notebook for processing.

For analysing the model performance, the input data is further divided into training set and testing set with a splitting factor of 0.25 by default.

```
In [19]: X_train,X_test,y_train,y_test = train_test_split(train,train_labels, random_state = 0)
```

Fig 6.1: Train-test split

```
train=pd.read_csv('../input/final-all-data-cleaning/finall_train_clean.csv',dtype=dtypes)
train=train.sample(n=100000, random_state=99)
print("train")
train = reduce_mem_usage(train)
```

```
train
Memory usage of dataframe is 342.70 MB
Memory usage after optimization is: 337.18 MB
Decreased by 1.6%
CPU times: user 1min 53s, sys: 6.27 s, total: 1min 59s
Wall time: 2min 26s
```

Fig 6.2: Training data preparation

That makes X_train and X_test with rows 75000 and 25000 rows respectively. These data frames along with correct labels are further used for comparison. This data is prepared for further processing and it is constant for all the models for training and testing.

6.2 Model preparation

The secret to a fair comparison of machine learning algorithms is to make sure that they are all tested on the same input data and same parameters. So we have only used the default machine learning algorithms that are present in sklearn libraries. The algorithms that are selected are based on different approaches towards data handling and data processing. Some are good with data handling and some are with low runtime. Every model has its own feature that is important in getting the desired output. So for our dataset, it is necessary to compare these algorithms on the same grounds. Following is the list of machine learning algorithms that we have selected for comparison.

- Light GBM
- Gradient Boosting
- Xgboost
- Decision Tree
- AdaBoost Classifier
- Logistic Regression
- Random Forest
- MLPClassifier- Neural Network
- KNeighborsClassifier
- SVM
- Linear Discriminant Analysis

6.3 Parameters for evaluation

The model's performance on unknown data will be determined by the final parameters discovered after training. Therefore, there are number of model parameters that can be analyzed to decide how the model is going to perform. Following is the list of parameters that are being analysed.

- Cross Validation Score
- AUC Score
- Confusion matrix
- Precision, recall and F1-score

6.3.1 Cross Validation Score

The statistical method of cross-validation is used to estimate the ability of machine learning models. It's widely used in applied machine learning to compare and choose a model for a given predictive modelling problem because it's simple to understand, simple to implement, and produces ability estimates with lower bias than other approaches. KFold divides all of the samples into folds, which are equal in size. Folds are used to learn the prediction function, and the fold that is left out is used to evaluate it. In our case as data itself is too large see table 6.1, we have taken $n=3$ i.e. 3 folds for cross validation.

The Accuracy of the model is the average of the accuracy of each fold. In figure 6.3 we can see different algorithms with their cross validation scores. Boxplots are a way to see how well a data set's data is distributed. The data set is divided into three quartiles. The graph depicts the data set's minimum, limit,

median, first quartile, and third quartile. Drawing boxplots for a data set allows us to compare the distribution of data across data sets. The orange line represents the average cross validation score for the given algorithm. After analysing the plot we can say that Light GBM, Gradient Boosting, AdaBoostClassifier, MLP Classifier are the models that are having average cross validation score greater than 0.6 and Decision Tree is having moderate score while algorithms like Logistic Regression, SVM, and KNeighborsClassifier are having less accuracy.

So analysing all we can say that the overall cross validation score for tree based algorithms is quite high as compared to other algorithms. Light GBM has the highest score of nearly 0.635. Neural Network also showing better cross validation results.

So these are the conclusions that we can draw by analysing cross validation scores.

