

IST 718 M003 Project Design Report

Report prepared by:

* Pranav Radhakrishna
* Prankur Garg
* Tejas Patil
* Sarthak Tandon

**Malware Detection on Microsoft Systems**

Abstract

* **Brief Description of Dataset:**

This dataset contains 82 independent assessment parameters which are a mix of categorical and numerical type. The target variable ‘HasDetections’ is a binary attribute indicating whether that machine was infected by malware or not. Malware detection is inherently a time-series problem, but it is made complicated by the introduction of different kinds of machines, for e.g. new machines, machines that receive new operating systems, online-offline machines, etc.

Link to the dataset: [Microsoft Malware Detection Dataset](https://www.kaggle.com/c/microsoft-malware-prediction/data)

* **Business Problem:**

Malware is any software intentionally designed to cause damage to any computer, server, client or computer network. Malware exploits security defects in the design of operating system, in applications or in vulnerable versions of browser plugins. The dataset contains the information about different machines with windows operating systems and their attributes that will help in determining the likelihood of that machine getting infected by malware. For preventing such instances happening in future, it is utmost important to find out whether the computer is infected by malware and resolve the issue accordingly.

* **Data Science Methodology:**

1. The training dataset here contains more than 8.9 million instances. Hence, we will be using stratified sampling to pick a sample that will represent the entire dataset. We have sampled about 700,000 rows for the final dataset.
2. Exploratory data analysis was performed by statistically and numerically analyzing the data using data visualizations. This involves a combination of univariate and multivariate analysis.
3. Data cleaning was done by dealing with missing values, outliers and imbalanced categorical attributes in the dataset.
4. The hold out method was used to evaluate the models in order to ensure that the models do not overfit. In order to achieve this, we split the data into training and testing sets.
5. Since this is a binary classification problem, several classification algorithms were used, this includes Gradient Boosting, Random Forests, Logistic Regression, Decision Trees and Naïve Bayes.
6. Wherever possible, the grid search technique was used in conjunction with K-Fold Cross validation to tune the hyperparameters in order to optimize the models.
7. The models were evaluated based on the Area Under the Curve (AUC) metric.

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 reports collected by Microsoft's endpoint protection solution, Windows Defender. Each row of the data set corresponds to a machine. Since there are 82+1 columns in the dataset, we will describe the most important columns (based on feature Importance) here for better understanding.

|  |  |
| --- | --- |
| **Column** | **Description** |
| MachineIdentifier | Unique Identifier for the machines |
| HasDetections | The target variable for the data. It indicates that Malware was detected on the machine |
| EngineVersion | Windows Defender Engine Version |
| ProductName | Windows Defender Version Name |
| AVProductStatesIdentifier | ID of the Antivirus Software’s Configuration |
| CountryIdentifier | ID of the Country where the machine is located |
| Processor | CPU of the system |
| OsVer | Operating System of the System |
| Firewall | Boolean value to determine if Windows firewall is enabled |
| Census\_TotalPhysicalRAM | Total RAM in the corresponding machine |
| Census\_PrimaryDiskTotalCapacity | Total disk space in the machine in MBs |
| SmartScreen | SmartScreen enabled string value from registry |
| Census\_ProcessorCoreCount | Number of logical cores in the processor |
| Census\_InternalPrimaryDiagonalDisplaySizeInInches | Diagonal length in inches of the primary display |
| Census\_InternalPrimaryDisplayResolutionVertical | Number of pixels in the horizontal direction of the display |
| Census\_InternalPrimaryDisplayResolutionHorizontal | Number of pixels in the vertical direction of the display |
| Census\_ChassisTypeName | Chassis type of the machine |
| Census\_OSInstallTypeName | Install methodology used in the machine |
| Wdft\_IsGamer | If the device is a gamer device or not |

The original dataset had over 8.9 million rows. Stratified sampling was performed to reduce the size of the data down to 700,000 rows. The stratified sampling method was chosen because it allows us to retain the distribution of the target variable in the original dataset.

