# Microsoft Malware Prediction

EE769: Introduction to Machine Learning
Course Project

Aniket Ghosh
*Dept. of Mechanical Engg.*
*Indian Institute of Technology, Bombay*
Mumbai, India
18D110014@iitb.ac.in

Vinit Awale
*Dept. of Electrical Engg.*
*Indian Institute of Technology, Bombay*
Mumbai, India
18D070067@iitb.ac.in

***Abstract*—In the present times, malware attacks are one of the rising concerns regarding the security of users' data. In this project we attempt to use Machine learning techniques to determine the probability of a machine to be affected by malware. Comparison of the the Machine Learning Models Logistic Regression, KNN, Random Forest Classifier and LGBM for making predictions has been done. Since the dataset is quite large (more than 8GB!), the problem becomes a very high dimensionality problem. Hence, data preprocessing and feature engineering becomes very important in handling the given dataset and this has been done in the implementation. A prediction accuracy of 63.15 percent was obtained in making predictions.**

***Index Terms*—Malware prediction, Machine Learning, Random Forest Classifier, KNN, Logistic Regression, LGBM (Light Gradient Boosting Machine)**

## I. Introduction

In the present times, malware attacks are one of the rising concerns regarding the security of users' data. The malware industry is a well-organized, well-funded market dedicated to evading traditional security measures. Once a computer is infected by malware, criminals can hurt consumers and enterprises in many ways.According to the survey conducted by NetMarketShare, 87% of the Operating Systems Market Share is ruled by Microsoft Windows. Hence, this problem statement was uploaded by *Microsoft, Windows Defender ATP Research, Northeastern University College of Computer and Information Science and Georgia Tech. Institute* on Kaggle competitions.

In this project we attempt to use Machine learning techniques to determine the probability of a machine to be affected by malware.The challenge one faces while solving this problem is obtaining the dataset with appropriate features for determining the probability of a malware attack. However, Microsoft has provided an unprecedented malware dataset which resolves this problem for us. However, the dataset is quite large and the problem becomes a very high dimensionality problem owing to the large number of features included in the data set. As a result, the data preprocessing and feature engineering becomes very important in handling the given dataset.

Summarising the results of our project:

- We compared the Machine Learning Models Logistic Regression, KNN, Random Forest Classifier and LGBM for making predictions.
- We obtained a prediction accuracy of 63.15% in making predictions
- Concluded that the feature AVSig Version (which is the version of the database of signatures of the anti virus) is the most important feature in determining if the machine is likely to face a malware attack

## II. Background and Previous Work

Since the dataset is quite large and the problem becomes a very high dimensionality problem data preprocessing and feature engineering becomes very important in handling the given dataset. Hence, apart from knowing basic ML having knowledge about the Computer Architecture and Cyber Security can help in determining the features which contribute more towards making the final prediction. Previously, various competitors have attempted to get the highest possible accuracy using various ML models. In this project, we attempt to compare the various models taught in the course and to compare them with the benchmark scores obtained by the other competitors.

## III. Datasets

For this project, we have used the dataset from the Kaggle competition where this problem statement was posted. The dataset has two files, train.csv and test.csv .

- Train.csv : This data file has 8921483 entries and 83 columns for each entry. These columns also include 'HasDetections' which indicates whether the given machine has been affected by malware.
- Test.csv : This data file has 7853253 entries. It contains 82 columns for each entry since it doesn't include the 'HasDetections' columns as compared to the Train.csv.

## IV. Procedure, Experiments and Results

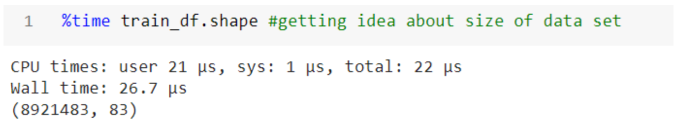
### *A. Reducing memory requirements*

The raw data obtained from Kaggle was too big to even be opened. Since the dataset is huge, our first task to load the data successfully will be accomplished by reducing the memory requirements, so what we planned to do is as follows

1) We switch the binary values to int8
2) We switch the binary values which contain the missing values to float16

## B. *Initial Look at the Data*

We first got an idea about the size of the dataset.

