MGT 6203 Final Report - Team 64

Pengcheng Chen *(903648701, pchen369)*, Mohammed Kashif Uddin Ansari *(903661810, mansari44)*, Sourav Sarkar *(903756468, ssarkar83)*, Prashant Durgayya Kudrigi *(903566073, pkudrigi3)*, Sandeep Arisetti *(903454629, sarisetti3)*

***Abstract*—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.**

I. Introduction

THE malware industry continues to be a well-organized, well-funded market dedicated to evading traditional security measures[1]. Once a computer is infected by malware, criminals can hurt consumers and enterprises in many ways. Malware can get onto your device when you open any attachments, files or any scam/virus infected website. With more than one billion enterprise and consumer customers, Microsoft takes this problem very seriously and is deeply invested in improving security.

Therefore, we are trying to develop machine learning models to predict if a machine will soon be hit with malware. We are considering a dataset that is provided by Microsoft, which is available on the Kaggle website.

II. Project Objective

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 provided in the dataset.

III. Data Exploration And Preparation

A. Data Down-sampling

The original Microsoft malware detection training dataset has around 8 million data points and the size of the training set is 4 GB. This dataset is too big to process in our personal computer, so we decided to down-sample the data to something we can work with. The final dataset is randomly sampled using Python default random module, and it has $200,000$ data points in total.

B. Data Exploration

The dataset has 82 independent variables and 1 dependent variable. As illustrated in bar chart in Figure 1, the dependent variable 'HasDetections' has almost equal instances of Detected(1) vs Not detected(0) which indicates that the dataset is balanced. Thus, there's no need to perform up-sampling techniques or giving extra weight to one value of the target variable while training the model

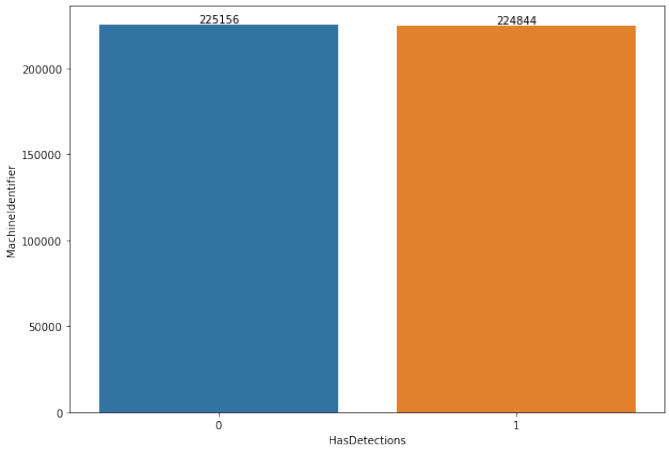

Fig. 1: Distribution of dependent variable, which is 'HasDetections'. This shows that we have a balanced dataset.

For 82 independent variables, 36 are in float format, 16 are in integer format, 30 are in object (string) format. However, we can not simply treat 52 numerical variables are as numerical, since most of them are IDs for system parameters or identifiers. As a result, we created our own definition of categorical and numerical variables based on the description of each variable. Based on our analysis, only 8 variables will be treated as numerical, the rest will all be treated as categorical. We also observed a lot of missing values in the dataset, and we will discuss how we handle the missing value in the next sub-section. In addition, it is also discovered that most of the categorical variables has large number of unique values, the fourth sub-section will cover what technique we utilize to group and encode these values.

C. Missing Values Handling

One of the biggest challenges for this dataset is missing values. There are 44 independent variables with missing values. 6 of them are numerical variables and 38 of them are categorical variables based on our modified data type definition. For all 6 numerical variables, the proportion of missing value is less than 5%. So we can use mean imputing to impute all of them except for "Census_InternalBatteryNumberOfCharges". For this variable, more than half of non-missing values are zero, we decide to try impute it with zero instead of mean. In term of categorical variables, we use mode to impute missing value if mode makes up greater or equal than half of the values and the missing value is less than 5%. For the rest of the categorical variables, we simply impute them with "missing"