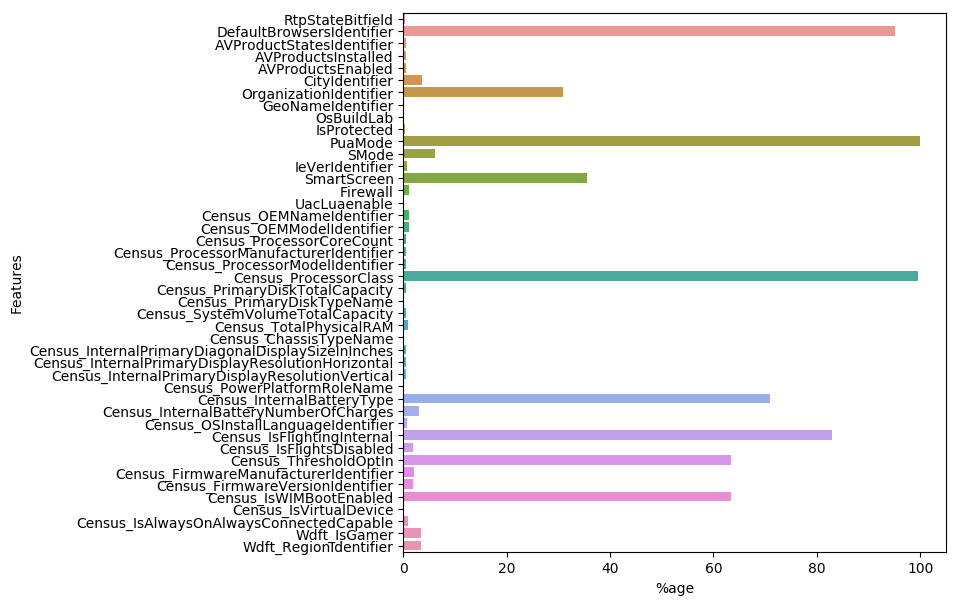
Here are certain observations about the dataset:

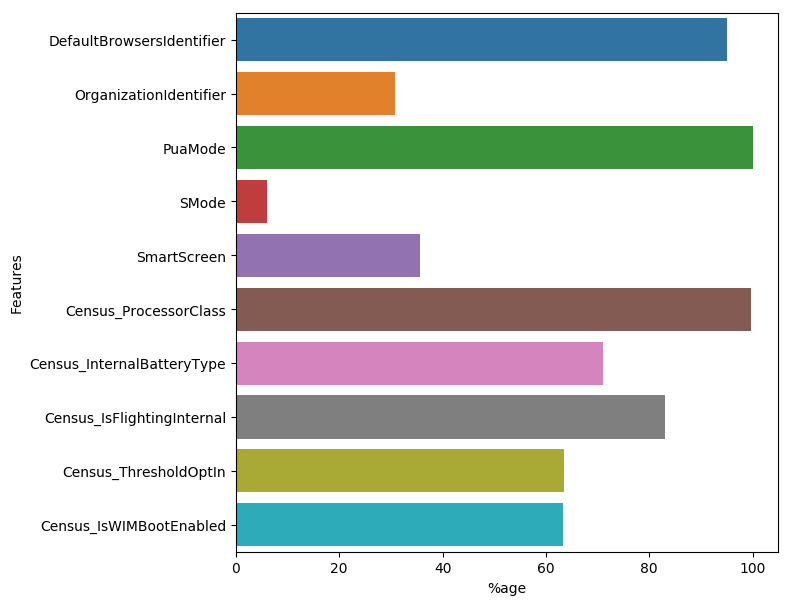
1. The target variable is balanced with equal distribution among the target variable.