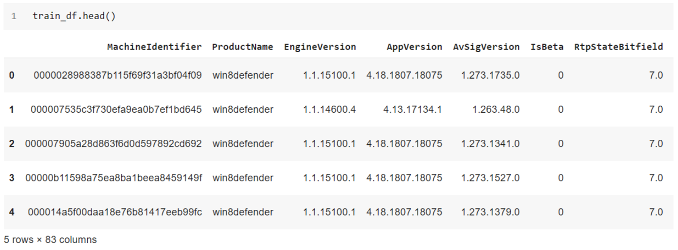


```
%time train_df.shape #getting idea about size of data set
```

```
CPU times: user 21 µs, sys: 1 µs, total: 22 µs
Wall time: 26.7 µs
(8921483, 83)
```

Here, the size of the training data set is as mentioned eariler, 8921483 entries and 83 columns corresponding to each entry.

Also we had a look at the various features included in the dataset.



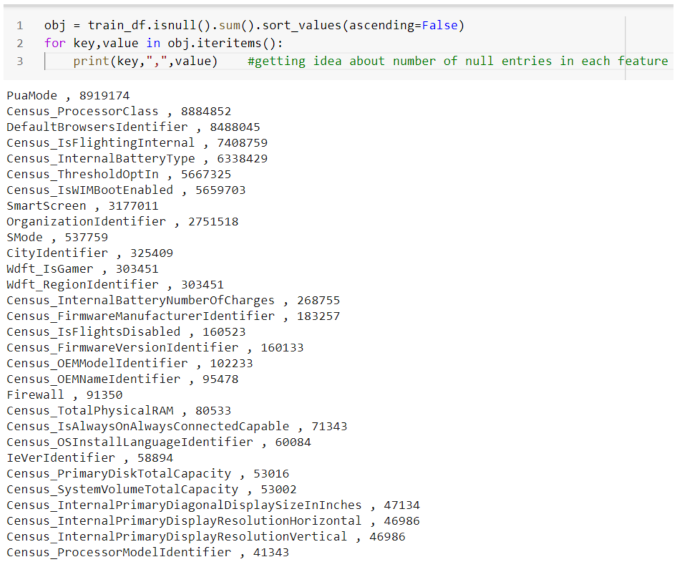
```
train_df.head()
```

| | MachineIdentifier | ProductName | EngineVersion | AppVersion | AvSigVersion | IsBeta | RtpStateBitfield |
|---|---|---|---|---|---|---|---|
| **0** | 0000028988387b115f69f31a3bf04f09 | win8defender | 1.1.15100.1 | 4.18.1807.18075 | 1.273.1735.0 | 0 | 7.0 |
| **1** | 000007535c3f730efa9ea0b7ef1bd645 | win8defender | 1.1.14600.4 | 4.13.17134.1 | 1.263.48.0 | 0 | 7.0 |
| **2** | 000007905a28d863f6d0d597892cd692 | win8defender | 1.1.15100.1 | 4.18.1807.18075 | 1.273.1341.0 | 0 | 7.0 |
| **3** | 00000b11598a75ea8ba1beea8459149f | win8defender | 1.1.15100.1 | 4.18.1807.18075 | 1.273.1527.0 | 0 | 7.0 |
| **4** | 000014a5f00daa18e76b81417eeb99fc | win8defender | 1.1.15100.1 | 4.18.1807.18075 | 1.273.1379.0 | 0 | 7.0 |