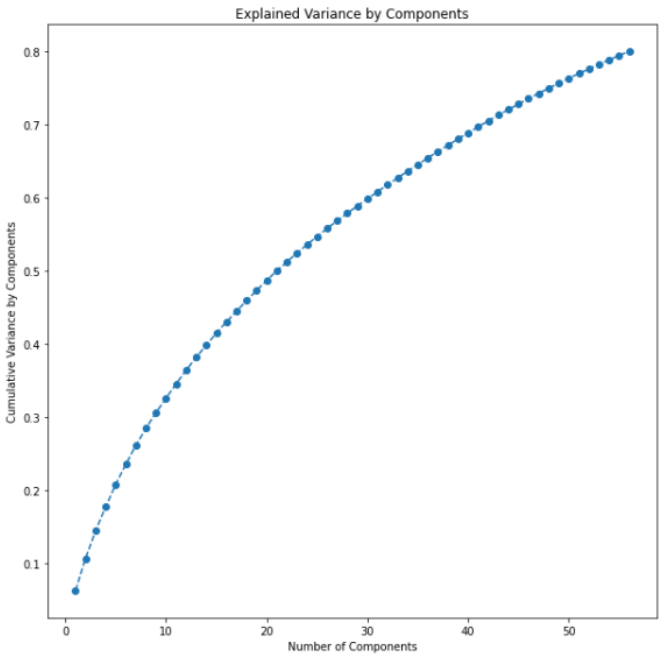

Fig. 2: Principal Component Analysis - Cumulative variance chart

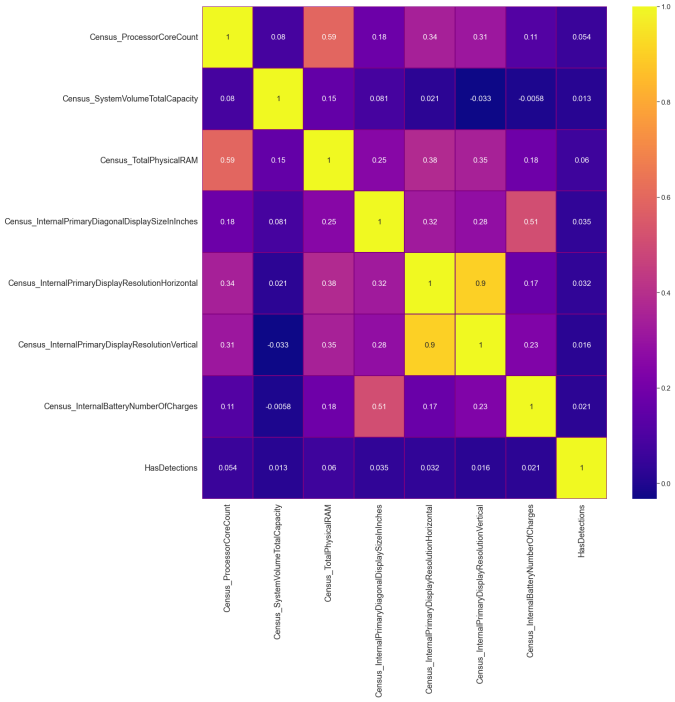

Fig. 3: Correlation Matrix: The relationship between the independent variables and dependent variables of Machine Malware dataset

string. The imputation was completed in Python with our custom imputation class.

D. *Data Encoding*

As mentioned in the previous paragraphs, the dataset has 74 categorical variables based on our own definition. Each of the variables could have hundreds of unique values, which makes it impossible to one-hot encode them directly. As a result, we implemented a Quantile-Other-OneHot Encoder class using python. For each categorical variable, the class will only keep values with high frequencies up until certain configurable quantile m, where m is between 0 and 1. For the rest of the values, the class will replace them with "other" string. For cases where the number of high frequency values are more than configurable parameter k, where k is an integer. The class will only cap the number of high frequency values at k and group the rest as "other". As such, each variable can have at most $k + 1$ values. Finally, we will one-hot encode all the transformed categorical variables. The binning before one-hot encoding was completed in Python with our custom encoder class.

E. *Data Transformation*

The dataset has 8 true numerical and 56 categorical variables. Since there are many categorical variables are in the dataset, we converted all the categorical variable to numerical variable using one-hot encoding technique. We got 212 binary variables after the conversion. For dimensionality reduction of the features, we decided to apply Principal Component Analysis (PCA). We used python for all our exploratory data analysis and data transformation. We've scaled the dataset before applying PCA. We used PCA from sklearn decomposition package for fit and transformation. Based on the cumulative variance chart, we decided to take .80 variance to use in the model. We got 56 PC components after considering .80 cumulative variance.

F. *Feature Selection*

Other than using PCA type of problem, we also tried using other methods such as correlation matrix and independent-vs-dependent plot.

For numerical variables, we ran a correlation heat-map, as shown in Figure 3, to identify the correlations between each pair of numerical variable. For each pair of variables of high correlation we will fit a logistic regression and then remove the less significant one. Besides correlation analysis for numerical variables, we also check the relationship between each numerical variables and the target variable. As shown in the top 2 charts in Figure 4, we plot the histogram for independent variable for $HasDetections = 0$ and $HasDetections = 1$, and observe the difference in pattern of 2 histograms. If the pattern are different, this means this independent variable is likely a good predictor, otherwise, it may be bad predictor. After checking both correlations matrix and histograms, we decided to remove 4 numerical variables and keep the other 4.

For categorical variables, we also plot the independent variable against dependent variable using stacked bar chart as illustrated in the bottom 3 charts in Figure 4. The X-Axis represent the unique values of categorical variable after binning. The blue part of the bar represent $HasDetections = 0$ and the orange part of the bar $HasDetections = 1$. If we see very

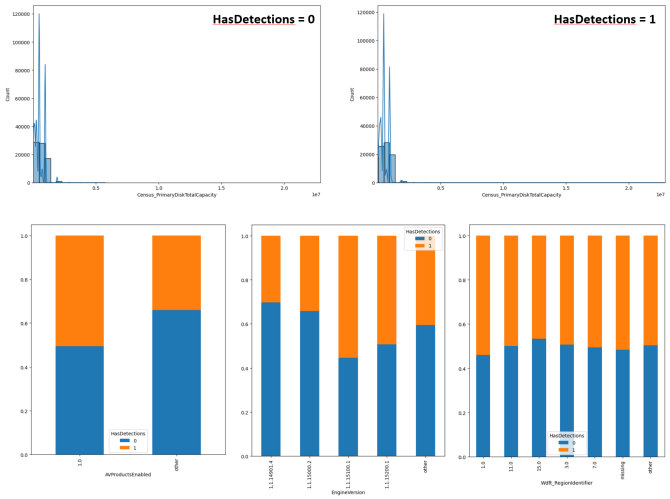

Fig. 4: Count plot for SmartScreen feature

different split of blue and range across different values, then this means this independent variable is likely a good predictor, and vice versa.