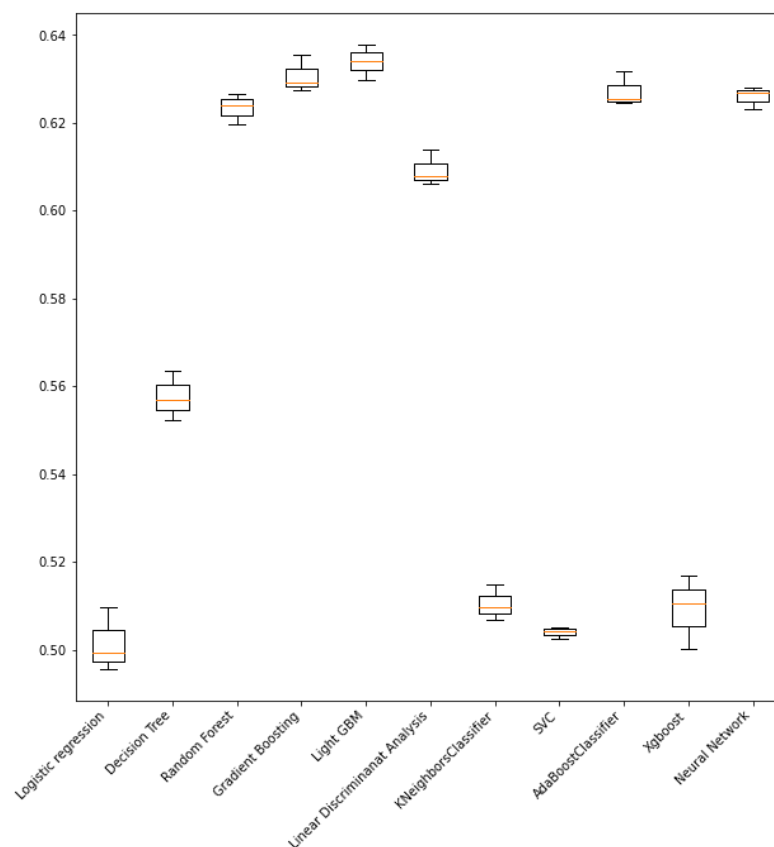


Fig 6.3: Cross validation score

6.3.2 AUC Score

To assess the precision of our classifiers, we used Receiver Operating Characteristic (ROC) scores and Area Under Curve (AUC). AUC measures how much a classifier can differentiate between groups and is represented by a probability curve called ROC. A classifier with a higher AUC is more likely to do well.

The true positive rate (TPR) is plotted against the false positive rate (FPR) on the ROC curve (FPR). To illustrate, the purple curve with ROC=1 (TPR=1 and FPR=0) is the optimal classifier, while the random classifier with ROC=0.5 is the worst.

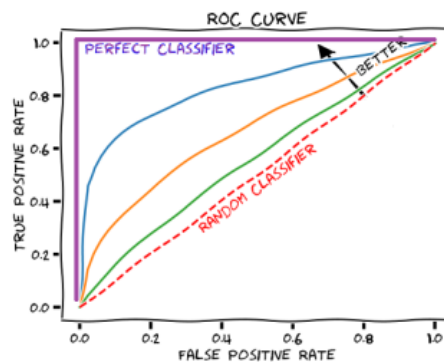


Fig 6.4 ROC curve study

We used a 75-25 split by default to divide our training data into training and validation sets, then checked the classifier trained on 75% of the data on the remaining 25%. The ROC AUC scores are then calculated using the prediction score.

Fig 6.5, ROC Curve Analysis that Light GBM has the greatest area under curve with value 0.6387. Along with that neural network is also showing great auc score. Likewise we can see auc scores of different algorithms.

