Microsoft Malware Prediction

Rathin Bhargava
IIIT-Bangalore
IMT2017522

Ram Srinivasa Bharathy
IIIT-Bangalore
IMT2017521

***Abstract*—As the number of computing devices increase, especially in the case of IOT, so does the need for security. There are more and more devices which are vulnerable to malware attacks. With the help of machine learning, we would like to predict whether a computer system will get affected by malware given its specifications. In this particular project, the dataset has been provided by Microsoft.**

***Index Terms*—Microsoft, malware prediction, machine learning, AdaBoost**

I. Introduction

The goal of this project is to predict a Windows machine's probability of getting infected by various families of malware, based on different properties of that machine. The telemetry data containing these properties and the machine infections was generated by combining heartbeat and threat reports collected by Microsoft's endpoint protection solution, Windows Defender.

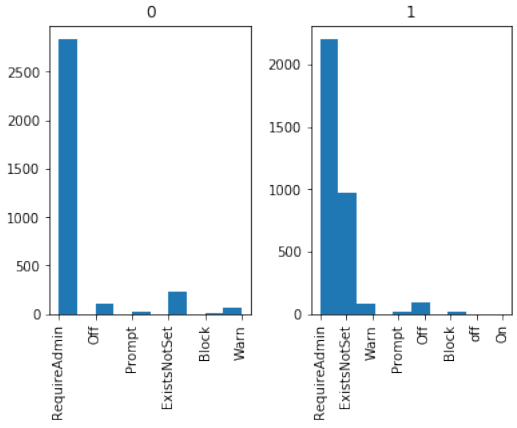
The link for the model's pickle file and the dataset has been uploaded on this google drive link.

II. Exploratory Data Analysis

A. Column Analysis

Let us analyse a few columns in the data set which we intuitively believe might be relevant to detecting malware.

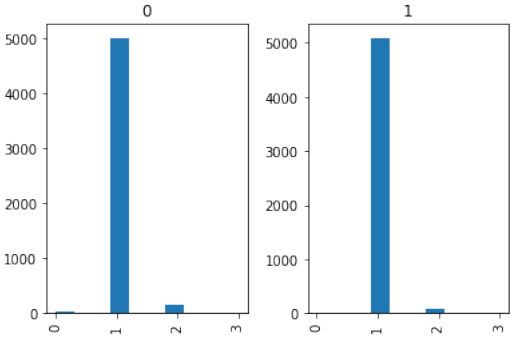
1) SmartScreen: Let us plot a histogram of $SmartScreen$ grouped by its $HasDetections$ values.



As we can see, the $ExistsNotSet$ value is much higher in comparison for 1 than for 0. Also, in general all values in the domain show much more presence for 1 than 0 suggesting that this feature must have some relevance to $HasDetections$.

B. AVProductsEnabled

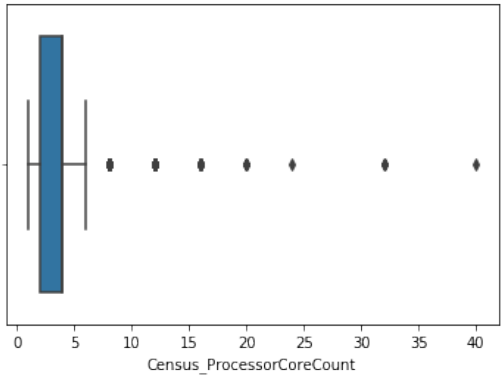
Let us now plot a histogram of $AVProductsEnabled$ groupy by $HasDetections$ values.



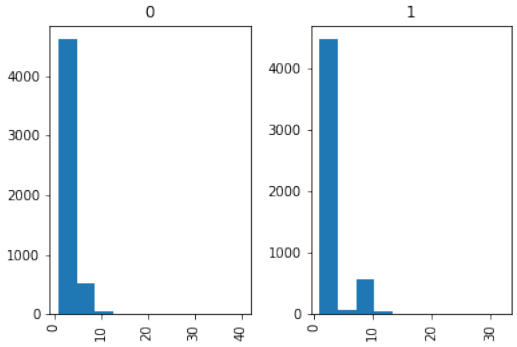
This feature however doesn't seem to be very useful as the values seem to be equally distributed across 0 and 1.

C. Census_ProcessorCoreCount

Let us now analyse a numerical feature such as $Census\_ProcessorCoreCount$. Let us draw a box plot for this values.