5 rows × 83 columns

## C. *Exploratory Data Analysis*

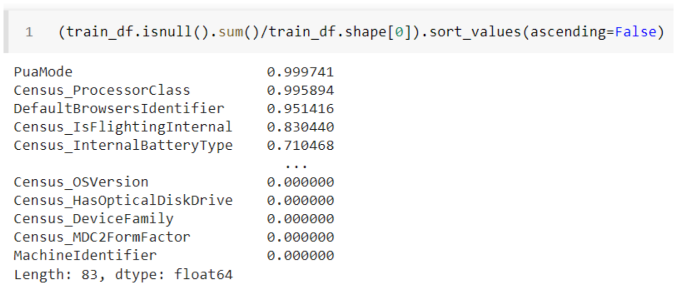
The raw data that we obtained from Kaggle has very high dimensionality and missing values. Also the size of the data is comparatively very large as mentioned earlier. Hence, this step was very important in getting the right data before implementing any Machine Learning Model on it.
First, we got an idea about the number of missing entries.



```
obj = train_df.isnull().sum().sort_values(ascending=False)
for key,value in obj.iteritems():
    print(key,",",value)    #getting idea about number of null entries in each feature
```

```
PuaMode , 8919174
Census_ProcessorClass , 8884852
DefaultBrowsersIdentifier , 8488045
Census_IsFlightingInternal , 7408759
Census_InternalBatteryType , 6338429
Census_ThresholdOptIn , 5667325
Census_IsWIMBootEnabled , 5659703
SmartScreen , 3177011
OrganizationIdentifier , 2751518
SMode , 537759
CityIdentifier , 325409
Wdft_IsGamer , 303451
Wdft_RegionIdentifier , 303451
Census_InternalBatteryNumberOfCharges , 268755
Census_FirmwareManufacturerIdentifier , 183257
Census_IsFlightsDisabled , 160523
Census_FirmwareVersionIdentifier , 160133
Census_OEMModelIdentifier , 102233
Census_OEMNameIdentifier , 95478
Firewall , 91350
Census_TotalPhysicalRAM , 80533
Census_IsAlwaysOnAlwaysConnectedCapable , 71343
Census_OSInstallLanguageIdentifier , 60084
IeVerIdentifier , 58894
Census_PrimaryDiskTotalCapacity , 53016
Census_SystemVolumeTotalCapacity , 53002
Census_InternalPrimaryDiagonalDisplaySizeInInches , 47134
Census_InternalPrimaryDisplayResolutionHorizontal , 46986
Census_InternalPrimaryDisplayResolutionVertical , 46986
Census_ProcessorModelIdentifier , 41343
```

Then we determined the proportion of data that was actually missing for each feature

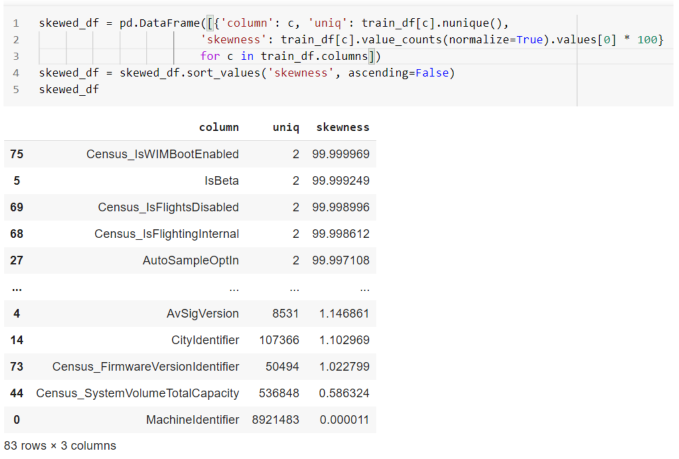


```
(train_df.isnull().sum()/train_df.shape[0]).sort_values(ascending=False)
```

```
PuaMode                        0.999741
Census_ProcessorClass          0.995894
DefaultBrowsersIdentifier      0.951416
Census_IsFlightingInternal     0.830440
Census_InternalBatteryType     0.710468
                                  ...
Census_OSVersion               0.000000
Census_HasOpticalDiskDrive     0.000000
Census_DeviceFamily            0.000000
Census_MDC2FormFactor          0.000000
MachineIdentifier              0.000000
Length: 83, dtype: float64
```

We can easily see from above that features named as ”PuaMode ” and ’Census_ProcessorClass’ consists of more than 99% of missing values. Hence, we dropped those features from the dataset under consideration.