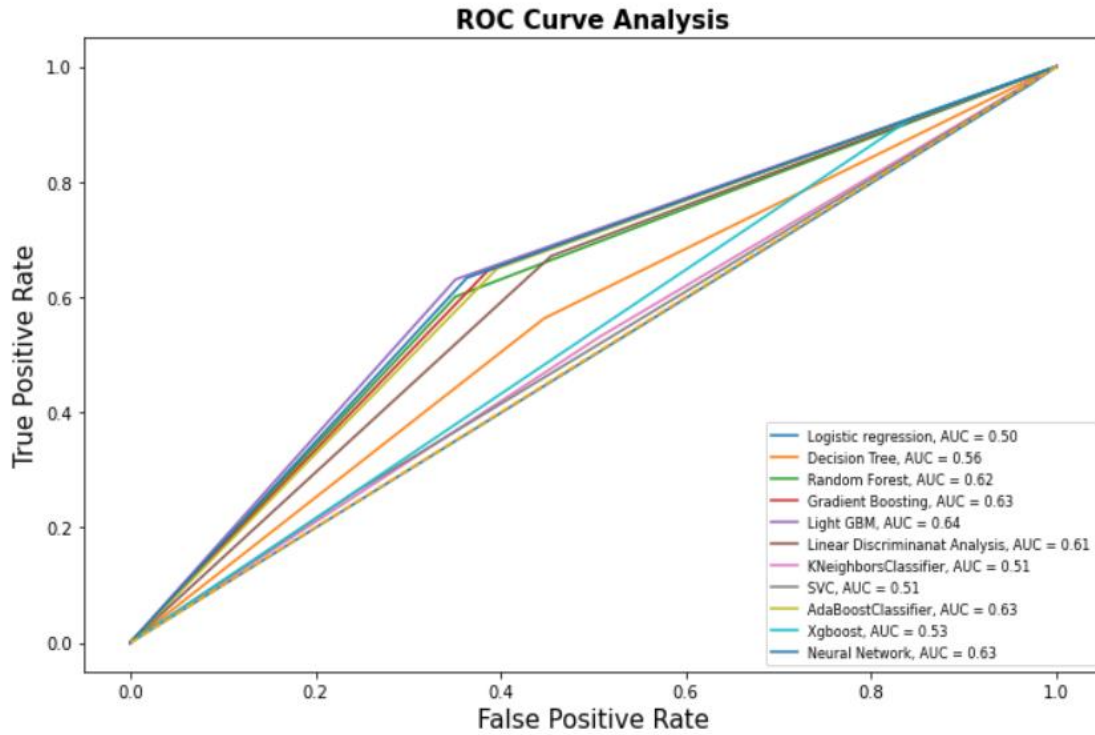


Fig 6.5: ROC Curve Analysis

Model	Accuracy	Precision	Recall	F1	AUC_Score
Logistic regression	0.49376	0.494187	0.968806	0.654510	0.498501
Decision Tree	0.55524	0.549624	0.561662	0.555578	0.555304
Random Forest	0.62756	0.627977	0.607322	0.617477	0.627358
Gradient Boosting	0.62932	0.621834	0.640779	0.631164	0.629434
Light GBM	0.63884	0.637034	0.628334	0.632654	0.638735
Linear Discriminant Analysis	0.60968	0.595488	0.659205	0.625729	0.610174
KNeighborsClassifier	0.51200	0.506739	0.528689	0.517481	0.512167
SVC	0.51160	0.512905	0.263375	0.348035	0.509123
AdaBoostClassifier	0.62492	0.616226	0.642072	0.628884	0.625091
Xgboost	0.53900	0.521624	0.827542	0.639900	0.541879
Neural Network	0.63388	0.628706	0.635769	0.632218	0.633899

Table 6.1: Comparison of confusion matrix parameters of Different Models

6.3.3 Confusion matrix

A confusion matrix, also known as an error matrix, is a basic table structure that enables visualisation of the output of an algorithm, usually a supervised learning one, in the field of machine learning and especially the problem of statistical classification. The Confusion Matrix is a tool for summarising a classifier's results. On a holdout test results, it includes details about the real and expected classification. Since there is no such term for the multiclass case, the concept can be interpreted using the binary classification problem. Using the one vs. rest technique, we will quantify the metrics for each class. For example: Let's take two classes of malware that are positive(1) and negative (0).

1 corresponds to Malware that will hit the windows system.