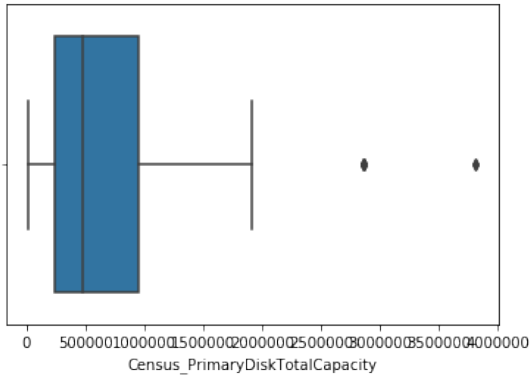
We can see that majority of the computers have 0-5 cores, but there are a few which have a much higher core count. This can be attributed to the fact that they are special kinds of computers with a large number of cores. Now, if we group these values by $HasDetections$, we get

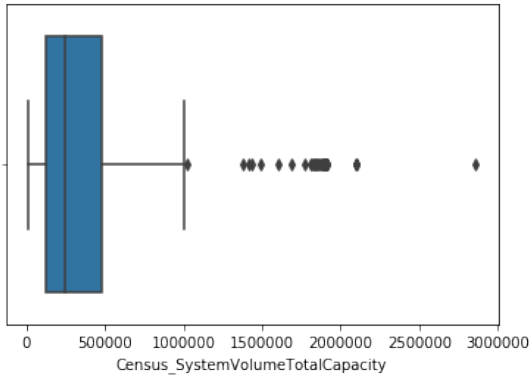


This data also doesn't seem to directly correlate with $HasDetections$ in a meaningful way.

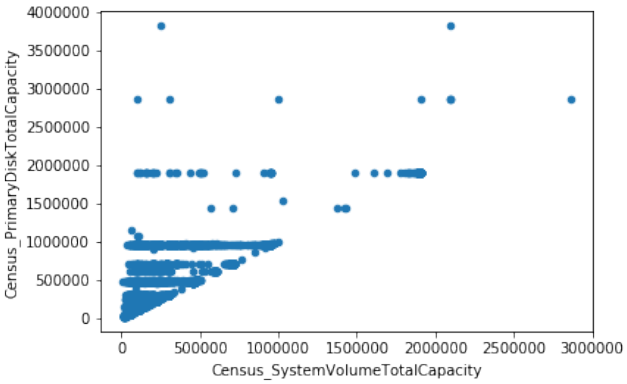
D. Census_SystemVolumeTotalCapacity and Census_PrimaryDiskTotalCapacity

Let us draw the box plots for both of these features.





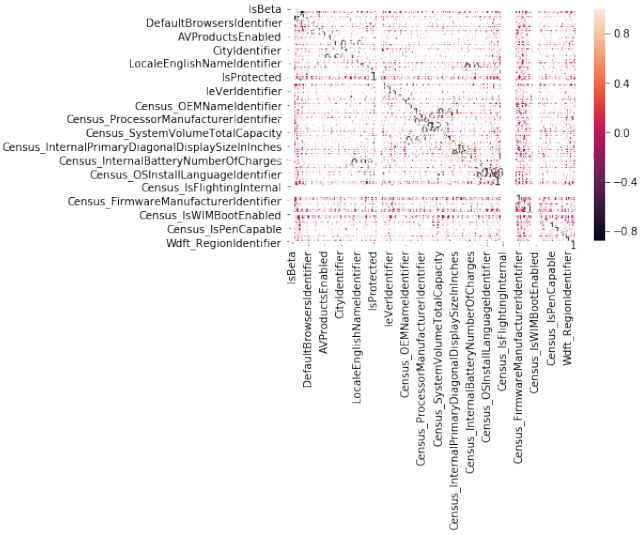
Let us try to draw some correlations across these two features by drawing a scatter plot between them.



There is an important point to observe here. We expect the Primary Disk capacity to be lesser than or equal to the Total Disk capacity for all computers.
This is because the Total Disk capacity should contain the Primary Disk as well.
When we observe our scatter plot, we see that there are no points below the line $x = y$. This proves that all the data points, at least with respect to these two features, seem to be valid points.

E. Heat Map

Provided below is the Heat Map of all the features in the data set.



III. Cross Feature Analysis

Here, we talk about which features which are similar to each other and which ones depend on each other.

One of the key ideas we will be using here is that if the null values of a few columns coincide, the probability of that happening is exponentially low. Hence, the only other conclusion is that the columns are related in some way, and a new feature could be made out of the two, if necessary.