*1) Removing the Skewed Features:* First we determined the skewness among the features.



```
skewed_df = pd.DataFrame([{'column': c, 'uniq': train_df[c].nunique(),
                           'skewness': train_df[c].value_counts(normalize=True).values[0] * 100}
                           for c in train_df.columns])
skewed_df = skewed_df.sort_values('skewness', ascending=False)
skewed_df
```

| | column | uniq | skewness |
|---|---|---|---|
| **75** | Census_IsWIMBootEnabled | 2 | 99.999969 |
| **5** | IsBeta | 2 | 99.999249 |
| **69** | Census_IsFlightsDisabled | 2 | 99.998996 |
| **68** | Census_IsFlightingInternal | 2 | 99.998612 |
| **27** | AutoSampleOptIn | 2 | 99.997108 |
| **...** | ... | ... | ... |
| **4** | AvSigVersion | 8531 | 1.146861 |
| **14** | CityIdentifier | 107366 | 1.102969 |
| **73** | Census_FirmwareVersionIdentifier | 50494 | 1.022799 |
| **44** | Census_SystemVolumeTotalCapacity | 536848 | 0.586324 |
| **0** | MachineIdentifier | 8921483 | 0.000011 |

83 rows × 3 columns

Now we can easily see that there are 15 columns which are highly skewed i.e have more than 98% skewness ,so we removed them from our dataset.

*2) Filling the Missing Values:* First, we determined the features with missing values which are of numeric type.

```
for label, content in train_df.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            print(label)     #checking how many of the labels are there with
                             # numeric content and empyty values
```

```
RtpStateBitfield
DefaultBrowsersIdentifier
AVProductStatesIdentifier
AVProductsInstalled
AVProductsEnabled
CityIdentifier
OrganizationIdentifier
GeoNameIdentifier
IsProtected
IeVerIdentifier
Firewall
Census_OEMNameIdentifier
Census_OEMModelIdentifier
Census_ProcessorCoreCount
Census_ProcessorManufacturerIdentifier
Census_ProcessorModelIdentifier
Census_PrimaryDiskTotalCapacity
Census_SystemVolumeTotalCapacity
Census_TotalPhysicalRAM
Census_InternalPrimaryDiagonalDisplaySizeInInches
Census_InternalPrimaryDisplayResolutionHorizontal
Census_InternalPrimaryDisplayResolutionVertical
Census_InternalBatteryNumberOfCharges
Census_OSInstallLanguageIdentifier
Census_FirmwareManufacturerIdentifier
Census_FirmwareVersionIdentifier
Census_IsAlwaysOnAlwaysConnectedCapable
Wdft_IsGamer
```

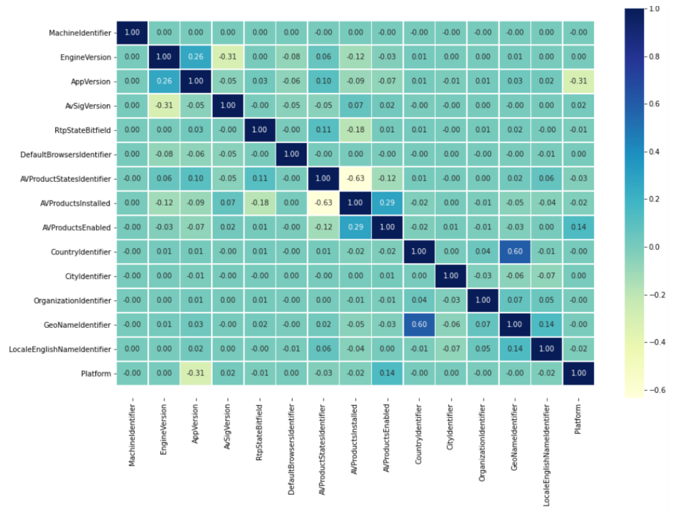
Then, we replaced the missing values with median of that column.

