Microsoft Malware Prediction

Pavel Zimin

Problem Statement

Computer infection by malware constitutes a serious security problem that could harm consumers and businesses in many different ways. The ability to predict the chances of malware infection before they occur would benefit both consumers and businesses. The signal that the infection is likely to occur would allow for timely countermeasures to be applied. This project would benefit software manufacturers who would be able to incorporate the model into their software for additional security measures aimed at preventing the infection by malware.

The dataset that I will use for this project is provided by Microsoft and is available as part of the Kaggle competition (https://www.kaggle.com/c/microsoft-malware-prediction/data). These data include telemetry properties and infection records of Windows machines as produced by the Windows Defender software. Each row represents an individual machine. Each column represents a variable. The “HasDetections” columns of the train data indicates whether the machine is infected. The goal of the competition is to be able to predict the value of “HasDetections” column for the test data. The train data contains 8,921,483 rows and 83 columns. This is a reasonably large size dataset that allows to explore many Machine Learning techniques. The dataset was resampled such that the frequency of machines with malware approximately matches the frequency of machines without malware detected. As an evaluation metric I have selected the area under the ROC curve, which was also used in the competition.

For this project, I have performed feature engineering, feature selection and machine learning using logistic regression, Random Forest, gradient boosting and neural networks.

The final outcome of this project is a Jupyter notebook with the code, a report and presentation slides.

Description of the Dataset and Cleaning Steps

The dtype of each column was specified in order to reduce the memory usage when reading from disk. The train dataset contains 8,921,483 rows and 83 columns. The goal of this project is to be able to predict the 'HasDetections' column.

To read the data into memory, the dtypes were specified to read_csv function such that the pandas DataFrame takes less RAM space. The dataset's attributes were classified manually into the categorical and numeric groups based on the attribute description on the Kaggle web page. Inspecting the value counts of the categorical Missing values are either coded as NaN or using various placeholders, such as 'UNKNOWN', 'Unknown', 'unkn' or 'Unspecified'. These values were substituted with np.nan values. Features with more than 1/3rd values missing were deleted from further analysis. In addition, for each column with missing data I created another column in which missing values were coded as 1, and 0 otherwise. Further analysis of missing values using the `missingno` package revealed that there are observations with 0-20 values missing. The correlation heatmap of missing values shows that there are features that are highly likely to be missing similarly (Figure 1). For example, missing values of 'IsProtected' are highly correlated with the missing values of 'AVProductStatesIdentifier', 'AVProducstsInstalled' and 'AVProductsEnabled'. This makes sense since these attributes describe the parameters of antivirus software. Another group of attributes that are highly correlated for the missing values: 'Census_ProcessorManufacturerIdentifier', 'Census_ProcessorCoreCount',

