Evaluation of Models in Windows Malware Threat Detection

Mohit Battu
Sai Keerthi Appili

Department of Computer Science Engineering
Blekinge Institute of Technology
Karlskrona
Sweden
2015

in cooperation with

Abstract

In this paper, we propose a Machine Learning Model for malware detection. There has been a huge increase in the volume of malware, which poses a serious security threat. Few Malware types are viruses, worms, or other malicious programs. If a computer is infected by malware, hackers can harm consumers and companies in numerous ways which results in slowing down the performance of the system.

In this study, we investigate which strategies are essential to predict if a machine will soon be hit with malware. Under Machine learning, Decision Tree Classifier and Extreme Gradient Boosting are popular tools to build prediction models. Windows Malware threat detection can be solved by using a semi-supervised learning approach which consists of the models like Sequential Neural-Network, Decision Tree Classifier and Light Gradient Boosting Model facilitates us to detect Malware Threats.

For this Malware Detection project, we implemented 5 different types of Models namely Extreme Gradient Boosting, Decision Tree Classifier, Sequential Neural-Network, Random Forest Classifier, and Light Gradient Boosting Model. These above-referred models had achieved 65.79%, 58.8%, 51.75%, 50%, and 77.4% accuracies respectively for detecting a Malware Threat. Thus, we propose a Light Gradient Boosting Model, a Machine Learning approach to detect malware samples.

The proposed model achieved 77.8% accuracy in detecting Windows Malware Threats. The proposed model has got a 68% precision and 71% recall rate on the Train dataset whereas on Test Dataset proposed model has achieved 66% precision and 69% recall rate.

We conclude that on Windows challenging malware detection dataset demonstrates that our proposed Model can handle large-scale data, takes lower memory to run, and achieves better performance in comparison with the other 4 models. Thus, it is the best prediction model for Windows Malware Threat Detection.

Keywords:

Decision Tree Classifier, Extreme Gradient Boosting, Light Gradient Boosting Model, Malware Detection, Random Forest Classifier, and Sequential Neural-Network.

Acknowledgements

We would like to express our deep and sincere gratitude to Prof. Wlodek Kulesza, for his valuable guidance and support. He taught us the methodology to present the research works as clearly as possible and convincingly conveyed the concept of critical thinking. It was a great privilege and honour for us to study under his guidance. Our completion would not have been accomplished without his timely advice despite his busy schedule. Finally, we would like to thank our friends and family for their motivation and encouragement throughout the learning period.

Sai Keerthi Appili
Battu Mohit

Contents

List of figures

List of tables

List of symbols

List of acronyms

Acronym	Unfolding
DTC	Decision Tree Classifier
RFC	Random Forest Classifier
LGB	Light Gradient Boosting
XGB	Extreme Gradient Boosting
SNN	Sequential Neural Network
CART	Classification and Regression Tree

1 Chapter: Introduction

Malware which is a hazard to the computer enters the user's system without permission. Malware is also known as "Malicious Software". It is a great threat in the computing world.

The performance of the system also gets degraded when the malicious software enters the system, thereby it scans for the vulnerabilities present in the operating system and performs unauthorized actions on the system. The volume of malware is evolving rapidly and is getting out of control. Malware is abundant over the internet and can easily infect various hosts simultaneously. Once the Malware enters the user's system, then the system results in unresponsive, and the application halts from opening.

Malware is user-friendly for hackers. Nowadays, it can earn more income for attackers by providing them to steal user's login credentials, credit card information, browser auto-fill data, and cryptocurrency wallets.

More and more companies are trying different strategies to solve this problem of Malware, but due to the large volume of Malware, it can still escape even complex forms of authentications.

A machine learning-based model "Light Gradient Boosting Model" is used to predict if the machine gets hit by the malware using certain features that are dependent on the identification of Malware.

2 Chapter: Survey of related work

In this section, there is a brief description of the Malware Dataset and the model's definition.

2.1 DataSet:

Malware Threat Data Set consists of 83 features and 8.92 million number of records. The total size of the Data Set is approximately 7.89GB. This Data Set was mainly divided into three main parts consisting of Numerical, Binary, and Categorical Columns. There are 7 numerical, 21 binary, and 54 categorical columns which sum up to the 83 features in our Dataset. Each row in this dataset corresponds to a windows machine, which is uniquely identified by a MachineIdentifier. *MachineIdentifier* is the first column in our dataset and is called an independent variable. *HasDetections* is the target variable and it is present in the last column of the Malware dataset. *HasDetections* is a binary column which indicates that Malware was Detected on the machine in the form of "0" and "1". Here "0" indicates that Malware is not found and "1" represents Malware is found, see Table 2-1 [1].

Table 2-1: DataSet Description

Serial#	MachineIdentifier	Detected#
1	0000028988387b115f69f31a3bf04f09	0
2	000007535c3f730efa9ea0b7ef1bd645	0
3	000007905a28d863f6d0d597892cd692	0
4	00000b11598a75ea8ba1beea8459149f	1
5	000014a5f00daa18e76b81417eeb99fc	1

This Data Set was downloaded from the Kaggle repository under the name of Microsoft Malware Prediction.