A picture containing screenshot

Description generated with very high confidence

1. Out of the 82+1 columns, 44 features had missing values with 10 features missing more than 5%.





1. Majority of the features are categorical, with 69 of the 83 of them. The remaining 14 are numerical.

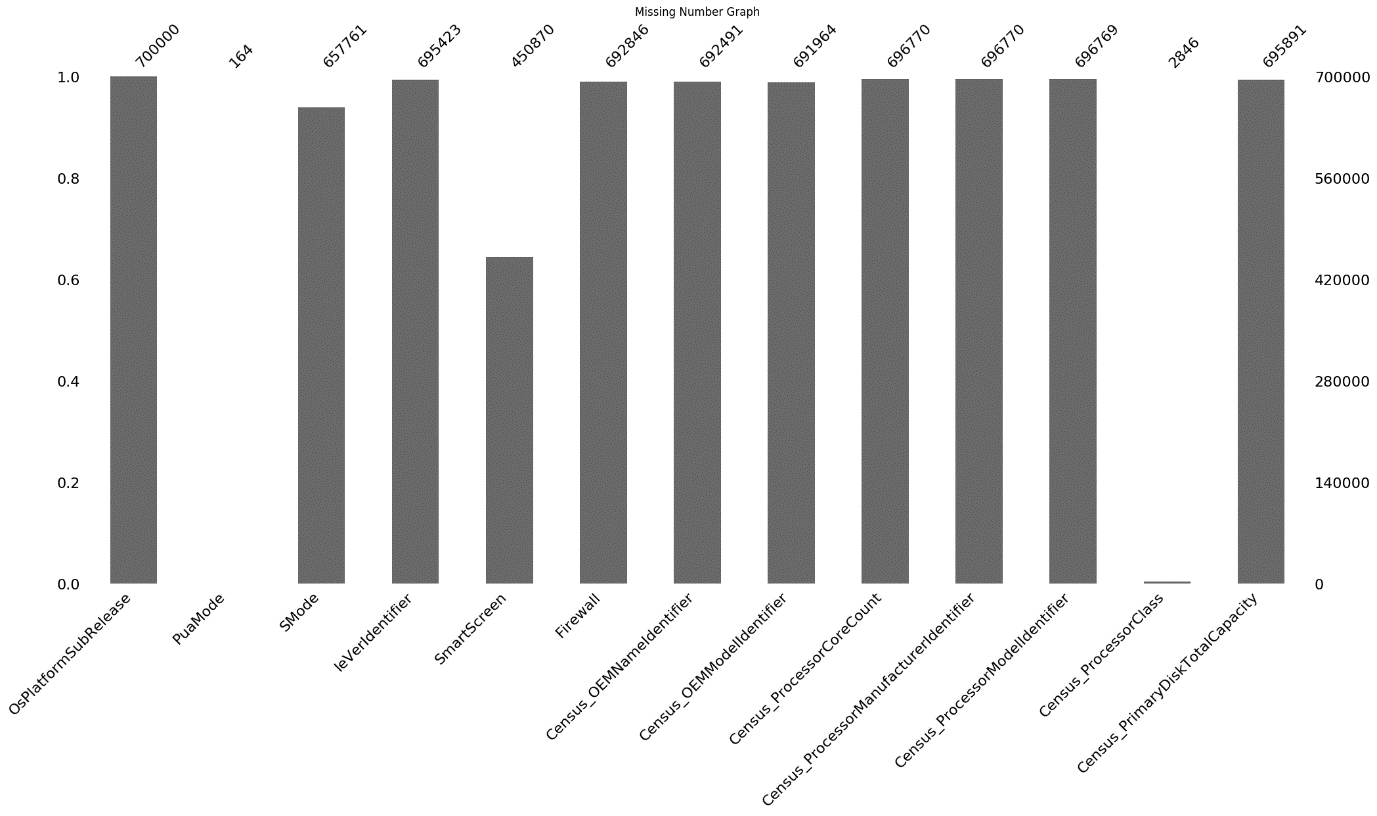
Methodology

# Data Preprocessing:

Before feeding the models with the data to make predictions, extensive preprocessing done. The following exercises were undertaken:

* **Removing columns with >50% missing values and Columns with single value:**

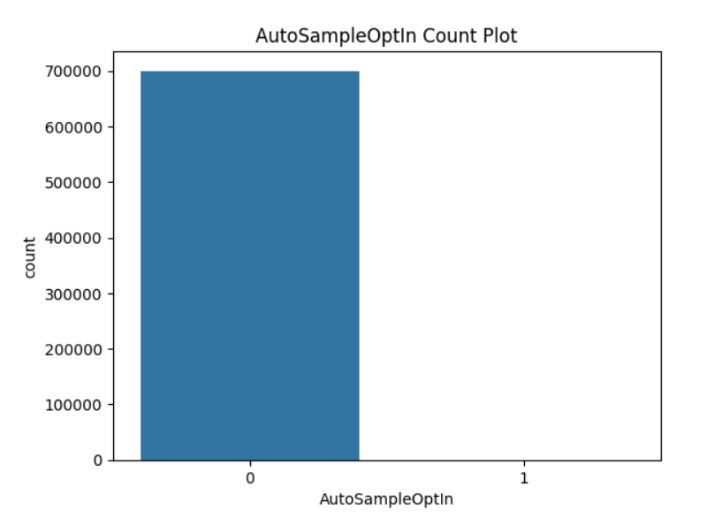
The visual below shows the fraction to which certain columns in the dataset are filled, with the amount showcased by the height of the black bar.