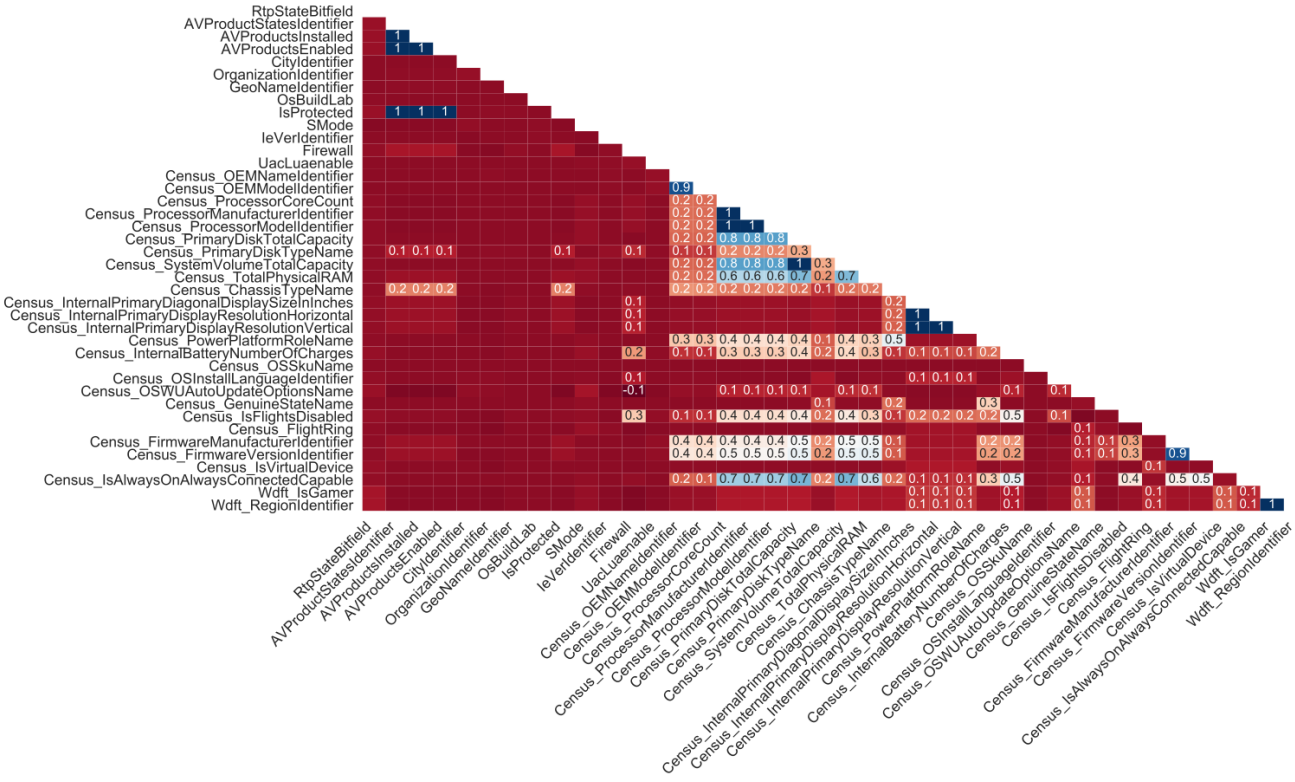

Figure 1. Correlation heatmap of missing values in columns. Only columns with missing values are visualized.

'Census_ProcessorModelIdentifier'. Again, this is not surprising since these features are the processor's parameters and will tend to be missing together.

Since most machine learning models require the complete dataset, I filled the categorical missing values with 0, while adding 1 to the existing values. The numeric columns were filled with the median values. The median values of all the numeric columns were saved for further use when need to fill the missing values of the test data or future data.

For outlier detection of numeric features 2 methods were used: 1) univariate outliers were identified as lower than 25th percentile minus 1.5 interquartile range or greater than 75th percentile plus 1.5 interquartile range, 2) multivariate outliers were detected by first normalizing the values by subtracting the mean and dividing by the standard deviation, then the principal component analysis (PCA) was applied (Figure 2). The scatterplot showing the distribution of the principal component 1 and principal component 2 shows the outlying data points with the greatest variance. Both methods were able to identify the outliers, however I decided against removing the outliers from further analysis since the most promising machine learning algorithms such as random forest are immune to outliers. In additions, outliers can contain information that could be important for the prediction.

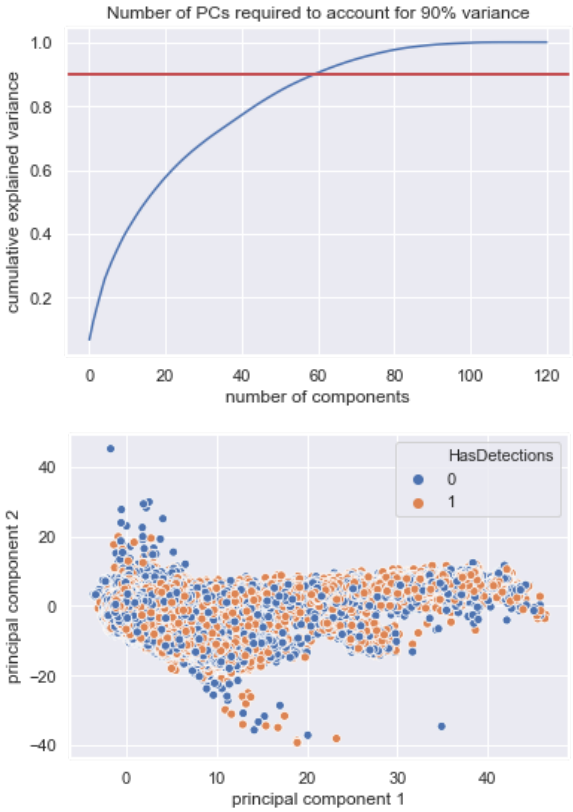

Figure 2. Principal Component Analysis of the training data. Top panel shows the cumulative explained variance as a function of the number of components. Bottom panel shows the scatter plot of the first 2 principal components.