IV. Modeling Training and Performance Measurement

A. *Data Splitting*

We have split the data (75/25) into train and test datasets for evaluating the performance of the models that we are building. All data imputation, encoding, and feature engineering will be only based on the training set.

B. *Models/Algorithms*

Once the data is split into training and test data sets, we will train the following models with the training data set. For each model, the performance will be measured based on the following 4 metrics on testing set.

Accuracy: measures the proportion of values that the model classified correctly over the total number of data points that were tested.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

Precision: measures how good the model is when the prediction is positive.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall: measures how good our model is at correctly predicting positive classes.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

Area Under the Curve:is an aggregate metric that evaluates how well a classification model classifies positive and negative outcomes at all possible thresholds.

*1) **Logistic Regression**:* is a linear model and it performs well to reveal linear relationship within the data[2]. Outcome from Logistic Regression Model is categorical values like (Yes/No) which exactly is what we are trying to predict in this use case. As we use the Logistic function in logistic regression, it always gives a probability of being in a group. Probability value is always between 0 and 1. Logistic function can be written as follows:

$$p(x) = \frac{e^{b_0 + b_1 x}}{1 + e^{b_0 + b_1 x}} \tag{4}$$

Logistic function with several predictors can be written as follows:

$$\frac{p(x)}{1 - p(x)} = e^{b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + \ldots + + b_n x_n} \tag{5}$$

We fitted logistic regression model with all the predictors using glm function. It gave an error 'contrasts can be applied only to factors with 2 or more levels'. So, we removed all the predictors that has unique values in the test data set and then fitted the logistic regression model. It showed that Engine Version value of 1.1.15100.1 is significant in predicting the malware attack with a p-value $<$ 2e-16. Similarly AV Product States Identifier, AV Products Installed, Smart Screen, Census Is Virtual Device, Census Is Always On Always Connected Capable, whether the User Is Gamer, Region Identifier, Census Primary Disk Total Capacity and Census System Volume Total Capacity are also significant with a p-value $<$ 2 e-16.

As the return value of the model prediction is a probability of value between 0 and 1, we need to have a threshold to categorize the predicted probability into yes/no. we tried with different probability values like 0.2, 0.4, 0.5, 0.6, 08 and 0.9 to calculate the metrics like Accuracy, Precision and Recall using test data. Threshold value of 0.5 gave the best results.

Figure 5 shows the Confusion Matrix, Accuracy, Precision, Recall and ROC curve for Logistic Regression model. Accuracy, Precision and Recall are close to 63%. Though it is good, it may not be enough in this use case as missing to predict an attack might cost lot more than falsely predicting a machine is vulnerable when it is not really. So, Recall and AUC values should be improved.

*2) **Step-wise Binary Logistic Regression**:* As per [3], step-wise binary logistic regression model performed better compared to multi Linear regression, so we created step-wise regression (backward selection) using Step AIC where it starts with all the predictors and keep reducing the number of predictors until it finds the best performing model.

Step AIC model returned a model that used the predictors EngineVersion, AppVersion, RtpStateBitfield, IsSxsPassiveMode, DefaultBrowsersIdentifier, AVProductStatesIdentifier, AVProductsInstalled, AVProductsEnabled, CountryIdentifier, LocaleEnglishNameIdentifier, Processor, OsBuild, OsBuildLab, SkuEdition, AutoSampleOptIn, PuaMode, SMode, IeVerIdentifier, SmartScreen, Census MDC2FormFactor, Census OEMNameIdentifier, Census ProcessorModelIdentifier, Census ProcessorClass, Census HasOpticalDiskDrive, Census ChassisTypeName, Census InternalBatteryType, Census