0 corresponds to Malware that will not hit the windows system.

True Positive(TP): It actually belongs when the output is positive and also predicted as positive.

True Negative(TN): It actually belongs when the output is negative and also predicted as negative.

False Positive(FP): It actually belongs when the output is negative but predicted as positive.

False Negative(FN): It belongs when the output is positive but predicted as negative.

True Positive Rate/Recall: TPR or Recall is the proportion of positive report which are correctly identified. It is given by the formula:

$$TPR = TP / (TP + FN)$$

Precision: It is the ratio of the number of positive correctly predicted as positive to the total number of positive predicted by the classifier.

$$Precision = TP / (TP + FP)$$

Low Precision means the higher number of false positives.

F1 score: F1-score generates a single score that accounts for both accuracy and recall issues in a single number. Neither accuracy nor recall can tell the entire story on their own. We may have excellent precision but poor recall, or vice versa, poor precision but good recall. With the F1-score, you can express all issues with a single score. Once precision and recall for a binary or multiclass classification problem have been determined, the two scores can be merged to determine the F1-score.

$$\text{F1-score} = (2 * \text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$$

Our model is predicting the probability of a machine getting hit by malware. So the model results should focus more towards the fact that it is accepted if the model is predicting 1 or true for a machine which is not actually more liable i.e. the false positive rate can be higher. It is necessary because, in our case, it is ok if the model is biased towards predicting higher false positive values. We can achieve this by reducing the threshold that is used to differentiate whether the machine is getting infected or not. So the model having lower precision will perform better in our case. So by analyzing the table, shown in table no 6.1 , we can conclude that the algorithms that have lower precision and higher recall will likely give better results for our dataset.

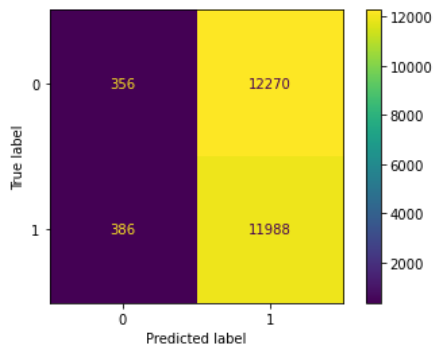
		Predicted	
		Positive	Negative
Observed	Positive	TP	FN
	Negative	FP	TN

Table 6.2: Confusion Matrix

Confusion Matrix Of Different Models :

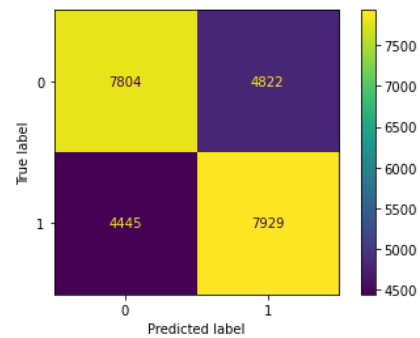
Logistic regression				
	precision	recall	f1-score	support
0	0.48	0.03	0.05	12626
1	0.49	0.97	0.65	12374
accuracy			0.49	25000
macro avg	0.49	0.50	0.35	25000
weighted avg	0.49	0.49	0.35	25000

<Figure size 504x504 with 0 Axes>



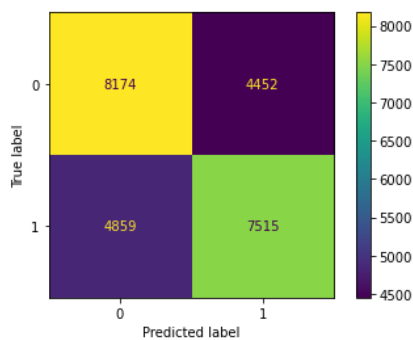
Gradient Boosting				
	precision	recall	f1-score	support
0	0.64	0.62	0.63	12626
1	0.62	0.64	0.63	12374
accuracy			0.63	25000
macro avg	0.63	0.63	0.63	25000
weighted avg	0.63	0.63	0.63	25000