Exploratory Data Analysis

What features are correlated with the target variable?

To identify the features that correlated with the target variable ‘HasDetectsions’, I plotted the hierarchical cluster heatmap (Figure 3). Due to the large number of features in the dataset for the figure I only filtered the features with the highest absolute correlation coefficient to the target variable. The highly correlated features are ‘AVProductsInstalled’, ‘AVProductStatesIdentifier’, ‘Processor’, ‘Census_OSArchitecture’.

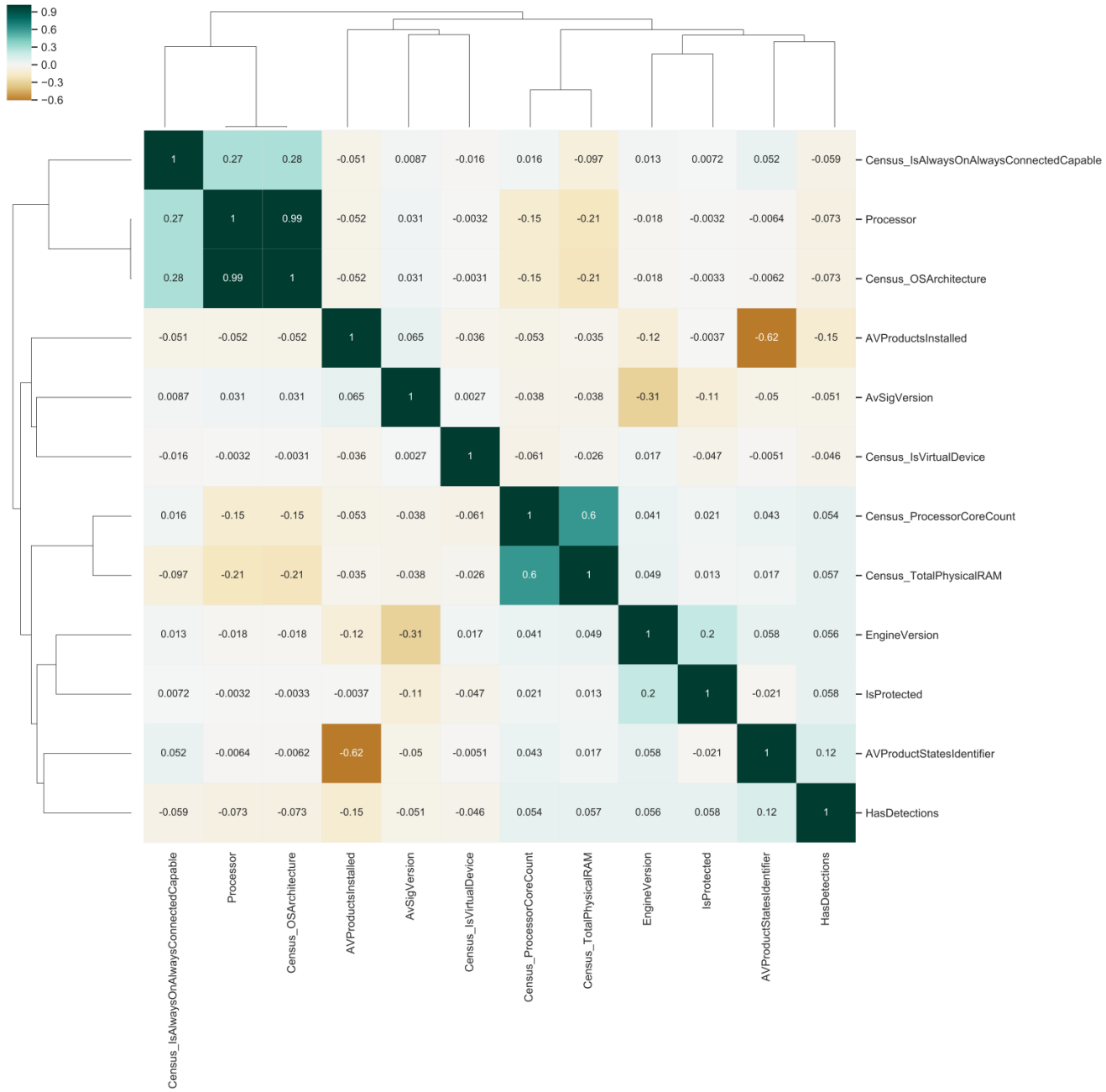

Figure 3. Correlation clustermap of the dataset features. Only features with the highest correlation with the target variable are visualized.