2.2 Decision Tree Classifier:

A Decision Tree is a simple representation of classifying examples. The Decision Tree has two major classes namely classification and regression tree. These two terms together are called CART. This CART was first invented in 1984 by Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone [2]. We can address the amount of uncertainty in the data for set S as

$$H(S) = -\sum p(x) log p(x) \quad (2\text{-}1)$$

Here *p(x)* is the proportion of the number of elements in class *x* to the number of elements in set *S* and *x* is the set of classes in *S*.

2.3 Random Forest Classifier:

Tin Kam Ho had created the first algorithm for the random forests in the year 1995. Random forests are often used as *BlackBox* models in industries since they produce sufficient estimates across a wide range of data [3]. We can address the estimated probability for predicting class k for a sample as

$$p(k|x) = \frac{1}{T}\sum_{t=1}^{T} P_t(k|x) \quad (2\text{-}2)$$

where $p(k|x)$ is the estimated density of class labels of the leaf of the t^{th} tree.

2.4 XGBoost:

Tianqi Chen had created the XGBoost algorithm. This algorithm was released on 27th March 2014. XGBoost algorithm was developed with the help of C++, Java, Python, R, Julia, and Scala. It provides a parallel tree boosting that solve many data science problems in a fast and accurate way. We can address that the objective function is a sum of the specific loss function and regularization terms for all predictions. Mathematically, it can be represented as:

$$obj(\theta) = \sum_{i}^{l} (y_i - \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \quad (2\text{-}3)$$

where y_i is original values and $\hat{y}_i$ is predicted values. Here $\sum_{k=1}^{K} \Omega(f_k)$ is a sum of the regularization term.

2.5 LightGBM:

Guolin Ke had created the Light GBM algorithm. It was first released in the year 2016. This algorithm was developed with the help of C++, Python, R, and C. It provides a faster training speed and higher efficiency. It is histogram-based and places continuous values into discrete bins, which leads to faster training and more efficient memory usage [4].

2.6 Sequential Neural Network:

Warren McCulloch and Walter Pitts had created a computation model for neural networks in the year 1943. These algorithms were developed with the help of threshold logic [5]. We can address the cost function of a sequential neural network as

$$J = \sum_{i}^{n} \frac{1}{2} (y - \hat{y})^2 \quad (2\text{-}4)$$

Here cost function sums over all samples from *i* to *n*, where *y* is a true value and $\hat{y}$ is a predicted value.

3 Chapter: Problem statement, objectives and main contribution

Malware is present in huge quantities and can easily affect as many hosts as possible. Malware avoids most of the security measures undertaken by the user thereby forcing the anti-malware groups to build more robust software to detect these malware attacks. A foremost part of protecting a computer system from a malware attack is to identify whether a given piece of file is malware or not.

Research Problem: How to develop a Machine learning Model for detecting the Malware threats on Windows Operating System?

1. For detecting malware threats in a windows system, "Microsoft Malware Prediction" has provided a train and test data set consisting of more than a billion observations which are made available in the Kaggle Dataset Repository.
2. Observed values present in the training dataset of Microsoft Malware Prediction are first refined by discarding missing values then feature engineered with the help of data science tools resulting in improved efficiency on unseen data.
3. Malware threat detection can be solved by using a semi-supervised learning approach which consists of the models like Sequential Neural-Network, Decision Tree Classifier, Random Forest Classifier, Extreme Gradient Boosting, and Light Gradient Boosting Model enables us to detect whether a windows machine will be soon hit with malware or not.
4. The above-referred models are different kinds of algorithmic approaches where only one of the best performing models shall be considered for predicting Windows Malware Threat Detection.

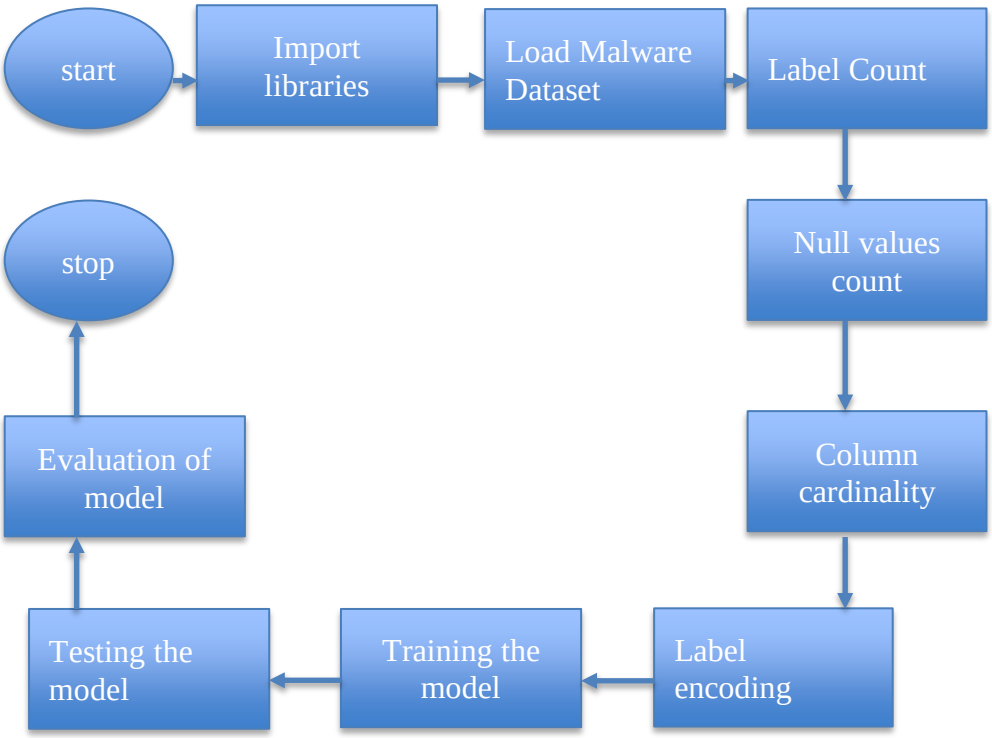
The main contributions of the project are as follows:

- Implementation of the model in Python 3 environments on Jupyter Notebook (Anaconda 3).
- The implementation of python libraries in the project are Pandas, NumPy, Matplotlib, Seaborn, Scikit-Learn, Keras, and Light Gradient Boosting [6].
- Validation of the model is done by using the Confusion matrix and Model Metrics.

4 Chapter: Solution

We anticipate that the Light GBM model can give better results and accuracy than Decision Tree Classifier, Random Forest, XGBoost, Sequential Neural Networks. The dataset we gathered is of 8.9 million records with many unknown values for certain features. To tackle the large dataset Light GBM seems more appropriate which takes lower memory to run and achieves better performance.

4.1 Modeling



4.2 Implementation or Application:

Implementation of algorithm is done in Jupyter notebook using a python programming language. Malware Dataset downloaded from the Kaggle Repository [1].

Table 4-1: DataSet Name and Type

Column Name	**Type**
MachineIdentifier	Category
ProductName	Category
EngineVersion	Category
AppVersion	Category
AvSigVersion	Category
IsBeta	Int8
RtpStateBitfield	Float16
IsSxsPassiveMode	Int8
DefaultBrowsersIdentifier	Float16
AVProductStatesIdentifier	Float32
AVProductsInstalled	Float16
AVProductsEnabled'	Float16
HasTpm	Int8
CountryIdentifier	Int16
CityIdentifier	Float32
OrganizationIdentifier	Float16
GeoNameIdentifier	Float16
LocaleEnglishNameIdentifier	Int8
Platform	Category
Processor	Category
OsVer	Category
OsBuild	Int16
OsSuite	Int16
OsPlatformSubRelease	Category
OsBuildLab	Category
SkuEdition	Category
IsProtected	Float16
AutoSampleOptIn	Int8
PuaMode	Category

Column Name	Type
SMode	Float16
IeVerIdentifier	Float16
SmartScreen	Category
Firewall	Float16
UacLuaenable	Float32
Census_MDC2FormFactor	Category
Census_OSVersion	Category
Census_OSArchitecture	Category
Census_OSBranch	Category
Census_OSBuildNumber	Int16
Census_OSBuildRevision	Int32
Census_OSEdition	Category
Census_OSSkuName	Category
Census_OSInstallTypeName	Category
Census_OSInstallLanguageIdentifier	Float16
Census_OSUILocaleIdentifier	Int16
Census_OSWUAutoUpdateOptionsName	Category
Census_IsPortableOperatingSystem	Int8
Census_GenuineStateName	Category
Census_ActivationChannel	Category
Census_IsFlightingInternal	Float16
Census_IsFlightsDisabled	Float16
Census_FlightRing	Category
Census_ThresholdOptIn	Float16
Census_FirmwareManufacturerIdentifier	Float16
Census_FirmwareVersionIdentifier	Float32
Census_IsSecureBootEnabled	Int8
Census_IsWIMBootEnabled	Float16
Census_IsVirtualDevice	Float16
Census_IsTouchEnabled	Int8
Census_IsPenCapable	Int8
Census_IsAlwaysOnAlwaysConnectedCapable	Float16
Wdft_IsGamer	Float16
Wdft_RegionIdentifier	Float16
Census_OEMNameIdentifier	Float16
Census_OEMModelIdentifier	Float32
Census_ProcessorCoreCount	Float16
Census_ProcessorManufacturerIdentifier	Float16

Column Name	Type
Census_ProcessorModelIdentifier	Float16
Census_ProcessorClass	Category
Census_PrimaryDiskTotalCapacity	Float32
Census_PrimaryDiskTypeName	Category
Census_SystemVolumeTotalCapacity	Float32
Census_HasOpticalDiskDrive	Int8
Census_TotalPhysicalRAM	Float32
Census_ChassisTypeName	Category
Census_InternalPrimaryDiagonalDisplaySizeInInches	Float16
Census_InternalPrimaryDisplayResolutionHorizontal	Float16
Census_InternalPrimaryDisplayResolutionVertical	Float16
Census_PowerPlatformRoleName	Category
Census_InternalBatteryType	Category
Census_InternalBatteryNumberOfCharges	Float32
Census_DeviceFamily	Category
HasDetections	Int8