As you can see The attribute Census\_ProcessorClass consists almost entirely of missing values and can be dropped. Similarly, columns with over fifty percent missing values were dropped. Some columns which were dropped along with the percentage missing values are:

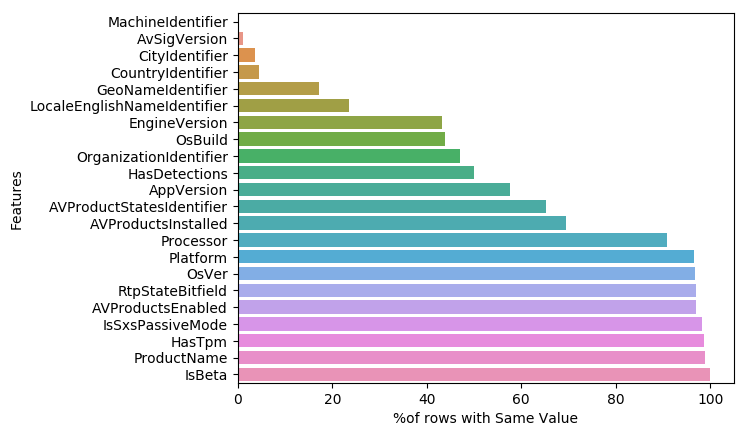
* Census\_IsFlightingInternal 82.90%
* Census\_ThresholdOptIn 63.40%
* Census\_IsWIMBootEnabled 63.80%

On the diagram at the bottom, you can see that the column AutoSampleOptin has only one value in all its columns. As a result, his column is of now use to train the model. Hence, we dropped this any other column with the same characteristics.



* **Removing columns with >90% similar values:**

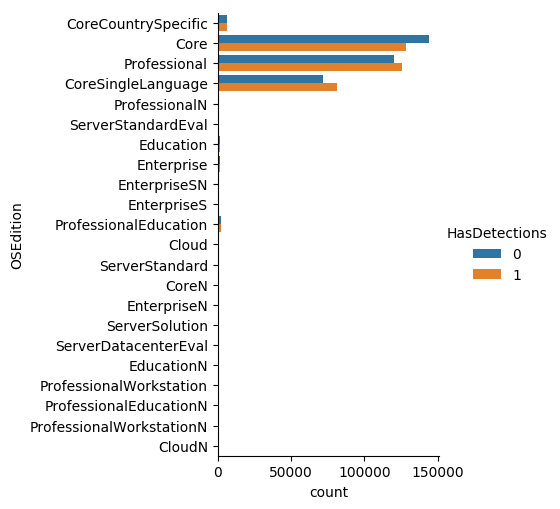
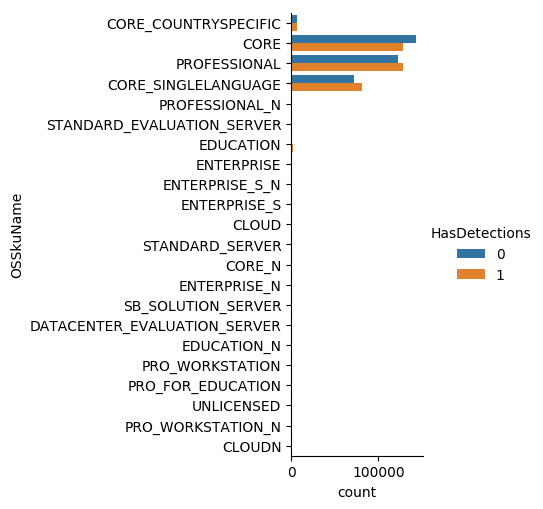
The next step was to drop columns with more than 90% similar values. This was done since having these columns will confuse the models. We want to make sure that the features in the data set are as balanced as possible.