Next, I plotted the fraction of the positive target variable as a function of selected categorical features. I also built a weighted least square models using the value count as a weight. Data points with the large count were weighted larger than the points with the fewer count. This

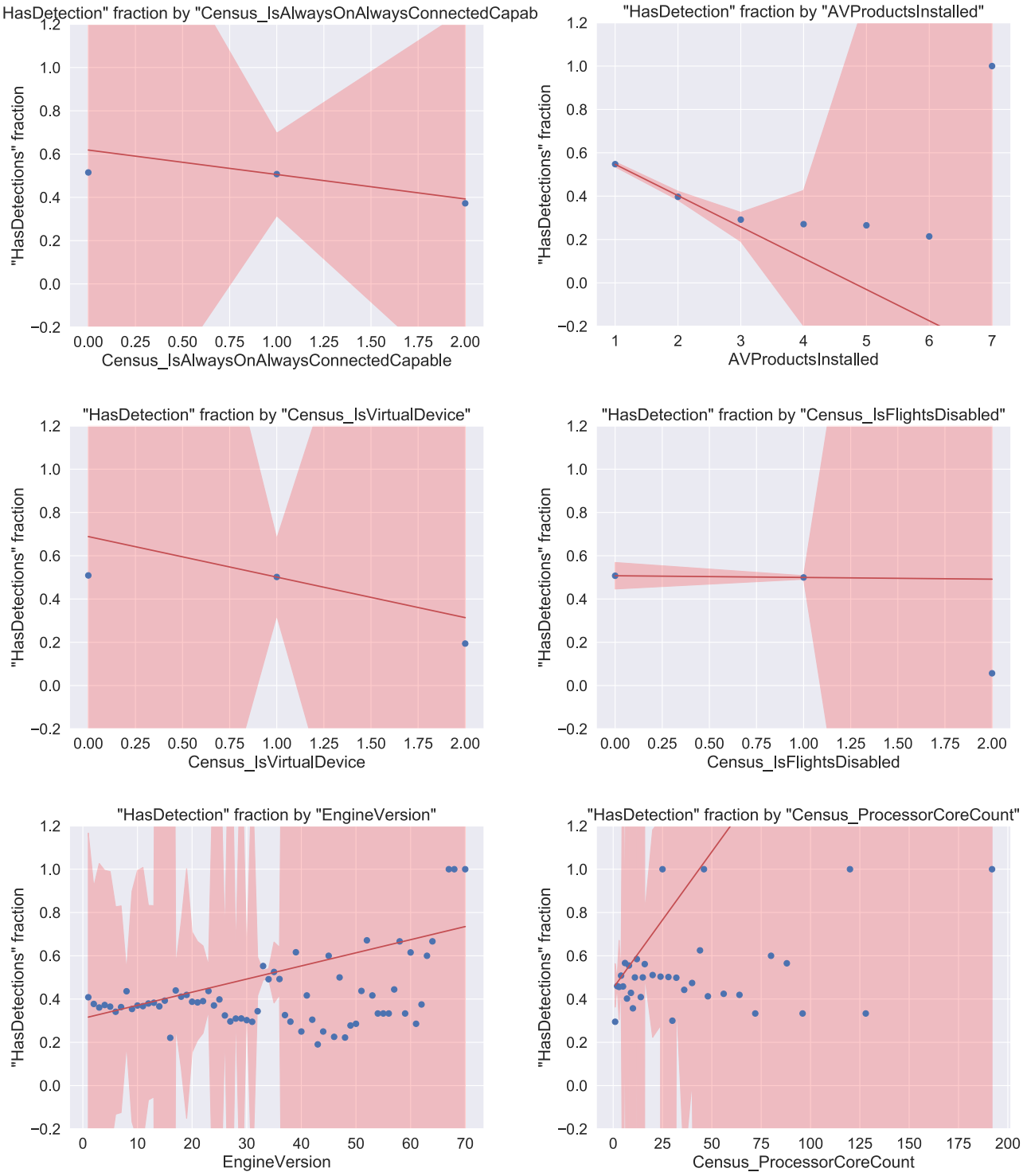

Figure 4. Fraction of the positive target variable as a function of selected categorical feature values. Weighted least square models were created using the value count for each categorical value as weight. Prediction interval is shaded in red shade.