The “HasDetections” column is the target variable used to predict if malware is hit. Required libraries are to be installed before the implementation of the model.

```
!pip3 install lightgbm
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import lightgbm as lgb
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, mean_squared_error, classification_report
%matplotlib inline
sns.set(style="whitegrid")
warnings.filterwarnings("ignore")
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

Figure 4-1: Importing Python Libraries

Loading DataSet:

The dataset is imported to Jupyter notebook using panda's module.

Load Data

```
In [3]: train = pd.read_csv("train.csv",dtype=dtypes)
```

Figure 4-2: Loading Train DataSet

Exploratory Data Analysis:

The data comprises more categorical columns than numerical columns and binary columns.

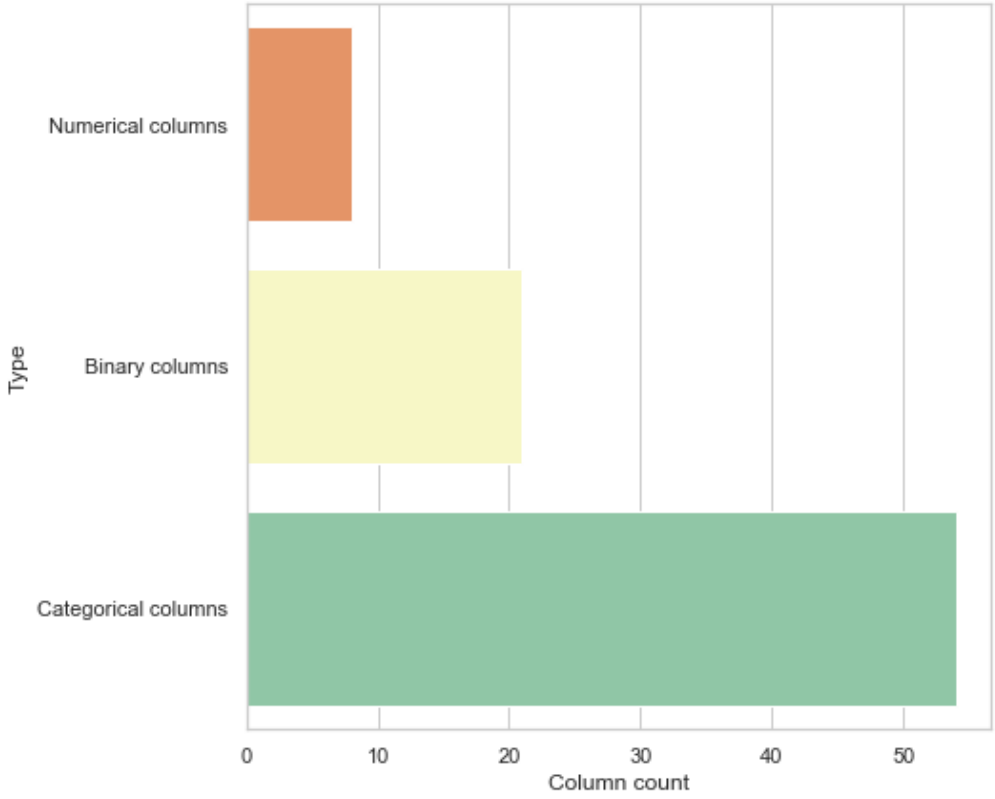


Figure 4-3: Column Types

Label Count:

The target variable “HasDetections” binary value counts are inspected to find if any bias exists or an imbalance of data. There is balanced data for “HasDetections” without any problem of misclassification in our dataset. Binary count of 0’s and 1’s in the “HasDetections” column is in the same proportion.

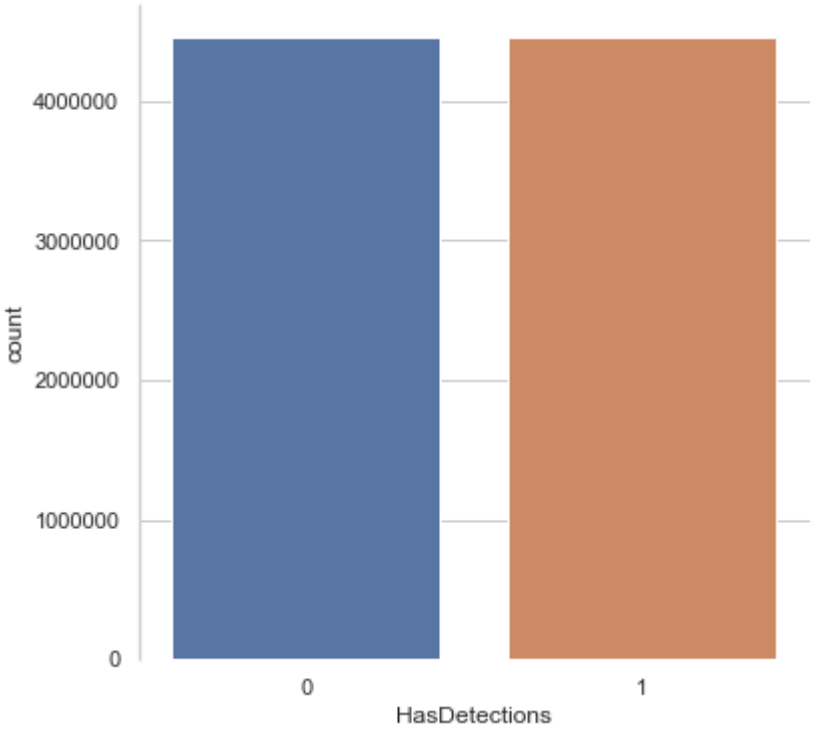


Figure 4-4: Binary Label Count

Column Cardinality:

Each category cardinality count is noticed and analyzed to remove columns with high cardinality.

Figure 4-5: Cardinality Graph