In the Chart you can see the percentage of the total rows which the similar value corresponding for that column. For example, 99% of the rows have one value in the IsBeta feature. Hence, we removed IsBeta and other similar columns. Some other columns which were dropped are:

* Census\_IsFlightsDisabled 98.10%
* Census\_FlightRing 93.67%
* Census\_IsVirtualDevice 99.10%
* Census\_IsPenCapable' 96.20%
* Census\_IsAlwaysOnAlwaysConnectedCapable 93.50%
* **Removing duplicate columns:**

Variables such as Census\_OSEdition and Census\_OSSkuName, had same categories and the distribution of those categories with respect to the target variables was also very similar. Hence, we decided to remove one of the two variables, in this case Census\_OSSkuName.

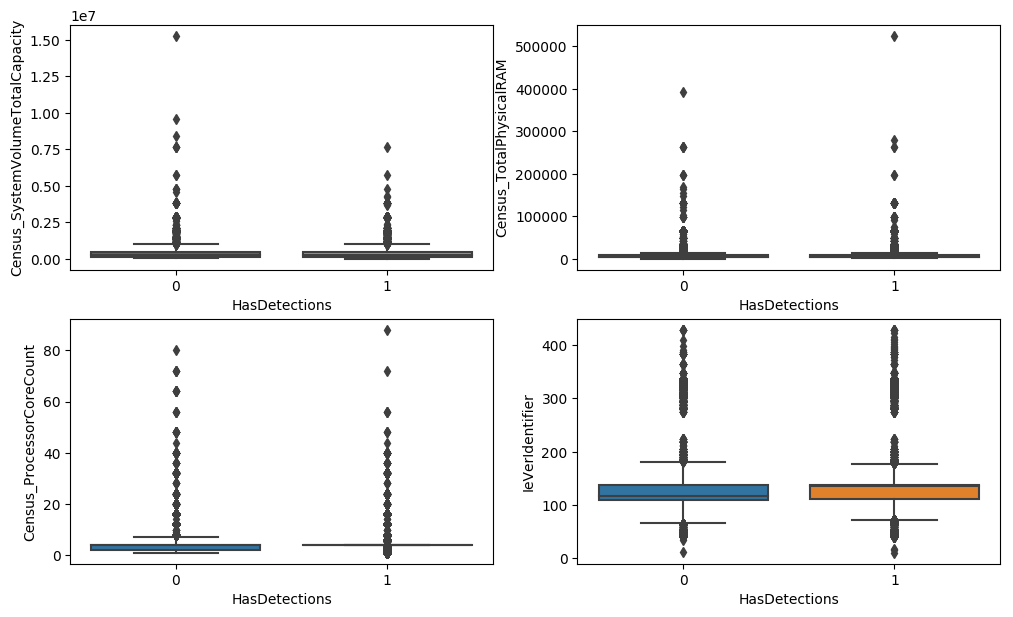


* **Removing rows with < 5% missing values:**

Once all these columns were removed, we decided to remove rows which had less and 5% missing values.

* **Imputing Missing Values and Removing Outliers:**

We then went ahead with imputing the missing values. For all the missing values that needs to be imputed, measures of central tendencies are the best estimates. Hence, for categorical variables, we used mode of the column (category with maximum frequency count), to impute the missing values. For missing values in numerical variables, we first checked the distribution of those variables. An observation was made that most of the numerical variables does not follow a normal distribution and instead has a skewed distribution. So, we used the median values of that variable to impute the missing values.



Data visualizations revealed the presence of outliers in many of the continuous attributes. As you can see on the left, these 4 numerical variables’ boxplot show outliers. The presence of outliers is undesirable as they represent noise in the dataset and can lead to misclassifications in the machine learning model. The outliers were handled by winsorizing them to the 95th or 99th percentile based on domain knowledge.

* **Aligning Data types:**

Certain columns in our data looked numerical but were categorical. Hence were aligned all the data types to reflect the nature of the features in the data.