allows to minimize the effect of the points with the few counts and enable to see the model that reflects the large portion of data. Below is the analysis of the graphs shown in Figure 4:

1. `Census_IsAlwaysOnAlwaysConnectedCapable`. This variable has the majority of data points with the value of 1. Since there are very few data points in either direction from 1, the feature's coefficient is not significantly different from 0 (p value is 0.25).
2. `AVProductsInstalled`. The target variable positive fraction shows significant dependence at the low values of `AVProductsInstalled` variable (slope of -0.1443, p value = 0.000). The fraction of the target variable at larger values of `AVProductsInstalled` variable cannot be predicted with this model due to the few value counts.
3. `Census_IsVirtualDevice` variable shows an issue similar to the `Census_IsAlwaysOnAlwaysConnectedCapable` variable. The most data points have a value of 1, few data points in either direction from 1. The feature's coefficient is not significantly different from 0 (p value = 0.424).
4. `Census_IsFlightsDisabled` variable is approximately equally split by the target variable at values 0, and 1. At value 2, the target variable positive fraction is below 0.5, but there are a few point counts, hence nonsignificant slope (p value = 0.261).
5. `EngineVersion` has a larger spread of possible values, but the data are concentrated at a few points. The target variable positive fraction tends to increase with the increase in the `EngineVersion` values. The feature's slope from the model is 0.0061 (p value = 0.000).
6. `Census_ProcessorCoreCount` has also a large spread of possible values, but the data is concentrated at the lower values where the relationship of the positive fraction of the target variable is increasing with the increase in the values of `Census_ProcessorCoreCount` values (slope = 0.0126, p value = 0.000).

Graphs visualizing additional features can be found in the accompanying Jupyter notebook.

Overall, the EDA demonstrated that the correlation of the features to the target variable is quite low, with some features showing statistically significant relationships with the target variable.

Machine Learning

Before building machine learning models, I created a function `read_clean` that reads the raw data and does all the preprocessing steps described earlier in this report. Then, I applied a custom function `data_split` to split the data in a training set (~70% of the data), hold-out set 1, hold-out set 2 and hold-out set 3. Each hold-out set contains ~10% of the data. Training set was used for training the model, the hold-out set 1 was used for the hyperparameter tuning, the hold-out set 2 was used for model selection, and hold-out set 3 was used only once at the end of this project to evaluate the final model on the data that was never exposed to any of the machine learning algorithms.

Feature Selection

The cleaned data contained 140 features. The aim of this section is to eliminate features that are not important for malware prediction. I tried 3 methods for feature selection: 1) Random Forest, 2) Correlation analysis, 3) Boruta algorithm. Below is the description of how each approach works:

1) Random Forest. scikit-learn's implementation of Random Forest algorithm outputs the feature importance for each feature. This is done by calculating impurity reduction for each feature and for each tree. Then this parameter is averaged across all the trees and is returned. I selected features with their feature importance higher than 0.0006 (arbitrary threshold).
2) Correlation analysis. Since the dataset contains highly correlated features, I calculated Pearson's correlation coefficients for each pair of features and eliminated a feature from each pair with the correlation coefficient greater than 0.95 (arbitrary threshold) until no more pairs are left with the correlation coefficient greater than the threshold.
3) Boruta algorithm. This algorithm creates shuffled versions of each features called shadow features and trains a random forest classifier on the dataset containing original features and shadow features. Then the algorithm evaluates feature importances of each feature and calculates z-scores. Features with the importance significantly lower than the maximum Z-score among shadow features is considered unimportant and removed from the process. Features are considered important if their feature importance is significantly higher than the maximum Z-score among shadow features. This process