```
Remove columns with high cardinality (more than 500 categories)

In [5]: high_cardinality_columns = [c for c in categorical_columns if train[c].nunique() > 500]
        high_cardinality_columns.remove('MachineIdentifier')  # Remove ID
        train.drop(high_cardinality_columns, axis=1, inplace=True)
        print('Columns with high cardinality: ', high_cardinality_columns)
```

Figure 4-6: Cardinality Code

Removing columns with cardinality number greater than 500 which we assumed as high cardinality.

Counting the number of null values:

The number of null values is analyzed for each feature to drop the columns with more percentage of missing values.

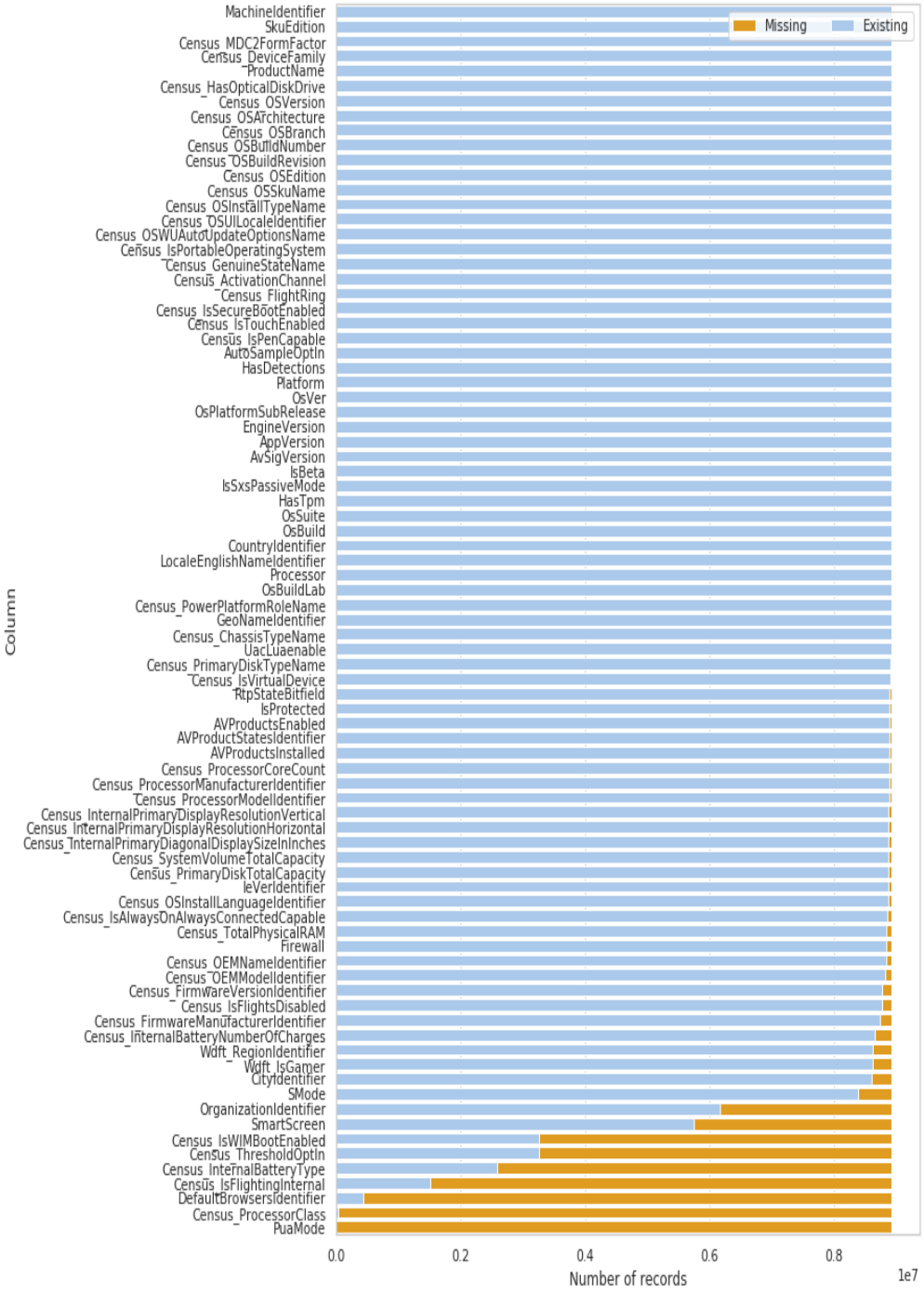


Figure 4-7: Missing Values Count

Removing rows with a null value:

All the rows where the null values exist are removed without leading to any problem while training the model.

Dropping the columns with more than 40% missing data.