<Figure size 504x504 with 0 Axes>



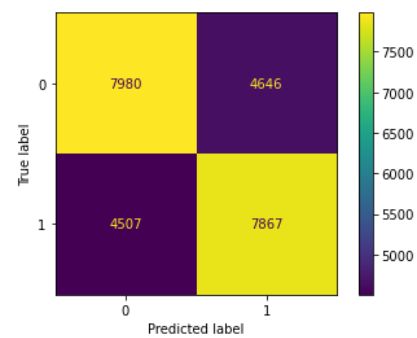
Random Forest				
	precision	recall	f1-score	support
0	0.63	0.65	0.64	12626
1	0.63	0.61	0.62	12374
accuracy			0.63	25000
macro avg	0.63	0.63	0.63	25000
weighted avg	0.63	0.63	0.63	25000

<Figure size 504x504 with 0 Axes>



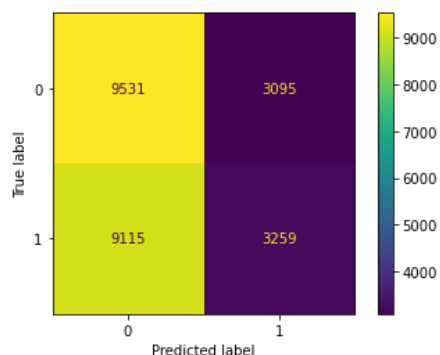
XGBoostClassifier				
	precision	recall	f1-score	support
0	0.64	0.63	0.64	12626
1	0.63	0.64	0.63	12374
accuracy			0.63	25000
macro avg	0.63	0.63	0.63	25000
weighted avg	0.63	0.63	0.63	25000

<Figure size 504x504 with 0 Axes>



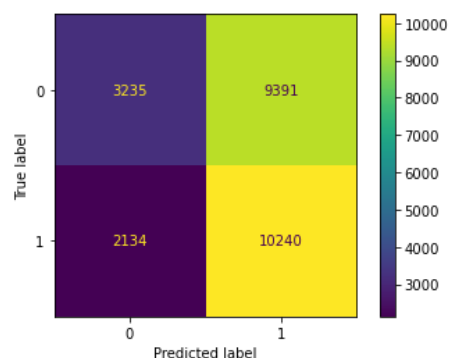
SVM	precision	recall	f1-score	support
0	0.51	0.75	0.61	12626
1	0.51	0.26	0.35	12374
accuracy			0.51	25000
macro avg	0.51	0.51	0.48	25000
weighted avg	0.51	0.51	0.48	25000

<Figure size 504x504 with 0 Axes>



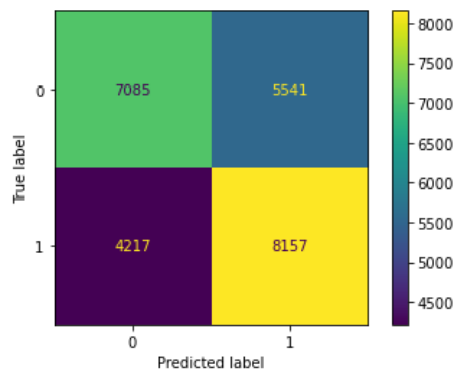
Neural Network	precision	recall	f1-score	support
0	0.60	0.26	0.36	12626
1	0.52	0.83	0.64	12374
accuracy			0.54	25000
macro avg	0.56	0.54	0.50	25000
weighted avg	0.56	0.54	0.50	25000