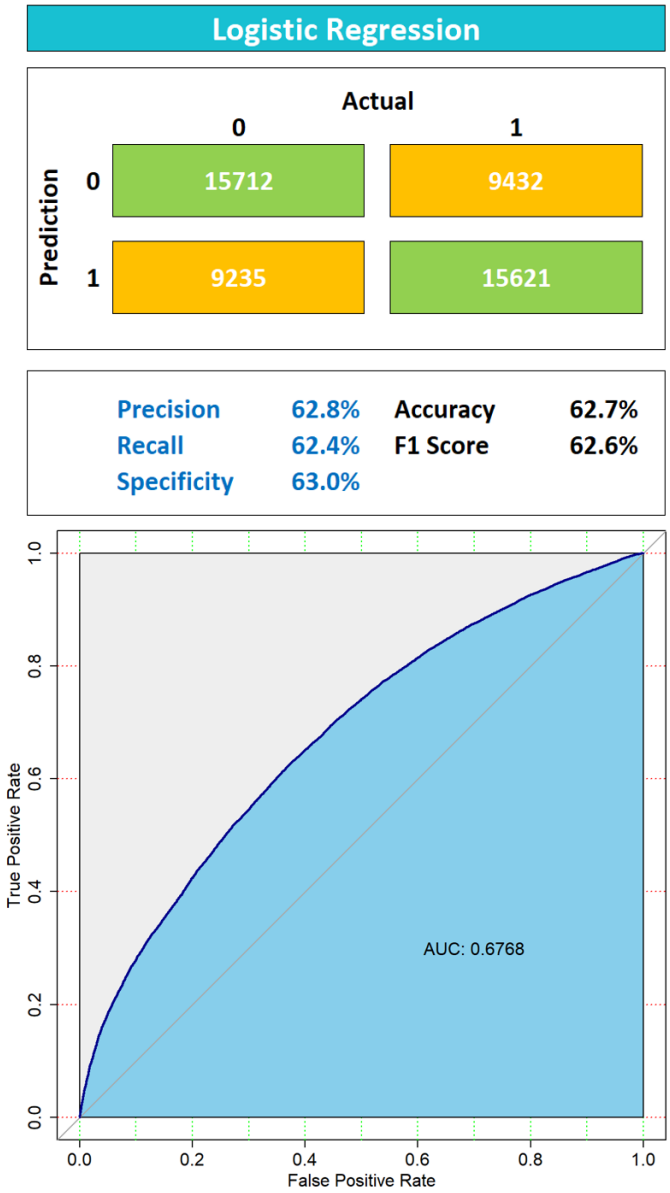

Fig. 5: Confusion Matrix and ROC Curve for Logistic Regression Model

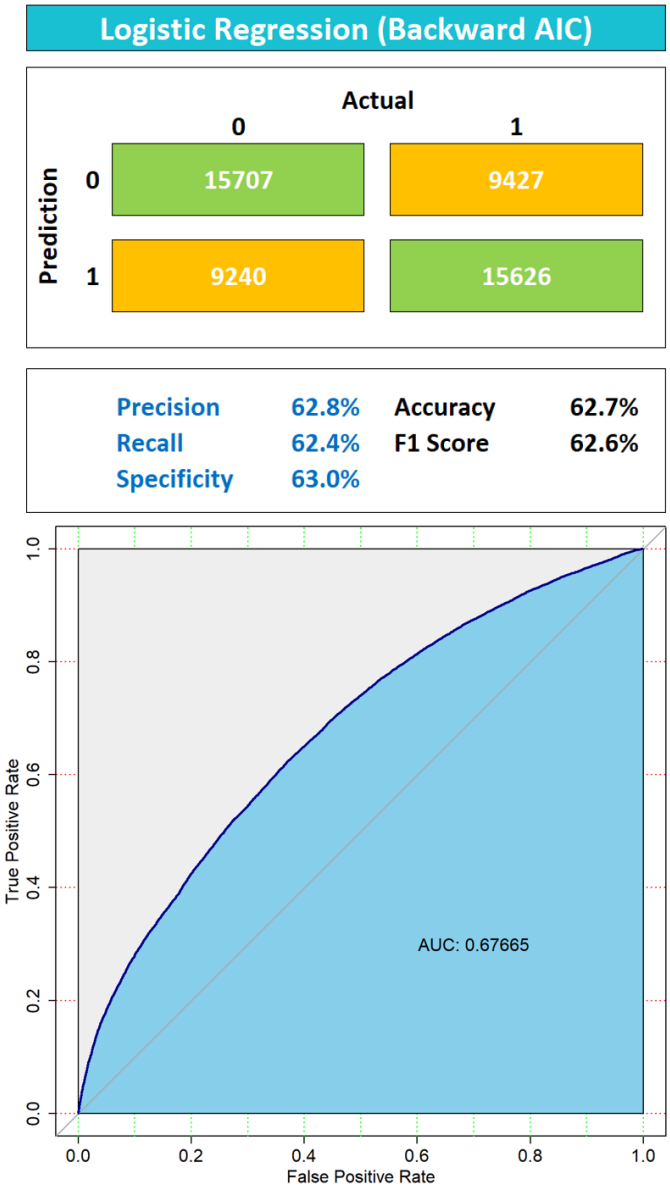

Fig. 6: Confusion Matrix and ROC Curve for Step AIC Logistic Regression Model

OSBuildNumber, Census OSBuildRevision, Census OSSkuName, Census OSInstallTypeName, Census OSInstallLanguageIdentifier, Census OSUILocaleIdentifier, Census OSWUAutoUpdateOptionsName, Census IsFlightsDisabled, Census FlightRing, Census FirmwareManufacturerIdentifier, Census IsWIMBootEnabled, Census IsVirtualDevice, Census IsTouchEnabled, Census IsPenCapable, Census IsAlwaysOnAlwaysConnectedCapable, Wdft IsGamer, Wdft RegionIdentifier, Census ProcessorCoreCount, Census PrimaryDiskTotalCapacity, Census SystemVolumeTotalCapacity, Census TotalPhysicalRAM.

Figure 6 shows the Confusion Matrix, Accuracy, Precision, Recall and ROC curve for Step AIC Logistic Regression model. The metrics are very similar to what we have got for the Logistic Regression model. This shows that the data analysis and feature selection methods we used are good. But, we can still try to find a better model.

*3) **Reduced Logistic Regression model**:* As a next step we wanted to try by removing some insignificant columns (p-value > 0.05). From the Step AIC model summary the fields that have high p-value are Locale English Name Identifier, Auto Sample Opt In, Census OEM Name Identifier, Census OS Install Language Identifier, Census OS UI Locale Identifier, Census Firmware Manufacturer Identifier. So, we created a reduced model by removing these fields from the Step AIC model.