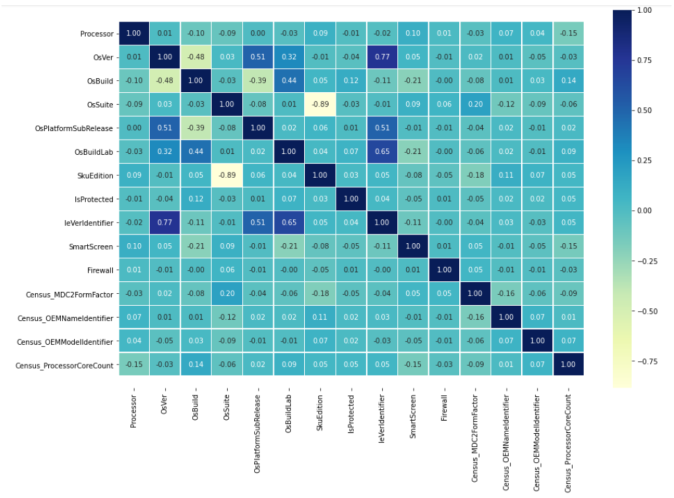
*3) Encoding categorical features:* First, we determined the features which were of category type

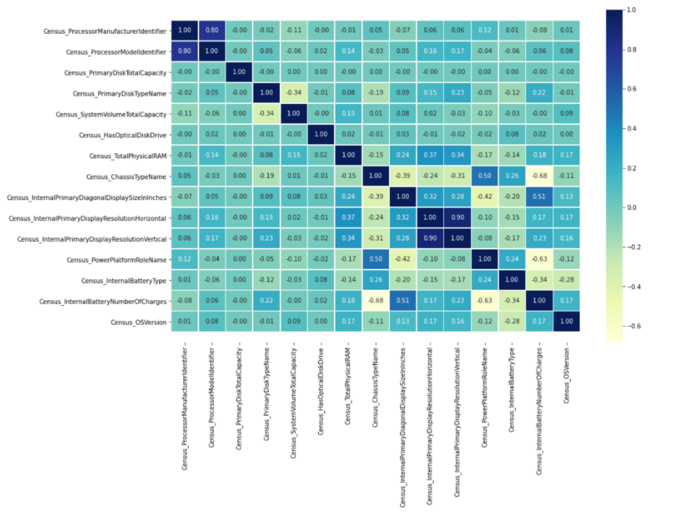
```
for col, val in train_df.items():
  if pd.api.types.is_numeric_dtype(val):
    train_df[col] = val.fillna(val.median()) #imputing the missing values with median of that column
  else:
    train_df[col] = train_df[col].cat.codes #encoding the categorical datatypes
```

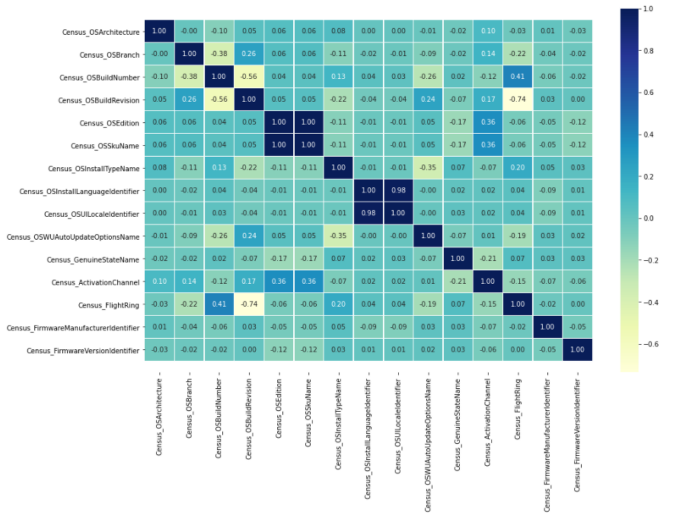
Then, we used one-hot encoding technique to encode those features