<Figure size 504x504 with 0 Axes>



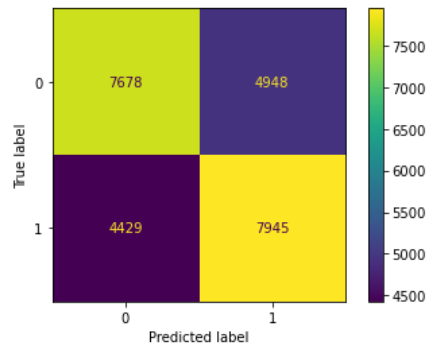
Linear Discriminant Analysis	precision	recall	f1-score	support
0	0.63	0.56	0.59	12626
1	0.60	0.66	0.63	12374
accuracy			0.61	25000
macro avg	0.61	0.61	0.61	25000
weighted avg	0.61	0.61	0.61	25000

<Figure size 504x504 with 0 Axes>



AdaBoostClassifier	precision	recall	f1-score	support
0	0.63	0.61	0.62	12626
1	0.62	0.64	0.63	12374
accuracy			0.62	25000
macro avg	0.63	0.63	0.62	25000
weighted avg	0.63	0.62	0.62	25000

<Figure size 504x504 with 0 Axes>



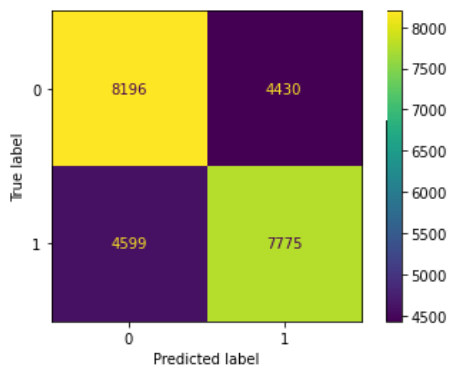
Light GBM

	precision	recall	f1-score	support
0	0.64	0.65	0.64	12626
1	0.64	0.63	0.63	12374
accuracy			0.64	25000
macro avg	0.64	0.64	0.64	25000
weighted avg	0.64	0.64	0.64	25000

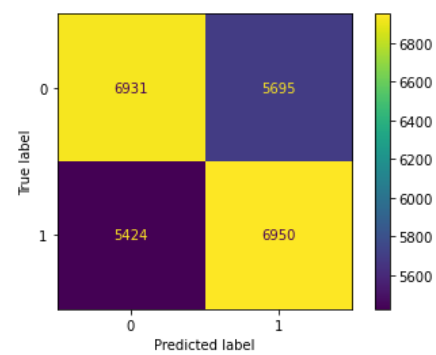
Decision tree

	precision	recall	f1-score	support
0	0.56	0.55	0.55	12626
1	0.55	0.56	0.56	12374
accuracy			0.56	25000
macro avg	0.56	0.56	0.56	25000
weighted avg	0.56	0.56	0.56	25000

<Figure size 504x504 with 0 Axes>



<Figure size 504x504 with 0 Axes>



KNeighborsClassifier

	precision	recall	f1-score	support
0	0.52	0.50	0.51	12626
1	0.51	0.53	0.52	12374
accuracy			0.51	25000
macro avg	0.51	0.51	0.51	25000
weighted avg	0.51	0.51	0.51	25000