Figure 7 shows the Confusion Matrix, Accuracy, Precision, Recall and ROC curve for Reduced Logistic Regression model. The metrics are very similar to what we have got for the Logistic Regression and Step AIC models. So, this model may

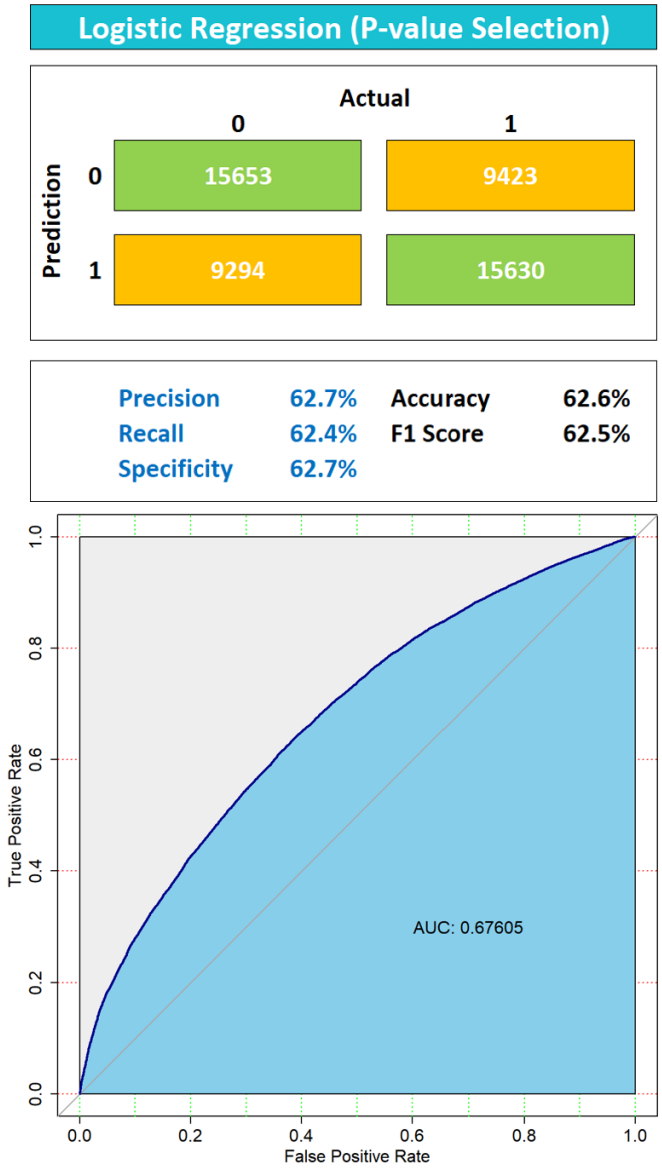

Fig. 7: Confusion Matrix and ROC Curve for Reduced Logistic Regression Model

not be of much use. So, we continued to explore K-Nearest Neighbors (KNN), Decision Tree and Ensemble models (Random Forest and xgboost).

4) ***K-Nearest Neighbors(KNN):*** KNN is the simplest non-parametric but effective classification method[5]. But it is slow as it has to find the 'k' nearest values for the given values of predictors. The data needs to be normalized. Another challenge with KNN model is to find the right k value that gives the best prediction. Generally, distance is measured in various distance measures. We will use Euclidean distance to find the best 'k' value. We will train the model with different values of k and choose a value that gives the best accuracy.

$$\text{Euclidean distance} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{6}$$

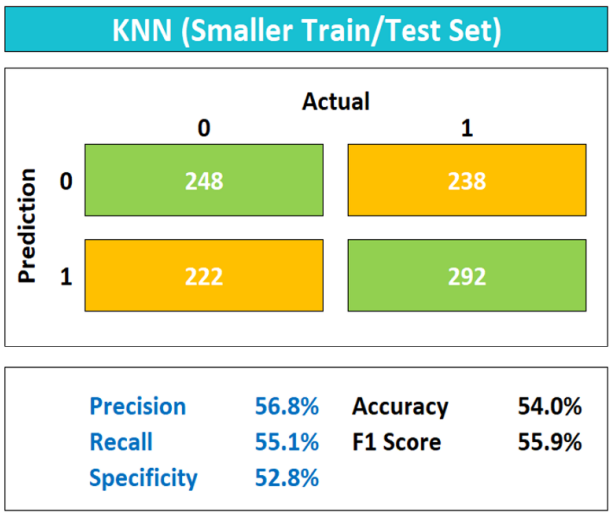

Fig. 8: KNN Confusion Matrix

It determine the class from nearest neighbor list. Take the majority vote of class labels among the k-nearest neighbors. Weigh the vote according to distance • weight factor, w = 1/d2

To fit KNN model, we need to convert all the categorical variables to numerical variables. We used dummyvar to convert all the categorical variables to numerical. We got 212 binary variables from the categorical variables. To reduce the features, we selected the features from Step AIC for fitting KNN model. Down sample the dataset because model was computationally inefficient, it was taking very high time for calculating the distance between a new point and each existing point. KNN model didn't perform well when compare to Logistics Regression and Step AIC models.

Figure 8 shows the Confusion Matrix, Accuracy, Precision, Recall for K-Nearest Neighbor model. Precision is slightly better than Accuracy (54%) and Recall (55.1%). It may not be good enough as an overall model performance.

5) ***Decision Trees:*** Decision Tree is a form of supervised machine learning model which takes the structure of a tree. It splits from the parent node on all available variables, then selects the split which results in the most homogeneous leaf-nodes. There are many methods to quantify the impurity of leaf nodes and decide which feature can create the best split. These methods includes Gini Impurity, Information Gain/Entropy, Chi-square, Variance (numerical variable) etc.