*4) Elimination of highly correlated features:* Since dataset is too large its not a good idea to see all the correlation of all features at once,we will break the correlations into size of 15 features. The Correlation Matrices obtained are as follows:









Among all the correlation matrices above we dropped the columns which have higher correlation than 90%. Hence we dropped the following features

1) 'Census_OSInstallLanguageIdentifier'
2) 'Census_InternalPrimaryDisplayResolutionVertical'
3) 'Census_OSSkuName'

### D. Test-Train Split

Since the dataset is quite large we take a relatively small portion of the dataset to test and train the model. We have taken 10000 entries to lower the computational time. Then we have split this data into training and testing (For validation) dataset.

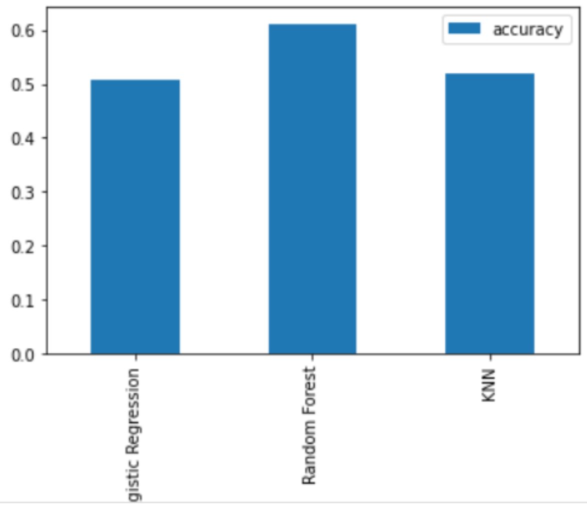
### E. Training the model

For making predictions we have used the SciKit Learn implementaions of the models[1]

- Logistic Regression
- KNN
- Random Forest Classifier

We then compared the accuracy of the three models used.

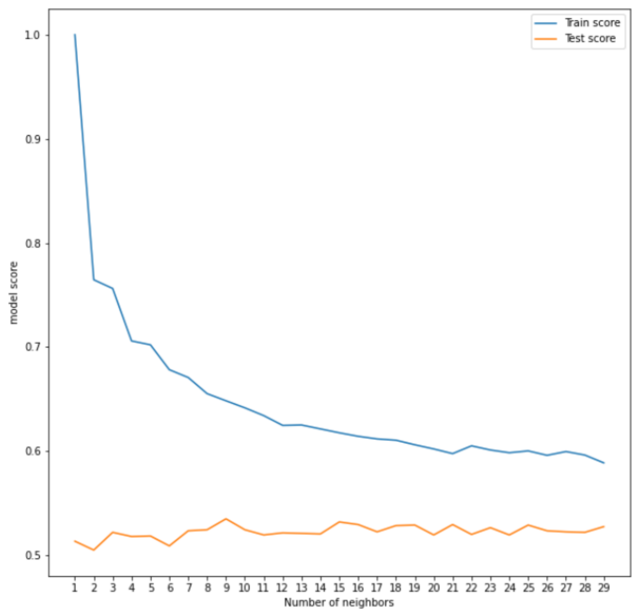
```
model_compare.T.plot.bar();
```



### F. Hyperparameter Tuning

*1) KNN:* We varied the value of the hyperparamter K from 1 to 30 and observed the accuracy on the testing data.

[1]The implementation code has been included in the Jupyter notebook. The notebook link has been mentioned in references.



