Authors: Akarsh Rastogi (ar4661), Anqi Wang (aw3615), Selim Shaalan(ss6952), Tseng-Han Yu (ty2487), Ziqi Li (zl3350)

Malware Prediction System

Topics in Computer Science: Applied Machine Learning
Group 28

Introduction

As digital reliance grows, so does the importance of cybersecurity in safeguarding against malware, which can lead to significant breaches and financial losses. This paper presents a Malware Prediction System that employs machine learning techniques to preemptively identify potential malware threats, thereby bolstering digital defenses.

Leveraging a robust Malware Prediction dataset from Microsoft, featuring about 9 million data points and 83 variables, our study evaluates various machine learning methods such as Logistic Regression, Random Forest, Gradient Boosting, and Neural Networks. These methodologies are selected for their efficiency in processing large datasets and their capacity to discern intricate patterns, aiming to enhance malware detection and contribute to cybersecurity advancements.

Data

In addressing the challenge posed by the large-scale dataset, our initial step involved a thorough examination of data quality and balance. We discovered that while there is a substantial amount of complete data, there are also notable instances of missing information. Specifically, out of the 83 variables, 39 exhibit no missing values, and 35 contain less than 10% missing data. However, the dataset also presents more problematic areas with 9 columns showing over 30% missing data, which we opted to exclude from our analysis to maintain integrity. The target variable 'Has_detections' is evenly distributed, with approximately 50% of the data points indicating malware presence. This balance is critical as it allows for a more unbiased learning process for the models.

During data preprocessing, a uniform strategy was employed where 20% of the data was randomly sampled to facilitate manageable computation across all models. Categorical features were processed using One Hot Encoding, with high cardinality features being selectively dropped or target encoded to mitigate the expansion of the feature space. Missing data was addressed by discarding columns with substantial missing entries and imputing values for the remainder. Feature selection was executed to exclude variables with low variance or insignificance to the predictive task, ensuring a refined set of predictors for modeling.

Methodology

1. Logistic Regression

We implemented logistic regression with default values of the parameters first. Then, we implemented lasso logistic regression and ridge logistic regression models with the Liblinear solver for optimization, and used grid search and 5-fold cross validation for hyperparameter tuning. The parameters we used to tune in grid search is the inverse of regularization strength (C) and the type of regularization (L1 and L2). We also implemented the elastic net logistic regression with the SAGA solver, and used grid search to tune the L1 ratio. We used accuracy as the metric to evaluate the model.

Authors: Akarsh Rastogi (ar4661), Anqi Wang (aw3615), Selim Shaalan(ss6952), Tseng-Han Yu (ty2487), Ziqi Li (zl3350)

2. Random Forest

We used sklearn's RF classifier with 5-fold cross validation and random search to implement the model and hyperparameter tuning, and each iteration took about 15 minutes to complete. Since our dataset was balanced, accuracy was used as the primary metric to maximize.

3. Gradient Boosting

A pipeline integrating preprocessing and model training to streamline the workflow was utilized. The XGBClassifier was selected due to its robust performance on structured datasets. To mitigate overfitting and enhance model performance, we engaged in hyperparameter tuning using RandomizedSearchCV, optimizing for the area under the receiver operating characteristic curve (ROC AUC) across a five-fold cross-validation scheme. The final model was evaluated on a hold-out test set, with accuracy, classification report, and confusion matrix serving as performance metrics. The entire methodology was encapsulated within a reproducible framework, with the final model and preprocessing pipeline serialized using joblib for future predictions or replication of the study.

4. Neural Networks

We finally tried a Deep Learning approach to tackle the problem. If we encode using OHE, the number of features produced are extremely high since there are 211,562 values. Hence we use a statistical trick to reduce the high number of One-hot encoded categorical variables, using a threshold. This only adds 350 new boolean variables instead of the larger number mentioned above.