Decision trees are best suited for Classification problems, easy to understand visualize and interpret that has flowchart-like structure that helps decision-making. The model requires little data preparation compared to other techniques that often require data normalization, dummy variables need to be created and blank values to be removed. Decision tree however does not support missing values. Decision-tree learners can create over-complex trees that do not generalize the data well. This is called over fitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.Decision trees can be unstable because small variations in the data might result in a completely different tree

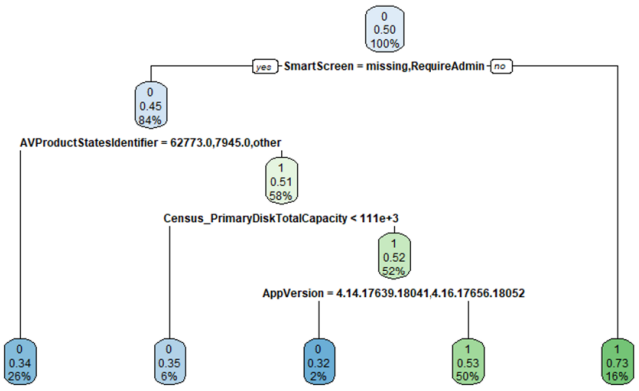

Fig. 9: Tree Structure of Decision Tree Model

being generated. This problem is mitigated by using decision trees within an ensemble. Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

The Decision tree model was built using rpart package in R. rpart uses the Gini Index to quantify the impurity of leave nodes and decide which is the best split. Fig. 9. shows the decision tree plot.

Fig. 10: shows the Confusion Matrix after running the decision tree algorithm using the rpart package.The rpart function was run with different values for tree depth and complexity parameter (CP) to further fine tune the algorithm. However, there was no change observed.

The predictions were run at different threshold values to observe the performance and the results were as follows:

1. For Threshold ≥ 0.54, the precision was high(72.9%) and recall was low(23.3%).

2. For Threshold ≤ 0.54, the precision was Low(58.1%) and recall was low(76.3%).

The Malware detection problem requires that the false negatives are as low as possible, as this factor could cause a major cost to the company. On the contrary false positives are acceptable as these cases would enable the company to be more proactive in taking effective measures to safeguard the machines that are not affected by malware yet. Therefore, the higher recall takes precedence over the precision in this analysis and hence Threshold ≤ 0.53 is considered for prediction.

Fig. 10 shows the ROC curve and the AUC is 63%. The Decision Tree has performed better than the KNN model.

*6) **Ensemble Models - Random Forest**:* Based on our research, ensemble models perform better with malware detection dataset compared to individual models or even neural nets[4]. So we decided to try both bagging based and boosting based models for our project.

Typical bagging algorithms randomly sample a subset of data with replacement for each decision tree, and combine all the decision tree's prediction together by taking the majority vote for classification and mean for regression to make the final prediction. Random Forest not only takes a random subset of data, but also randomly samples a subset of features for each

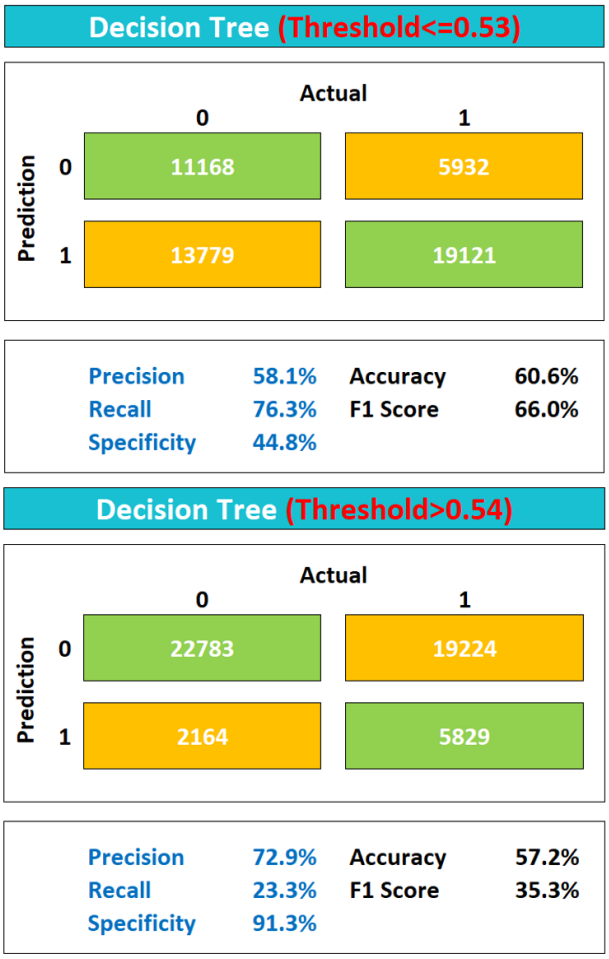

Fig. 10: Confusion Matrix for Decision Tree Model

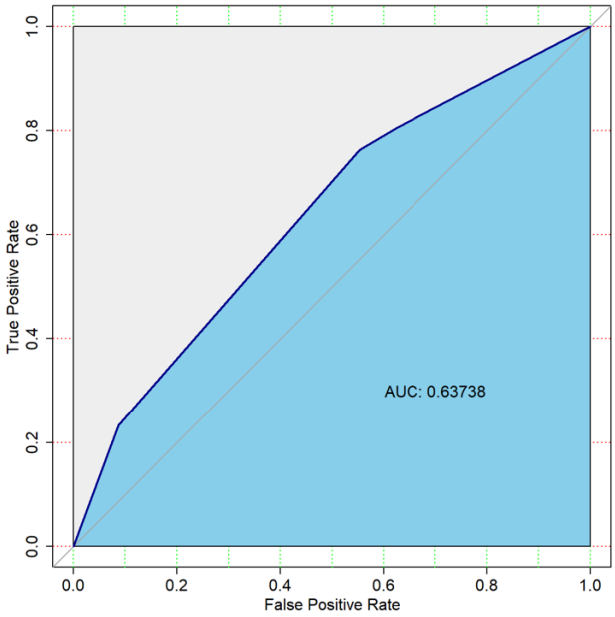

Fig. 11: ROC Curve for Decision Tree Model

decision tree, which further improve the performance of the model.