We can see that we get the highest accuracy on test data for K=9. The obtained accuracy is as follows.

```
best_efficiency_in_train_scores
```

```
1.0
```

```
best_efficiency_in_test_scores
```

```
0.535
```

From the obtained results we can easily see that KNN model is overfitting and the reason is that because of excessive number of data points.

*2) Logistic Regression:* We used Randomized Search CV for varying the following hyperparameters of Logistic Regression.

```
log_reg_grid = {"C": np.logspace(-4, 4, 25),
                "solver": ["liblinear"]}
rf_grid = {"n_estimators": np.arange(10, 1100, 50),
           "max_depth": [None, 3, 5, 11],
           "min_samples_split": np.arange(2, 26, 2),
           "min_samples_leaf": np.arange(1, 26, 2)}
```

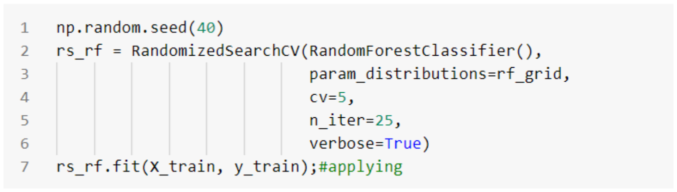
The best hyperparameters obtained are

```
rs_log_reg.best_params_#best parameters of logisstic regression
```

```
{'C': 46.41588833612773, 'solver': 'liblinear'}
```

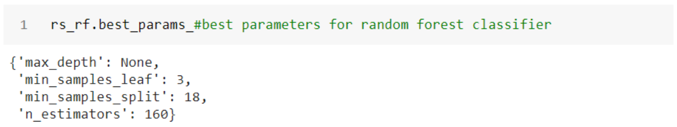
For these hyperparameters we get an accuracy of 52.25%

*3) Random Forest Classifier:* We used Randomized Search CV for varying the following hyperparameters of Random Forest Classifier.



```
np.random.seed(40)
rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                           param_distributions=rf_grid,
                           cv=5,
                           n_iter=25,
                           verbose=True)
rs_rf.fit(X_train, y_train);#applying
```

The best hyperparameters obtained are



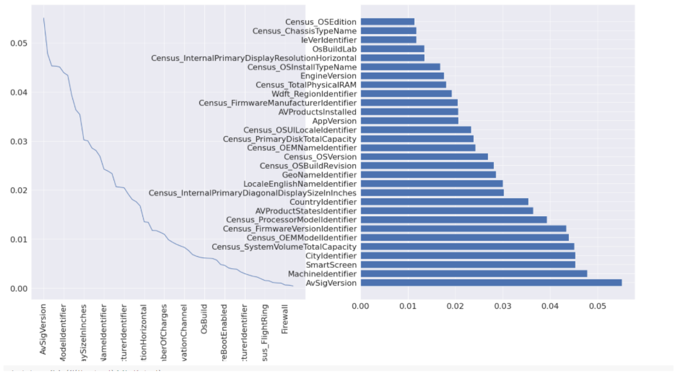
```
rs_rf.best_params_#best parameters for random forest classifier
```

```
{'max_depth': None,
 'min_samples_leaf': 3,
 'min_samples_split': 18,
 'n_estimators': 160}
```

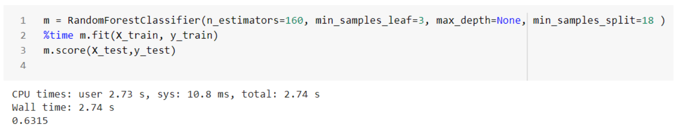
For these hyperparameters we get an accuracy of 62.80%

## G. *Feature Engineering and Training Model with important Features*

We obtained the importance of the various features. The plot of it is as follows.



We decided to keep only the features with a score greater than 0.01. Using these features we again trained a Random Forest Classifier using the best hyperparameters as obtained earlier.