```
Remove columns with more than 40% missing data

In [6]: high_null_columns = [c for c in train.columns if train[c].count() < len(train)*0.6]
        train.drop(high_null_columns, axis=1, inplace=True)
        print('Columns with more than 40% null values: ', high_null_columns)
```

Figure 4-8: Dropping Columns Code

Dropping the rows with null values

```
In [11]: # Remove rows with NA
         train.dropna(inplace=True)
```

Figure 4-9: Dropping Rows with Null

Model Implementation:

1. Light GBM:

Training data is fitted into the LightGBM Model. The hyperparameters are given to fine-tune the model while training the data to give much better accuracy.

```
In [15]: params = {'num_leaves': 60,
                   'min_data_in_leaf': 100,
                   'objective':'binary',
                   'max_depth': -1,
                   'learning_rate': 0.1,
                   "boosting": "gbdt",
                   "feature_fraction": 0.8,
                   "bagging_freq": 1,
                   "bagging_fraction": 0.8 ,
                   "bagging_seed": 1,
                   "metric": 'auc',
                   "lambda_l1": 0.1,
                   "random_state": 133,
                   "verbosity": -1}

In [16]: lgb_train = lgb.Dataset(X_train, label=Y_train)
         lgb_val = lgb.Dataset(X_val, label=Y_val)
```

Figure 4-10: Hyperparameter Code

After training the data the Light GBM accuracy is approximately 74%.

```
Training until validation scores don't improve for 200 rounds
[100]   training's auc: 0.740585        valid_1's auc: 0.739741
[200]   training's auc: 0.745586        valid_1's auc: 0.743382
[300]   training's auc: 0.74852 valid_1's auc: 0.744822
[400]   training's auc: 0.75089 valid_1's auc: 0.745784
[500]   training's auc: 0.752965        valid_1's auc: 0.746435
[600]   training's auc: 0.754791        valid_1's auc: 0.746862
[700]   training's auc: 0.756549        valid_1's auc: 0.747208
[800]   training's auc: 0.758189        valid_1's auc: 0.747502
[900]   training's auc: 0.759716        valid_1's auc: 0.747668
[1000]  training's auc: 0.761179        valid_1's auc: 0.747877
[1100]  training's auc: 0.76262 valid_1's auc: 0.74798
[1200]  training's auc: 0.76398 valid_1's auc: 0.748088
[1300]  training's auc: 0.765323        valid_1's auc: 0.74819
[1400]  training's auc: 0.766574        valid_1's auc: 0.748205
[1500]  training's auc: 0.767805        valid_1's auc: 0.74822
[1600]  training's auc: 0.769055        valid_1's auc: 0.748248
[1700]  training's auc: 0.770259        valid_1's auc: 0.748273
[1800]  training's auc: 0.771429        valid_1's auc: 0.74831
[1900]  training's auc: 0.772599        valid_1's auc: 0.748307
[2000]  training's auc: 0.773737        valid_1's auc: 0.748349
[2100]  training's auc: 0.774891        valid_1's auc: 0.748356
[2200]  training's auc: 0.77602 valid_1's auc: 0.74834
Early stopping, best iteration is:
[2052]  training's auc: 0.774329        valid_1's auc: 0.748376
```

Figure 4-11: LGB Accuracy Score

2. Decision Tree Classifier:

Label encoding of categorical variables is done to fit in the Decision Tree Classifier. Training data is fitted into the Decision Tree classifier. The model showed an accuracy of approximately 58%.

Training the model using Decision Tree Classifier

```
In [21]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.model_selection import train_test_split
         x_train, X_test, y_train, y_test = train_test_split(train,labels, test_size=0.2, random_state=42)

In [22]: # TODO: Define the classifier, and fit it to the data
         model = DecisionTreeClassifier()
         model.fit(x_train, y_train)

Out[22]: DecisionTreeClassifier()

In [23]: # Actual class predictions
         accuracy1 = model.score(X_test,y_test)
         print(accuracy1)

         0.5885956829882839
```

Figure 4-12: DTC Accuracy Score

3. Random Forest Classifier:

Converting all categorical columns to numerical type with label encoders. The training data after fitting into the model shown an accuracy of approximately 50%.