After having dealt with the features, we first built a 3 layer fully connected network with 100 neurons on each hidden layer. We use a batch size of 32, ReLU activation for all layers (except final), Batch Normalization, Dropout, Adam Optimizer, and Learning Rate scheduler (starting from 0.01 and decaying). After each epoch, we use a custom Keras callback to display the current Accuracy and AUC and continually save the best model on the basis of AUC scores obtained. Considering the size of the data, it took a significant amount of time to train.

Our second experiment deepened the network to 5 layers with the following ANN architecture : [200 -> 300 -> 100 -> 10 -> 1], and a larger batch size (64). The rest of the setup was kept the same as before. For the inference part on Kaggle's test set, even after deleting the training data, our network still needed a lot of RAM, so we load in the data by chunks and predict by chunks. A third experiment used 128 batch size, a balanced data sample for training and validation (stratified), and a higher starting learning rate.

Model Results

1. Logistic Regression

The baseline model achieved an accuracy of 63.0047% on the training dataset and 62.93% on the validation dataset. After hyperparameter tuning, the best model we found with best parameters has an accuracy of 63.0077% on the training dataset and on the validation dataset. The best parameters are C = 1.0 and L2 regularization. The elastic net logistic regression model achieved an accuracy of 63.0017% on the training dataset and 62.95% on the validation dataset. The best parameters are C = 0.03162 and the L1 ratio = 0.3.

2. Random Forest

The baseline model achieved an accuracy of almost 100% on the training set and 61% on the validation dataset.. It should be noted that many other hyperparameters were attempted to be tuned such as min impurity decrease and leaf node limitations, but none had a positive effect on the performance of the model initially. Further trials were made with max depth, criterion and min samples split until the models performance became 64% on the training and 62% on the validation and test datasets. This is to be expected since we're massively limiting our amount of estimators and samples used due to time and hardware constraints, as well as the best performing models only being able to achieve accuracy scores of around 67%.

3. Gradient Boosting

The baseline model achieved an accuracy of 64% on training and 63% on validation datasets. In contrast, the fine-tuned model showed a slightly better accuracy of 66% on training and 64% on validation, along with a test accuracy of approximately 63.7%. These results highlight the benefits of hyperparameter optimization, with an evident improvement in the precision-recall balance, especially for the positive class, as reflected in the f1-scores. Feature importance analysis (see Appendix 1) revealed that `Census_IsVirtualDevice_0.0`, `AVProductStatesIdentifier_53447.0`, and `AVProductStatesIdentifier_63682.0` were particularly influential in the model's predictions, emphasizing the critical role of specific system attributes. This suggests that such features are valuable for model performance and could be pivotal in guiding further feature engineering and data strategy.

4. Neural Networks

We achieved a training accuracy of 64%, a validation accuracy of 64.2%. The Train AUC was 0.710 and Validation AUC was 0.702. Upon submission to Kaggle we received a test accuracy of 62% on the private dataset provided, and 67% on the public leaderboard. For the 2nd experiment with the deeper NN with 5 layers and larger batch size, we only ran 10 epochs. We achieved a training accuracy of 64.5%, a validation accuracy of 64.3% (see Appendix 2). The Train AUC was 0.715 and Validation AUC was 0.705. Upon submission to Kaggle we received a test accuracy of 61% on the private dataset provided, and 67% on the public leaderboard, which are similar to what we attained before. The third experiment did improve results but insignificantly.