<Figure size 504x504 with 0 Axes>

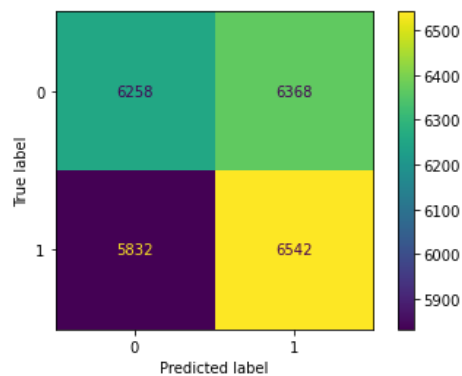


Fig 6.6: Confusion Matrix Of Different Models

CHAPTER 7

CONCLUSION

7.1 Conclusion

In our project, we have compared the eleven different machine learning algorithms including, Light GBM, Gradient Boosting, Xgboost, Decision Tree, AdaBoost Classifier, Logistic Regression, Random Forest, MLPClassifier- Neural Network, KNeighborsClassifier, SVM and Linear Discriminant Analysis. Various cross validation techniques were used to analyze the performance of these algorithms. Before that, the dataset provided by Microsoft is studied and various data optimisation techniques along with feature engineering are used to reduce and clean the dataset. The data is optimized, which helps us in reducing the memory usage and also saving the runtime. After training the dataset on above mentioned algorithms, various parameters are studied and analyzed. Then from that based on AUC score we choose the Light GBM model to process it with better hyperparameter tuning. So Light GBM provides the better results along with great efficiency and low runtime. This shows that Light GBM is the best gradient boosting algorithm in predicting the malware detection problem. So the Light GBM model has a great potential in predicting the malware hits and improving the cyber security for the future which will eventually lead in saving billions of dollars.

7.2 Future Work

Our research's key goal is to find the most effective algorithm for forecasting malware detection in computers. In our project, we have compared the eleven different machine learning algorithms including, Light GBM, Gradient Boosting, Xgboost, Decision Tree, AdaBoost Classifier, Logistic Regression, Random

Forest, MLPClassifier- Neural Network, KNeighborsClassifier, SVM and Linear Discriminant Analysis. We got that light gbm is most suitable model for this case it gives the most auc score compared to other models. We plan on doing more hypertuning on the parameters so that we can get more best results. We're still looking forward to the publication of another insightful dataset, similar to the one we used from Microsoft, which is currently the most informative dataset on our subject available over the internet. This way, we will solidify our theory while still contributing to the global battle against cybercrime.

REFERENCES

1. Microsoft Malware Prediction, research prediction competition, Kaggle.
<https://www.kaggle.com/c/microsoft-malware-prediction>
2. Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 8, pp. 1798–1828, 2013.
3. Jeff Heaton, "An Empirical Analysis of Feature Engineering for Predictive Modeling", 1 Nov 2020
4. M. Anderson, D. Antenucci, V. Bittorf, M. Burgess, M. J. Cafarella, A. Kumar, F. Niu, Y. Park, C. Ré, and C. Zhang, "Brainwash: A Data System for Feature Engineering.," in Proc. CIDR 2013, 2013.
5. K. P. Bennett and O. L. Mangasarian. Bilinear separation of two sets in n-space. Computational Optimization & Applications, 2:207–227, 1993.
6. Jerome H Friedman. Greedy function approximation: a gradient boosting machine. Annals of statistics, pages 1189–1232, 2001.
7. Guolin Ke , Qi Meng , Thomas Finley , Taifeng Wang , Wei Chen , Weidong Ma , Qiwei Ye , Tie-Yan Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree", 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.
8. [Tim Yee](#), "[What I've Learned: Microsoft Malware Prediction Competition on Kaggle](#)" Apr 20, 2019 ,
9. James Dietle, "[Microsoft Malware Prediction and its 9 million machines](#)", Mar 20,2019
10. James Bergstra,Yoshua Bengio, "Random Search for Hyper-Parameter Optimization", Journal of Machine Learning Research 13 (2012) 281-305,2012

APPENDIX A: PLAGIARISM REPORT

Microsoft Malware Prediction

ORIGINALITY REPORT

10 %	4 %	4 %	7 %
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Deakin University Student Paper	1 %
2	Submitted to The University of Law Ltd Student Paper	1 %
3	Andreas François Vermeulen. "Industrial Machine Learning", Springer Science and Business Media LLC, 2020 Publication	<1 %
4	Submitted to University of Alabama at Birmingham Student Paper	<1 %
5	Submitted to University of Westminster Student Paper	<1 %
6	Submitted to Swinburne University of Technology Student Paper	<1 %
7	xz.aliyun.com Internet Source	<1 %
8	Submitted to Lambeth College Student Paper	<1 %