* **Grouping Categories:**

A screenshot of a social media post

Description generated with very high confidenceAnother issue which we had to deal with was having a lot of categories in a column but only a few predominating in the data, while other categories remained in a handful of rows. In the graph on the right, we a feature named OrganizationIdentifier, which has 44 unique categories, but as you can see, two categories predominate. Since the other categories will not be as useful for the model while training, we decided to group the rest of the categories into one category name ‘others.

Once we were done with all these steps, were down to 655319 rows and 56 columns.

# Running Models:

Before we ran any model, we decided to split the data to training and testing, keeping a standard ratio of 70:30 respectively. We decided to run 5 models on our dataset. They are as follows:

1. **Naïve Bayes Classifier:**

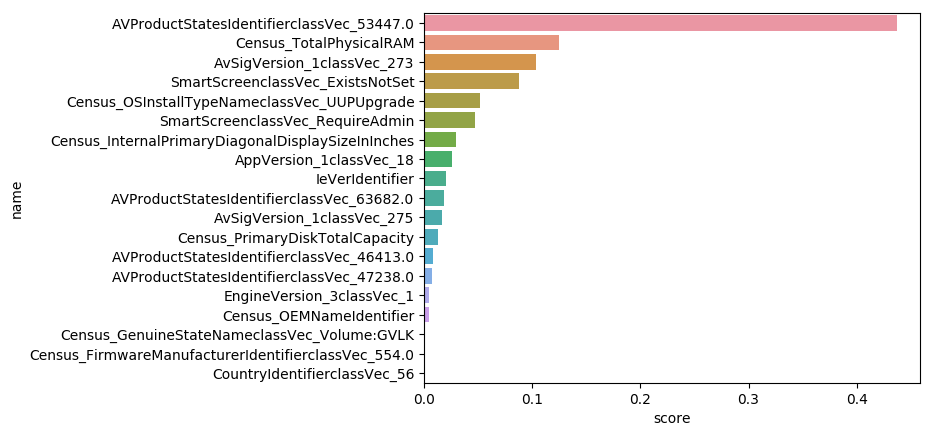
Naïve Bayes is a classification technique based on Bayes’ Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a feature in a class is unrelated to the presence of any other feature.

1. **Decision Tree Classifier:**

Decision tree is a versatile Machine Learning Model that can perform classification, both binary and multioutput. They are also the fundamental components of Random Forest, which we will see later in the report. The main purpose of the running decision tree models are as follows:

* **To get the base performance to compare future tree-based classifiers.**
* **To determine feature importance so that non-essential features can be removed which will reduce data and runtime.**

As a result, we decided to run the model on base hyper parameters. The following figure illustrates the most important features in the model:



As you can see, AVProductStatesIdentifier and RAM play a significant role in determining the susceptibility of a machine getting infected.

1. **Logistic Regression Classifier:**

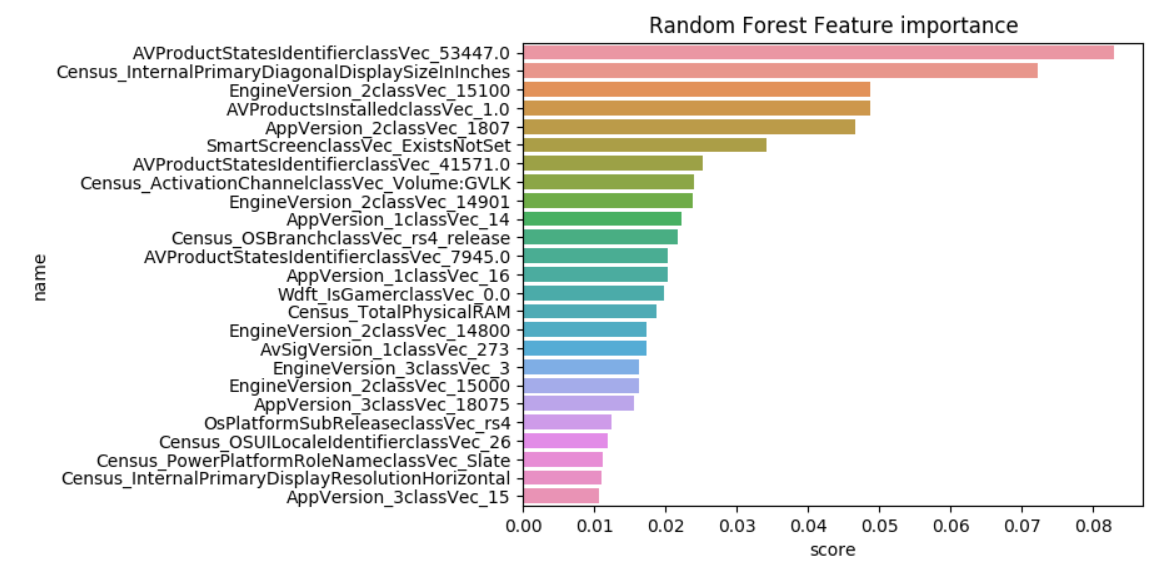
Using the variable importance from decision tree, we sampled the data for features having non-zero values of feature importance. We then used the reduced data to run a logistic regression model. Like linear regression, logistic regression also performs a linear combination of variables and their weights for prediction. Since, the target variable for logistic regression is a binary class variable or a multinomial class variable, the output of regression must be passed through an activation function. Generally, for binary classification we use sigmoid function, whose output is in the range of 0 to 1. Hence, Logistic regression uses a linear hyperplane for classification, by setting a threshold value to the output of sigmoid function.