```
In [35]: from sklearn.ensemble import RandomForestClassifier
         clf = RandomForestClassifier(criterion='gini', max_features=3, bootstrap= False, min_samples_split=40,
                                      max_depth=None, min_samples_leaf=10)
         clf.fit(x_tr,y_tr)

Out[35]: RandomForestClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                                criterion='gini', max_depth=None, max_features=3,
                                max_leaf_nodes=None, max_samples=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=10, min_samples_split=40,
                                min_weight_fraction_leaf=0.0, n_estimators=100,
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)

In [36]: y_pred = clf.predict(x_ts)
         predictions = [round(value) for value in y_pred]
         accuracy = accuracy_score(yts, predictions)
         print("Accuracy: %.2f%%" % (accuracy * 100.0))

         Accuracy: 50.07%
```

Figure 4-13: RFC Accuracy Score

4. XGBoost:

The model after fitting the data shown an accuracy of about 65%.

Training the model using XGBoost

```
In [28]: xtr,xts,ytr,yts = train_test_split(train, labels, test_size=0.2)

In [30]: binary_columns, true_numerical_columns, categorical_columns = update_feature_lists()

         # Label encoder
         indexer = {}
         for col in categorical_columns:
             _, indexer[col] = pd.factorize(xtr[col])

         for col in categorical_columns:
             xtr[col] = indexer[col].get_indexer(xtr[col])
             xts[col] = indexer[col].get_indexer(xts[col])

In [31]: from xgboost import XGBClassifier
         from sklearn.metrics import accuracy_score
         model = XGBClassifier()
         model.fit(xtr,ytr)
         y_pred = model.predict(xts)
         predictions = [round(value) for value in y_pred]
         accuracy = accuracy_score(yts, predictions)
         print("Accuracy: %.2f%%" % (accuracy * 100.0))

         Accuracy: 65.79%
```

Figure 4-14: XGB Accuracy Score

5. Sequential Neural Network:

The layers are added sequentially with activation function as relu, loss as binary entropy, optimizer as adam. Forward Propagation and Backward propagation occur for every iteration and to minimize the loss adjusts the weights accordingly.