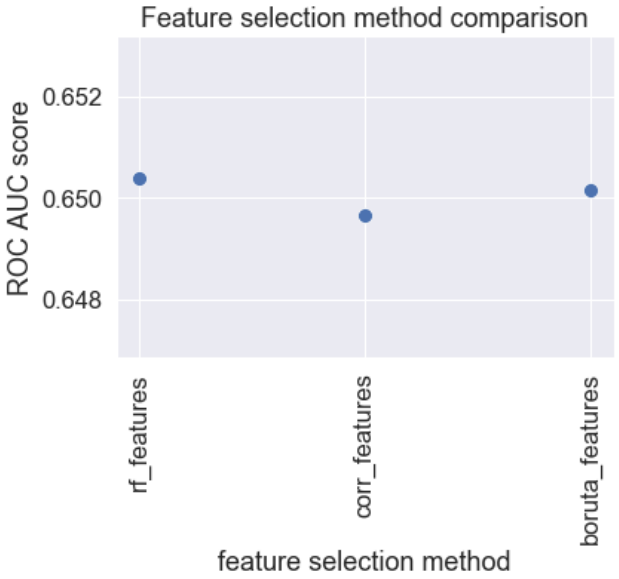

Figure 5. Comparison of feature selection algorithms.

is repeated until the predefined number of cycles is reached or until all features are labeled as important or unimportant whatever comes first. One of the advantages of this algorithm is that it does not require an arbitrary threshold to be provided. One of the disadvantages is that it requires a lot of resources to train so many random forests.

After features were selected with all 3 algorithms three Random Forest models were trained each using just features determined with one of the three algorithms. Then the ROC AUC score was compared for each of the models (Figure 5). All three algorithms produced similar scores. The highest score was obtained with the Random Forest-based feature selection. This algorithm selected 95 features which will be used for the remaining part of the project.

Hyperparameter Tuning

To build the predictive model I selected the following algorithms: logistic regression, random forest, gradient boosting and neural network. For each model I performed a hyperparameter tuning by doing a grid search over predefined ranges of hyperparameters. To do that I built a custom function `grid_search` which iterates over all combinations of hyperparameters in the parameter grid, trains the model on the training data and tests the model on the supplied validation data (hold-out set 1 in this case). Since this process requires a large number of model training and the training dataset has over 6 million rows, I decided to perform the hyperparameter tuning on a one tenth sample of the whole training set.

Logistic Regression. Data were scaled to the range 0-1 using scikit-learn's MinMaxScaler method. Then I used SGDClassifier class from scikit-learn to perform a grid search for regularization parameter alpha in the range of 0.00001 to 100; regularization methods of 'l1', 'l2', and elastic net; l1 ratio for elastic net was in the range of 0.1 to 0.9. The best ROC AUC score on the hold-out set 1 was obtained for the alpha of 0.0001 and 'l1' regularization.

Random Forest. Since random forest is insensitive to how the features are scaled, no scaling was performed on the data here and for the gradient boosting. I fixed the parameter for the number of estimators as 100 and performed a grid search over the following ranges of hyperparameters: maximum depth (10-100, None), minimum samples considered for a split (5-1000), maximum features (10-60). The greatest ROC AUC score on the hold-out set 1 was obtained for the following hyperparameters: maximum depth – 40, minimum samples considered for a split – 100, maximum features – 30.

Gradient Boosting Classifier. Since the gradient boosting model has many parameters and slow to train, I used an iterative approach in which I first optimized the hyperparameters that are most influential followed by optimization of parameters that have less effect on the performance of the model. The parameters were trained in the following order: number of estimators, maximum depth and minimum samples split, minimum samples leaf, maximum features, subsample.

Neural Network. I used a functional API from keras package to build, train and test neural network models. Since many features are categorical, I used entity embeddings for such features. I tried a few architectures of neural networks. The best model had all numerical features projected to a dense layer with 200 neurons. This layer was concatenated with the entity embeddings of categorical features and projected to a dense layer with 500 neurons. This layer was followed with 2 layers with 150 neurons each. Each dense layer had a dropout applied. The output layer was a single binary neuron.

Model Selection and Testing

To compare the models, I have combined the complete training set and the hold-out set 1 and trained each model on this aggregate training set. Then I evaluated the performance of each model on the hold-out set 2 by computing ROC AUC scores (Figure 6). The model with the greatest score was gradient boosting classifier (~0.72). The model performed the worst was logistic regression (~0.64). Neural network models were in the middle.

Finally, the final gradient boosting classifier model was trained on the combined training set, hold-out set 1 and hold-out set 2 (Figure 7). Its performance was evaluated on hold-out set 3 which has not been used before in any machine learning steps. The ROC AUC score of this model was 0.74 which is close to the one obtained at the previous step. This indicates that the model should generalize well to the new data. The model was saved to disk for future use.