First, we ran a base model of logistic regression, using the default parameters that are set by spark. It was observed that time required for model to run was very high. We know that regularization adds a penalty term to the cost function and reduces the weights of the variables, but only L1 regularization can make weights of the variables zero. Since, we already had the reduced set of features, we do not want to make the weights of variables completely zero. Hence, we set smaller values for alpha so that L2 regularization will be dominant over L1. The time required for running the cross validation along with the grid search for regularization hyperparameters was very high. We saw cluster getting detached multiple number of times. Hence, we decided not to perform grid search and manually find the best values for regularization parameter and elastic net regularization parameter.

1. **Random Forest Classifier:**

It is an ensemble of Decision Trees, which are trained via the bagging method. Random Forest searches for the best features among the subset of features which are random. The hyperparameters tuned were maxDepth, featureSubsetStrategy and numTrees. Initially, a grid search was attempted but due to the large volume of data, the DataBricks environment could not successfully execute the grid search. Therefore, the parameters had to be tuned manually. The maxDepth parameter was selected because allowing the trees to grow to a greater depth leads to low bias. The numTrees parameter allows us to reduce variance by increasing the number of trees. The featureSubsetStrategy parameter indicates the number of features to be considered for splits at each tree node, this helps lower the model’s generalization error.

The following figure illustrates the most important features in the random forest model:

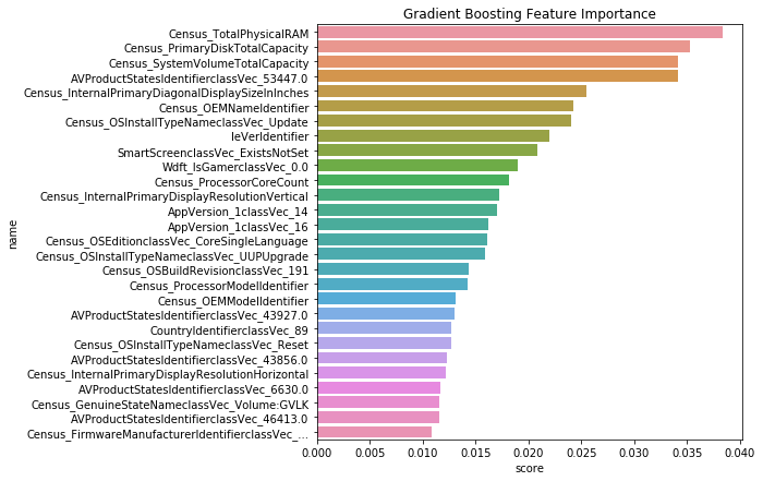


The feature importance plot shows that attributes such as “AVProductStatesIdentifier” and “AppVersion” both of which hold information about the antivirus software used by the machine prove to be important, this goes to show the importance of selecting a good antivirus software.