```
Epoch 1/10
 - 199s - loss: 152585343.5228 - accuracy: 0.5008
Epoch 2/10
 - 188s - loss: 4591110.4726 - accuracy: 0.5141
Epoch 3/10
 - 162s - loss: 0.8504 - accuracy: 0.5174
Epoch 4/10
 - 163s - loss: 0.6926 - accuracy: 0.5174
Epoch 5/10
 - 172s - loss: 0.6926 - accuracy: 0.5174
Epoch 6/10
 - 163s - loss: 0.6926 - accuracy: 0.5174
Epoch 7/10
 - 165s - loss: 0.6926 - accuracy: 0.5174
Epoch 8/10
 - 167s - loss: 0.6926 - accuracy: 0.5174
Epoch 9/10
 - 167s - loss: 0.6926 - accuracy: 0.5174
Epoch 10/10
 - 164s - loss: 0.6926 - accuracy: 0.5174
Accuracy: 51.63
```

Figure 4-15: SNN Accuracy Score

The accuracy after 100 iterations in training the data is about 51%.

4.3 Validation or Verification

The validation of the model is performed, and the results are interpreted to know the misclassification count and accurate prediction count. The confusion matrix is used to represent true positives, true negatives, false positives, and false negatives.

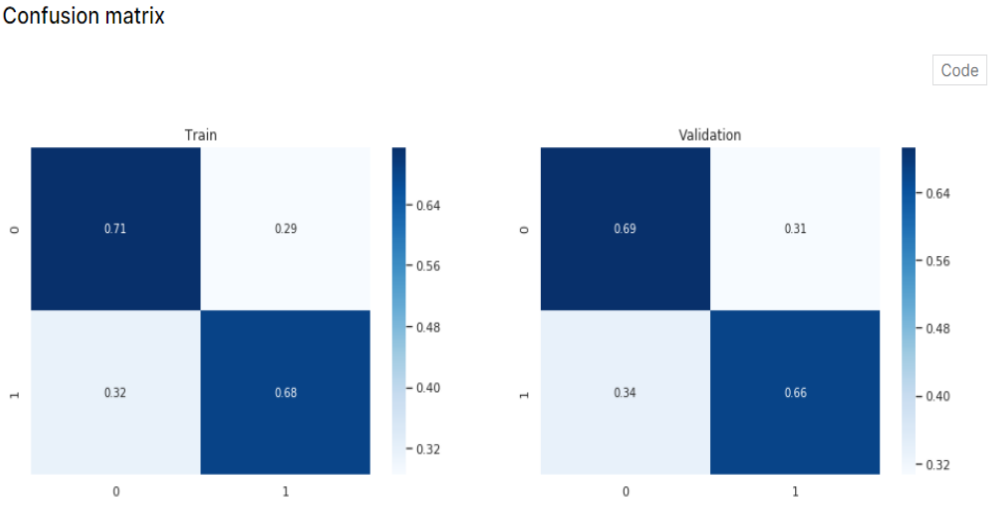


Figure 4-16: Confusion Matrix for LGBM

Model Metrics:

The metrics are analysed to know the correctness of the LightGBM Model thereby, evaluated precision, recall, and f1score.

Table 4-2: Model Metrics for Training

Training	Precision	Recall	F1score
HasDetections = 0	0.68	0.71	0.70
HasDetections = 1	0.72	0.68	0.70

Table 4-3: Model Metrics for Testing

Validation	Precision	Recall	F1score
HasDetections = 0	0.66	0.69	0.68
HasDetections = 1	0.70	0.66	0.68

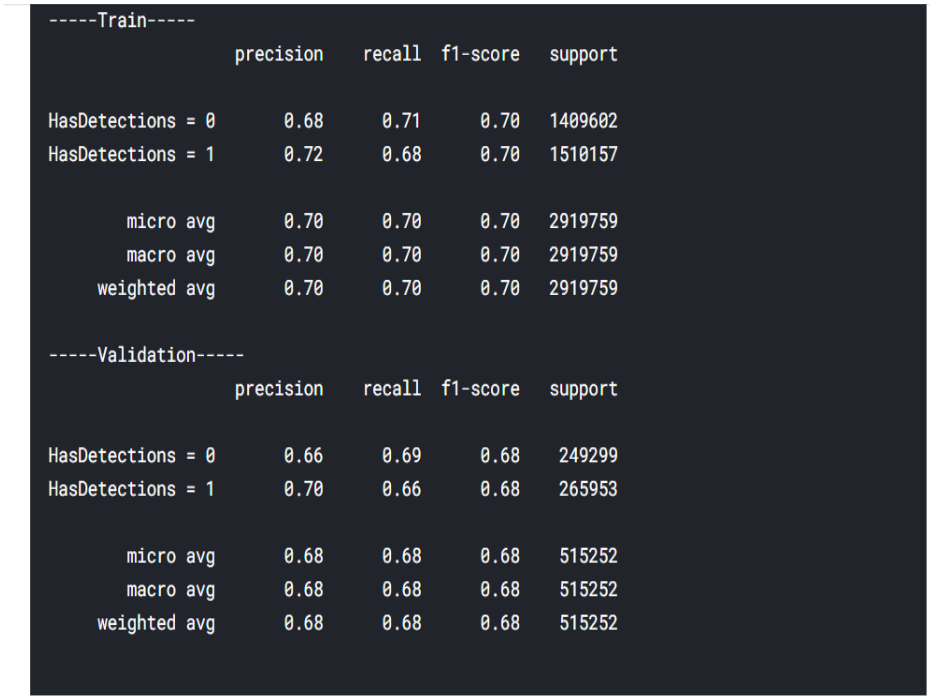


Figure 4-17: Model Metrics

5 Chapter: Conclusion and future work

5.1 Conclusion:

We can summarize that Light Gradient Boosting will produce better results and generates more accurate results than the Random Forest, Decision Tree Classifier, XG Boost and Sequential Neural Networks. The Light GBM will produce the 77.4% accuracy that is 11.61% more accurate than XGBoost,18.6% more accurate than Decision Tree Classifier, 25.7% more accurate than Sequential Neural Network, and 27.4% more accurate than Random Forest Classifier.

5.2 Future Works:

Try Dimensionality reduction on high cardinality features, techniques like Principal Component Analysis, t-distributed stochastic neighbor embedding or auto encoders may help. Other approaches to parameter tunning like Bayesian optimization may help as well. Impute the data using multivariate imputation by chained equations. Use other models which can use categorical data without encoding.

6 Reference

[1] “Microsoft Malware Prediction.” https://kaggle.com/c/microsoft-malware-prediction (accessed Sep. 17, 2020).

[2] P. H. Swain and H. Hauska, “The decision tree classifier: Design and potential,” *IEEE Transactions on Geoscience Electronics*, vol. 15, no. 3, pp. 142–147, Jul. 1977, doi: 10.1109/TGE.1977.6498972.

[3] V. Y. Kulkarni and P. K. Sinha, “Pruning of Random Forest classifiers: A survey and future directions,” in *2012 International Conference on Data Science Engineering (ICDSE)*, Jul. 2012, pp. 64–68, doi: 10.1109/ICDSE.2012.6282329.

[4] G. Ke *et al.*, “LightGBM: A Highly Efficient Gradient Boosting Decision Tree,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 3146–3154.

[5] L. Denoyer and P. Gallinari, “Deep Sequential Neural Network,” *arXiv:1410.0510 [cs]*, Oct. 2014, Accessed: Sep. 17, 2020. [Online]. Available: http://arxiv.org/abs/1410.0510.

[6] P. Lemenkova, “Python libraries matplotlib, seaborn and pandas for visualization geospatial datasets generated by QGIS,” *Analele stiintifice ale Universitatii ”Alexandru Ioan Cuza” din Iasi - seria Geografie*, vol. 1, pp. 13–32, Sep. 2020, doi: 10.6084/m9.figshare.13010069.