1) If we look at the 3 antivirus product columns, $AVProductStatesIdentifier$, $AVProductsInstalled$ and $AVProductsEnabled$ we can see that each of them have 36 null values that coincide. There is a high correlation between these 3 columns.

2) If we look at $IsProtected$,there are 37 null values, 36 of which coincide with $AVProductStatesIdentifier$. This makes sense, since $IsProtected$ is the derived field made from Spynet's $AVProduct's$ columns.
To process this column, we have 2 choices here, keep $IsProtected$, trust this derived field, or just keep all 4 columns and let the model do all the work.

3) If we look at $Wdft\_IsGamer$ and $Wdft\_RegionIdentifier$, there are 325 null values which coincide with each other. It's an awful coincide that this happens, unless these 2 features are positively correlated.

IV. Data Processing and Feature Engineering

A close look at our data reveals that we have $\sim 10,000$ rows and 83 columns. There were a lot of challenges associated with the data preprocessing like handling null values, categorical data and columns with duplicate data. Here, we show how we dealt with each of these challenges.

A. Data Cleaning

There were some invalid values, like $UNKNOWN$ and $Unspecified$ in various columns which were converted to nan before any other data preprocessing. The column $Census\_OsEdition$ has the same values as $Census\_OSSkuName$. So, we deleted the column $Census\_OSSkuName$.

B. Handling columns with null values

We deleted the following columns because more than 30% of the values were null. Imputing these values in any form would just result in biasing the data. Also, it didn't seem like these columns had any relation with the other columns, so we couldn't apply any model to figure out these values based on the other values.

1) DefaultBrowsersIdentifier
2) OrganizationIdentifier
3) PuaMode
4) Census_ProcessorClass
5) Census_InternalBatteryType
6) Census_IsFlightingInternal
7) Census_ThresholdOptIn
8) Census_IsWIMBootEnabled

The following columns were deleted because they didn't convey a lot of information. The values here were either 0 or null.

1) IsBeta
2) AutoSampleOptIn
3) SMode
4) Census_IsFlightsDisabled
5) Census_IsVirtualDevice

On a side note, all the other columns which have been deleted are the ones which have been one hot encoded or have derived features.

The following columns were taken care of by imputing values.

1) RtpStateBitfield
2) AVProductStatesIdentifier
3) AVProductsInstalled
4) AVProductsEnabled
5) IsProtected
6) Firewall
7) UacLuaenable
8) Census_OEMNameIdentifier
9) Census_OEMModelIdentifier
10) Census_ProcessorCoreCount
11) Census_ProcessorManufacturerIdentifier
12) Census_ProcessorModelIdentifier
13) Census_PrimaryDiskTotalCapacity
14) Census_PrimaryDiskTypeName
15) Census_TotalPhysicalRAM
16) Census_InternalBatteryNumberOfCharges
17) Census_OSInstallLanguageIdentifier
18) Census_GenuineStateName
19) Census_FirmwareManufacturerIdentifier
20) Census_FirmwareVersionIdentifier
21) Census_IsAlwaysOnAlwaysConnectedCapable
22) Census_IsVirtualDevice
23) Wdft_IsGamer
24) Wdft_RegionIdentifier

There were a few ways we could impute these values. Imputing with the mean did not make sense, because these values are discrete and specific. They are model numbers, identification numbers, etc. Median doesn't make much sense either, since most of the columns have no ordering per se. However, mode seems to be a good choice for imputing the columns with specific numeric data. This is because given the data, the probability that a null value is the mode is higher than it being for any other value. Here, we illustrate 3 different strategies for imputing the missing values.

1) Use the mode for imputing null values for every column
2) For categorical data, impute the null values with -1 so that it can be encoded later as a different value altogether. For numeric data, use the mode.
3) Sample from the inverse probability distribution function. By doing so, the imputed values obey the underlying probability distribution which inherently biases the data the least. This method can be used irrespective of the kind of data - Categorical or numeric. The steps to do this is as follows:
    a) Set an order to all the unique elements in the domain
    b) Calculate the probabilities of each of them
    c) Take the cumulative sum of all the probabilities
    d) Generate a random number from $[0, 1]$
    e) Find the index in the cumulative sum array which is just less than the random number.
    f) Choose that value as your sample.
    g) Repeat this k times if you want k samples.

h) It can be proved that uniformly sampling the cumulative distribution array samples the original array according to its underlying probability distribution.

This must be used in caution, however. Because this is a randomized process, there will be variations in the results on running the same data again and again.