1. **Gradient Boosting Classifier:**

Another popular algorithm is the Gradient Boosting algorithm. It is a sequential algorithm that keeps adding predictors to the ensemble model with each one correcting its predecessor. The difference lies in the fact that the model tries to fit the new predictors to the residual errors made by the previous predictor. Due to limitations on databricks, we were unable to run the model on the entire data. As a result, the data was sliced and reduced to make sure the system does not crash.

The following figure illustrates the most important features in the Gradient boosting model.

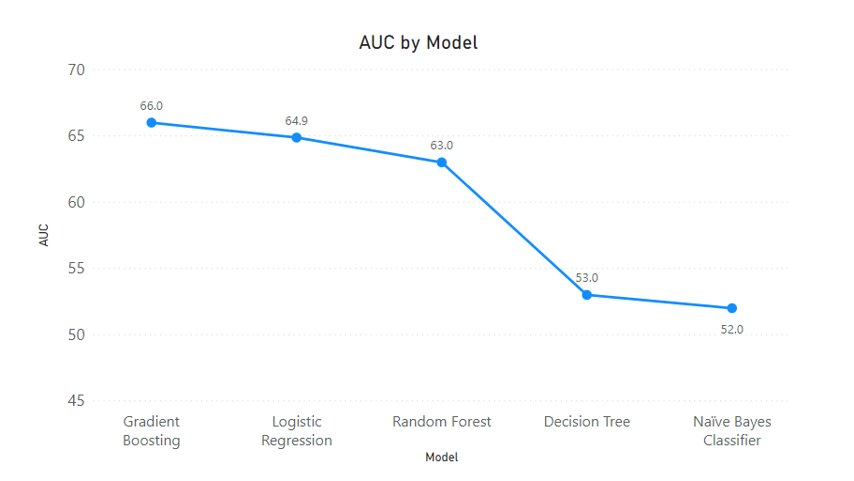


As you can observe, the feature importance for Random forest and Gradient boosting are almost identical, with a few changes in the order.

Result

Before discussing the results, we had to finalize our evaluation metric. We wanted to make sure that the models were able to differentiate between the two target variables. Hence it was decided to have the evaluation metric as AUC.

Here is the graph comparing the performances of all the models



The Naïve Bayes model was the worst performing model with the AUC score as 52.6%. The Base model for the Decision Tree did not fare a lot better with the score of 53%. However, we can agree on the fact that more complex model will give better performance than these.

For Logistic Regression, The AUC score for base model extracted from the summary was 63.8%. On the other hand, the AUC score calculated by transforming the test data was calculated to be 64.9%.

The Random Forest model produced an AUC of 63.3% on the test data. In order to ensure that the model is not overfitting the performance of the model on the test dataset was compared to the performance on the training dataset. The training dataset produced an AUC of 63.7% which is very close to the test value. Therefore, we can conclude that the model is not overfitting.

However, the best model turned out to be the Gradient Boosting model, with an AUC score of 66%.

It might be though that AUC score in the 60s might not be ideal, these values are in line with the best models submitted on Kaggle for the same data.

Conclusion

We conclude that we have successfully worked on a big data project, while performing all the operations using spark session. A stratified sample was taken with respect to the target variable. Data was cleaned, preprocessed and made ready for the machine learning models. The problem statement was to identify if the machine will be affected by the malware or not. We successfully built 5 machine learning models including Naïve Bayes classifier, Decision Tree model, Random Forest classifier, Logistic Regression model and Gradient Boosting machine model to classify the machine as either positive or negative for getting infected by malware.

We realized the problems that we had to face while working with a big data and the techniques to counter these issues. The size of the data makes it difficult to process the data cleaning operations and model building on our local machine. We even experienced the cluster getting detached multiple number of times. While working on this project, we also learned how the work should be distributed among the team, to ensure everyone is actively involved in the work and is aware of changes, if any.

Finally, talking about the AUC score, we observed that the highest AUC we got, was for the Gradient Boosting model. The AUC score obtained were not a great figure but still much better than a random model prediction. The AUC score obtained are still comparable to the range of best performing notebooks in Kaggle competition.

The future scope for this project lies in pre-processing of the data. We believe that the categories can be better grouped together using more detailed analysis and combining the domain knowledge. Also, when using variable importance for feature selection, we need to be careful about discarding the variables that might be useful for classification.