We build our random forest model with $randomForest$ package in R. We only experimented models with the 3 hyper-parameter due to extremely long run time for each training. Table 1 summarizes the hyper-parameters for our best performing random forest model.

Hyper-parameters	Values
$ntree$	100
$mtry$	2
$nodesize$	10

TABLE I: Hyper-parameter for Random Forest

As shown in the Figure 12. The Area under ROC for random forest is 67.9%, which is only slightly higher than the logistic regression. We may be able to achieve better performance if we had the resource and time to fine tune the model with more hyper-parameters. It is also possible that the decision tree model might be sub-optimal for this dataset because the single decision tree model performs much worse than the logistic regression model. Random forest have already improved the ROC by 4% compared to decision tree model.

*7) **Ensemble Models - Xgboost**:* Boosting is a type of ensemble method where weak learner are built sequentially in order to reduce the overall bias and variance of the model. For each weak learner, the algorithm will randomly sample a subset of data and try to solve for the remaining residual errors.

Gradient Boosting or eXtreme Gradient Boosting (Xgboost) redefines problem as a numerical optimization algorithm. Gradient boosting consists of 3 components, namely, loss function, weak learner, and additive model. Firstly, we need to define a differentiable loss function, for classification problem, we normally use logistic loss. Secondly, we need to define a weak learner such as decision trees. Finally, we add weak learners one at a time as a smooth function in the additive model, where each smooth function is represented as $f_j(X_j)$ in the following equation for additive model. Functional gradient decent is used to minimize the loss when adding each weak learners.

$$Y = \beta_0 + \sum_{j=-1}^{p} f_j(X_j) + \epsilon \tag{7}$$

Hyper-parameters	Values
eta	0.15
$gamma$	5
$max.depth$	5
$subsample$	0.6
$colsample\_bytree$	0.6
$lambda$	0.9
$nrounds$	10

TABLE II: Hyper-parameter for Xgboost

For this project, we decide to use Xgboost library in R and optimize the hyper-parameters using random grid search. However, we had to adjust our search space multiple times to obtain relatively good results. In our first attempt, we provided 0.8 and 0.9 as candidates values for $subsample$ and

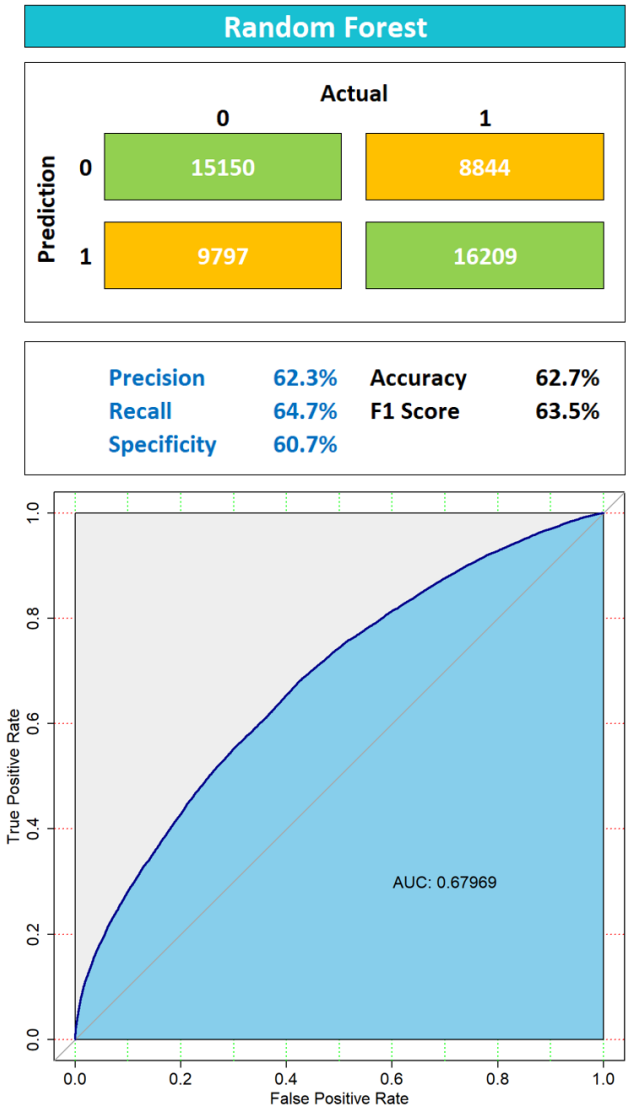

Fig. 12: Confusion Matrix and ROC Curve for Random Forest Model

$colsample\_bytree$ parameters. These two parameter defines the proportion of data and proportion of features we use for each weak learner. The best ROC we could achieve on the test set was only 67%, but the ROC on the training set is almost 75%. This suggests that we overfit the model. As a results, We ran another sets of grid search with $subsample$ and $colsample\_bytree$ ranging from 0.5 to 0.7. This time we can achieve ROC of over 68% on test set, but the ROC on the training set still tells us that we overfit the model. In the final grid search run, we added $lambda$, which is the L2 regularization term on weights, and achieved a ROC of 69.1%. Table II summarizes the hyper-parameters for our final Xgboost model.