9	towardsdatascience.com Internet Source	<1 %
10	Submitted to Hong Kong Baptist University Student Paper	<1 %
11	Submitted to Middle East College of Information Technology Student Paper	<1 %
12	Submitted to Coventry University Student Paper	<1 %
13	legaljobs.io Internet Source	<1 %
14	"Artificial Intelligence in Medicine", Springer Science and Business Media LLC, 2020 Publication	<1 %
15	Submitted to Georgia Institute of Technology Main Campus Student Paper	<1 %
16	github.com Internet Source	<1 %
17	Submitted to Higher Education Commission Pakistan Student Paper	<1 %
18	Submitted to University of Liverpool Student Paper	<1 %
19	www.geeksforgeeks.org Internet Source	

		<1 %
20	Submitted to University of Hertfordshire Student Paper	<1 %
21	Submitted to California State University, Sacramento Student Paper	<1 %
22	Submitted to ABV-Indian Institute of Information Technology and Management Gwalior Student Paper	<1 %
23	Submitted to American InterContinental University Student Paper	<1 %
24	Chengang Li, Yu Liu, Chengcheng Li. "Analysis and Research on the Use Situation of Public Bicycles Based on Spark Machine Learning", Proceedings of the 2019 International Conference on Data Mining and Machine Learning - ICDMML 2019, 2019 Publication	<1 %
25	en.wikipedia.org Internet Source	<1 %
26	Herbst, Allen, Sean McIlwain, Joshua J. Schmidt, Judd M. Aiken, C. David Page, and Lingjun Li. "Prion Disease Diagnosis by	<1 %

Proteomic Profiling", Journal of Proteome Research, 2009.

Publication

27	Submitted to Queen Mary and Westfield College	<1 %
	Student Paper	

28	Submitted to University of Stirling	<1 %
	Student Paper	

29	hdl.handle.net	<1 %
	Internet Source	

30	"Advanced Data Mining and Applications", Springer Science and Business Media LLC, 2017	<1 %
	Publication	

31	Chang LIN. "Naive Transfer Learning Approaches for Suspicious Event Prediction", 2019 IEEE International Conference on Big Data (Big Data), 2019	<1 %
	Publication	

32	Hanlin Sun, David Saad, Andrey Y. Lokhov. "Competition, Collaboration, and Optimization in Multiple Interacting Spreading Processes", Physical Review X, 2021	<1 %
	Publication	

33	www.hindawi.com	<1 %
	Internet Source	

34	Arooj Arif, Nadeem Javaid, Abdulaziz Aldegheishem, Nabil Alrajeh. "Big data analytics for identifying electricity theft using machine learning approaches in microgrids for smart communities", Concurrency and Computation: Practice and Experience, 2021 Publication	<1 %
35	www.aclweb.org Internet Source	<1 %
36	Submitted to University of Edinburgh Student Paper	<1 %
37	aclweb.org Internet Source	<1 %
38	digitalcommons.calpoly.edu Internet Source	<1 %
39	machinelearningmastery.com Internet Source	<1 %
40	www.canberra.edu.au Internet Source	<1 %
41	Erik Kostelansky, Emil Krsak, Tomas Kello. "Estimate Train Driving Time with Artificial Intelligence", 2019 International Conference on Information and Digital Technologies (IDT), 2019 Publication	<1 %
Submitted to Monash University		

42	Student Paper	<1 %
43	arxiv.org Internet Source	<1 %
44	fcsn.sites.usa.gov Internet Source	<1 %
45	scholarspace.library.gwu.edu Internet Source	<1 %
46	www.coursehero.com Internet Source	<1 %
47	Željko Vujović. "A case study of the application of WEKA software to solve the problem of liver inflammation", Research Square Platform LLC, 2021 Publication	<1 %

Exclude quotes On
Exclude bibliography On

Exclude matches Off