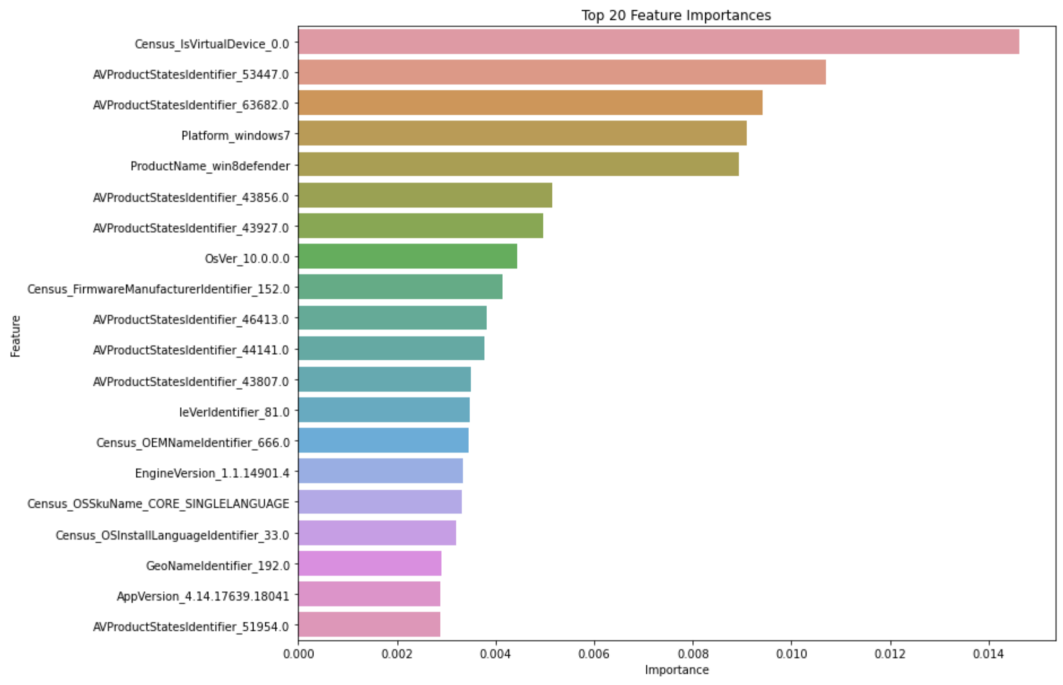
Conclusion

Our study on malware prediction using a large Microsoft dataset reveals the potential of machine learning in cybersecurity. We prioritized data quality, leading to strong Logistic Regression, Random Forest, Gradient Boosting, and Neural Network models. The superior performance of Random Forest and Neural Networks, along with Gradient Boosting's insight into feature relevance, highlights the impact of precise data preprocessing and model optimization. These findings lay the groundwork for advanced, adaptable malware prediction systems, providing a pathway for ongoing research in this critical field.

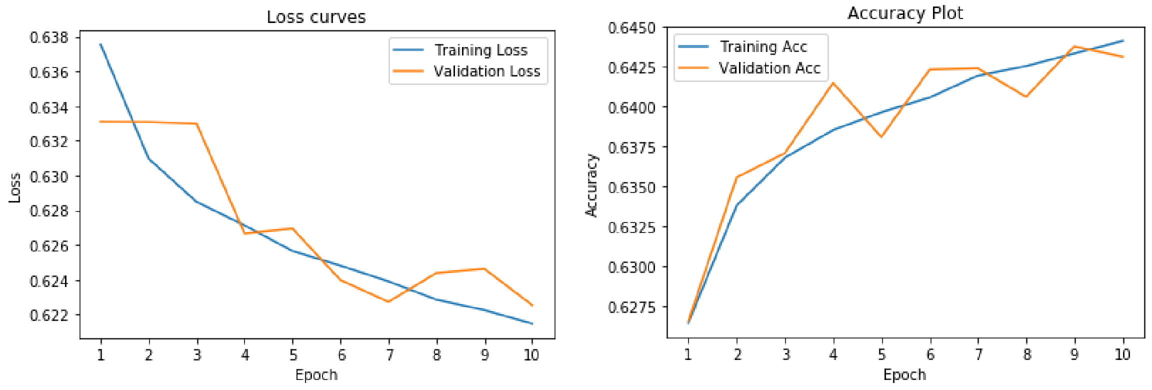
Authors: Akarsh Rastogi (ar4661), Anqi Wang (aw3615), Selim Shaalan(ss6952), Tseng-Han Yu (ty2487), Ziqi Li (zl3350)

Appendix

Appendix 1



Appendix 2


Loss curves
Training Loss
Validation Loss
Loss
Epoch
Accuracy Plot
Training Acc
Validation Acc
Accuracy
Epoch