C. Model Comparison

We have compared the models based on Accuracy, Precision, Recall, Area under the ROC. All models have similar

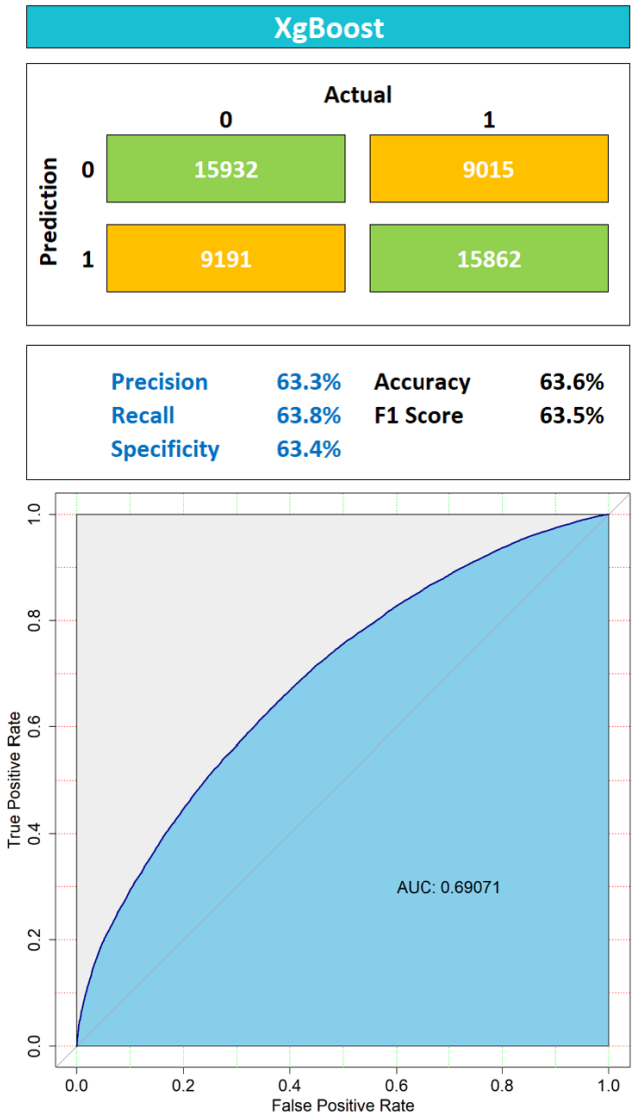

Fig. 13: Confusion Matrix and ROC Curve for Xgboost Model

performance likely because the data does not have any strong predictors or all the values with strong predictability have very low frequency. As shown in Table III, Xgboost perform slightly better than other models.

For malware detection, We would rather give false alarm instead of ignoring some of the potential risk. Therefore, false negative is much worse than false positive. In a real world use case, We should prioritize recall over precision by setting a lower threshold.

According to Hudal Et al [4], their malware detection models achieved accuracy ranging from $96\% - 99.9\%$, Which is significantly higher than what we saw with out model. It is possible that we made some mistake during the data ingestion or preparation steps. However, we believe the under-performance is likely due to the size of our down-sampled data. The original training set had around 8 million data points. Our down-sampled training set only has $150,000$ data points. Considering the data set has 82 independent variables and thousands of levels for all categorical variables, Our training set is relatively small for such feature space.

Models	Accuracy	Precision	Recall	ROC
Logistic Regression	62.7%	62.8%	62.4%	67.7%
KNN	54.0%	54.0%	55.1%	N/A
Decision Tree	60.6%	58.1%	76.3%	63.7%
Random Forest	62.7%	62.3%	64.7%	68.0%
Xgboost	63.6%	63.3%	63.8%	69.1%

TABLE III: Performance comparison across all models

V. Challenges

Here are the challenges faced working on this dataset:

A. Data Size:

The size of the training data is 8 million rows. We had to down sample the data for this project.

B. Missing Values:

There were a lot of features with missing values, and we had to perform data imputations/dropping features.

1) Drop: Drop a feature depending on the percentage of data points that are missing in that feature.

2) Imputation: Impute missing values — including mean, median, mode, or model based.

C. Data Encoding:

Another major challenge was to encode categorical features. Many of the categorical features had large unique values, example - feature 'AvSigVersion' which represents anti-virus definition versions. And performing One-Hot-Encoding would have created thousands of dummy features for such categorical features, hence we had to perform Q-Q Hot encoding of these categorical features.

D. Project Change:

We had to change our project midway as the project we originally chose didn't align properly with the group project requirements. And we spent significant effort coordinating the timings of different team members working from different time zones, and sometimes across continents to successfully complete this project within the timelines.

References

[1] Microsoft Malware Prediction, *https://www.kaggle.com/competitions/microsoft-malware-prediction/data*

[2] Qiangjian Pan et al 2020 J. Phys.: Conf. Ser. 1684 012041, *The Application of LightGBM in Microsoft Malware Detection.*

[3] Shamsul Huda1, Jemal Abawajy, Mali Abdollahian, Rafiqul Islam and John Yearwood, *A fast malware feature selection approach using a hybrid of multi-linear and stepwise binary logistic regression.*

[4] Damaševi˘cius, R.; Ven˘ckauskas, A.; Toldinas, J.; Grigaliunas, Š., *Ensemble-Based Classification Using Neural Networks and Machine Learning Models for Windows PE Malware Detection.*

[5] Guo, G., Wang, H., Bell, D., Bi, Y., Greer, K. (2003)., *KNN Model-Based Approach in Classification.*