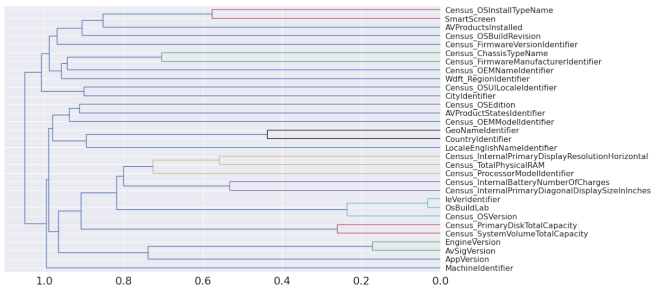
```
m = RandomForestClassifier(n_estimators=160, min_samples_leaf=3, max_depth=None, min_samples_split=18 )
%time m.fit(X_train, y_train)
m.score(X_test,y_test)

```

```
CPU times: user 2.73 s, sys: 10.8 ms, total: 2.74 s
Wall time: 2.74 s
0.6315
```

Hence, by feature engineering our accuracy on test data went up from 62.80% to 63.15%.

## H. *Correlation among the features*

Finally we plotted a dendogram to obtain the correlation among the various features.



## I. *Using LGBM (Light Gradient Boosting Machine) model for Predictions*

We used the data under consideration after feature engineering to train the LGBM model. We used Light GBM because it is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms.
Finally we obtained a prediction accuracy of **60.55**% at a runtime of $468\mu s$.

## V. Conclusions

- Whenever we choose different machine learning models for carrying out training and testing, for similar model complexity there is a tradeoff. Some models offer higher prediction accuracy but consume higher computuational resources whereas some models give lower prediction accuracy but at a much lower time.
- This tradeoff can also be observed in the project. The comparision of run-time and prediction accuracy of the models under comparison is as below.

| Model | Runtime | Prediction Accuracy |
|---|---|---|
| Logistic Regression | $72.3\mu s$ | 52.25% |
| KNN | $276\mu s$ | 53.75% |
| Random Forest | $2.83s$ | 63.15% |
| LGBM | $468\mu s$ | 60.55% |

TABLE I
Comparison of runtime and prediction accuracy for the algorithm

- We can easily see that LGBM model comparatively works very well on this dataset. It takes very less runtime compared to Random Forest, however its prediction accuracy is comparable to that of Random Forest.
- We obtained a prediction accuracy of 63.15% in making predictions.
- Concluded that the feature AVSig Version (which is the version of the database of signatures of the anti virus) is the most important feature in determining if the machine is likely to face a malware attack.

## VI. Limitations

- Firstly, due to limited computational resources we worked on a very small sample from the dataset. Hence, we can expect the prediction accuracy to be higher when we use the complete dataset with all the models. The highest prediction accuracy obtained in the competition was roughly $69\%$ using LGBM, however, they used more number of samples from the data set and hence higher computational time.
- Due to time constraints of the project, we could not compare the Neural Networks model. However, we did expect LGBM to work better on the given dataset and hence we skipped Neural Networks and directly moved on to LGBM.
- With some prior knowledge of Operating Systems and Computer Architecture, we could have dropped some of the less important features by mere observation, which couldn't be done in this project.

## VII. Work Contribution

| Task | Aniket | Vinit |
|---|---|---|
| Project Conception | 2.5 hours | 2.5 hours |
| Data Collection | - | 1 hour |
| Coding | 10.5 hours | 4.5 hours |
| Report | 4.5 hours | 9.5 hours |
| Getting introduced LGBM Algorithm | 0.5 hours | 0.5 hours |
| Slides and Video Presentation | 2 hours | 2 hours |

## References

[1] https://colab.research.google.com/drive/1Q7pwCLvbUV4G1kgtcy8RWJveq2-XkCYE?usp=sharing

[2] https://www.kaggle.com/c/microsoft-malware-prediction/data?select=test.csv

[3] https://lightgbm.readthedocs.io/en/latest