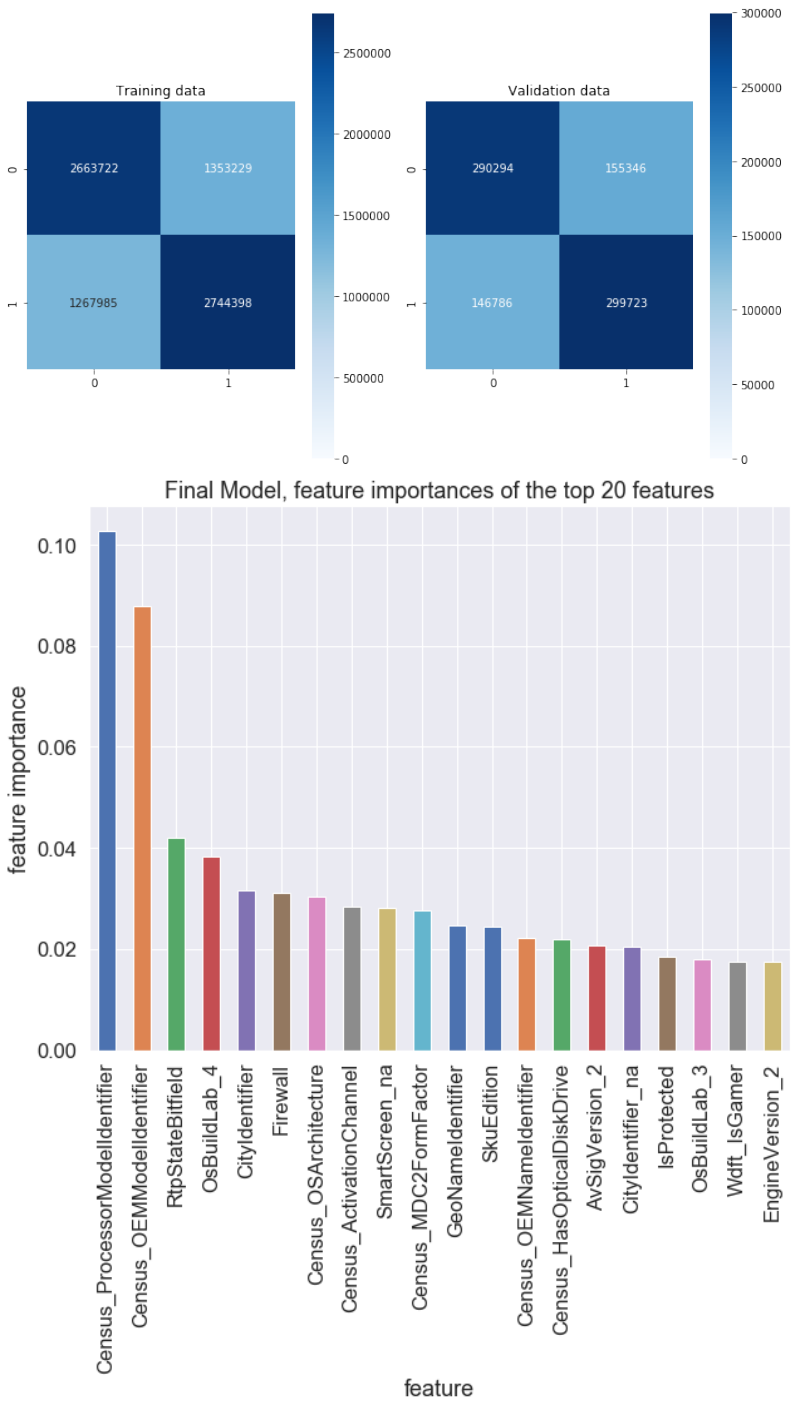

Figure 7. Final model performance and feature importance.

Recommendations

The ROC AUC score on the test data was 0.74 which is significantly better than a guess, but is quite far from the ideal value of 1. This model outputs a probability of malware infection. Since it is more important to catch as many potentially infected machines as possible, the probability cutoff should be selected at less than 0.5.

Acknowledgments

I would like to thank my mentor, Ramkumar Hariharan, for his advice and guidance.