We have use the third method to impute all the null values. This is done because the column's are roughly independent.

The column $Census\_SystemVolumeTotalCapacity$ was imputed based on $Census\_PrimaryDiskTotalCapacity$. Since there was a high correlation of the system disk being equal to the primary disk (there were no other disks), we made the same assumption for all the values which were missing from the column $Census\_SystemVolumeTotalCapacity$.

C. Handling Categorical data

We have one hot encoded all columns with categorical data where the size of the domain wasn't very high. After a little bit of tweaking, we empirically arrived at the desired domain size, 22. Any more would result in too many columns, any less would lose out on a lot of data. After one hot encoding these columns, the original column is deleted, lest it jeopardize the results.

The other categorical columns were then label encoded. Label encoding essentially gives every value a random number. There were some columns which while categorical, had a certain order to them. Label encoding these columns would lose the inherent 'order' they possess. Hence, we decided to customize the label encoding for them.

We now explain the customized label encodings, or feature engineering.

1) Versions: All the version numbers were of the form $a.b.c.d$. This format was uniform throughout the five version columns - $AvSigVersion$, $EngineVersion$, $AppVersion$, $Census\_OSVersion$ and $OsVer$. This format essentially meant $major.minor[.build[.revision]]$.
We made a basic assumption that new update is better than it's predecessors i.e 2.0.1.0 is better than 2.0.0.999. To establish such an ordering, we converted the string into a number by multiplying the major, minor, build and revision numbers with the appropriate powers of 10 and adding them up. For example, 1.2345.678.9 gets converted to 12345006780009. Note that the revision number - 9, gets padded with zeroes. This 'padding' number has been uniquely identified for each column and has been taken into consideration while calculating the number.

There is another column called $OsBuild$. Most of the build values of $OsVer$ were 0, and hence it was assumed that $OsVer$ did not have the complete information. By incorporating $OsBuild$ as the build numbers of $OsVer$, we made a new feature called $OsVer\_Build$.

Because of this format, we noticed that the columns $Census\_OSBuildNumber$ and $Census\_OS-BuildRevision$ are useless as this information is incorporated in the column $Census\_OSVersion$.

2) PlatformUpdate: The domain of this attribute is $\{windows7,\ windows8,\ windows10,\ windows\ 2016\}$. This is in the order of its release. We assumed that the security of the system increases as one updates their OS. Hence, we gave them the encoding $1, \ldots, 4$

3) OsPlatformSubRelease_Order:
The domain of this attribute is $\{windows7,\ windows8.1,\ th1,\ th2,\ rs1,\ rs2,\ rs3,\ rs4,\ prers5\}$. The th stands for $Windows\ 10\ Threshold$ and rs stands for $Windows\ 10\ Redstone$. The above values are listed in the order of their releases. Our assumption is that as windows got updated, their security became tighter and tighter. So, we label encoded the domain from $1, \ldots, 9$ in the order mentioned above.

4) Census_OSBranch: The domain of this attribute is $\{th1,\ th1\_st1,\ th2\_release,\ th2\_release\_sec,\ rs\_prerelease,\ rs\_prerelease\_flt,\ rs1\_release,\ rs2\_release,\ rs3\_release,\ rs3\_release\_svc\_escrow,\ rs3\_release\_svc\_escrow\_im,\ rs4\_release, rs5\_release\}$. These are $Windows\ 10$ updates in the order of their releases. Again, we assumed that later the release, the more secure the system and hence have label encoded this domain form $1, \ldots, 13$ in the order mentioned above.

V. Models

A. Logistic Regression

Logistic Regression is a statistical model that in its basic form uses a logistic function. shows as follows,

$$f(x) = \frac{1}{1 + e^{-k(x-x_0)}}$$

to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression is estimating the parameters of a logistic model(a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labelled "0" and "1". The model itself simply models the probability of output in terms of input, and does not perform classification, though by making use of a threshold in the probability, it is generally made into a binary classifier.

B. AdaBoost

AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favour of those instances misclassified by previous classifiers. However, AdaBoost is sensitive to noisy data and outliers. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.

C. Running the models

We tried running both the models above. AdaBoost performed significantly better than Logistic Regression. One plausible explanation is that AdaBoost has non linear decision boundaries, so can generalize better.

Logistic Regression accuracy: 50.15984654731458%

AdaBoost is a randomized algorithm which takes in a seed. For demonstration purposes, we have taken the seed value to be 1. Some seed values increase our accuracy, while others decrease it.