Malware Prediction : Microsoft's Kaggle Challenge

Utkarsh Mittal
Stanford University
mittal12@stanford.edu

Yun Kun Zhang
Stanford University
wyzhang@stanford.edu

Samarpreet Singh Pandher
Stanford University
samar89@stanford.edu

Project Report
CS 229 : Machine Learning

December 8, 2021

Abstract

Malware is a worldwide epidemic. Since the advent of computers and internet, there have been rising threats of data breaches, causing companies to lose billions of dollars and file for bankruptcy. A recent study published by Cyber Security attack statistics [1], collate the colossal damage by malicious actors. According to the study almost 1.3m new pieces of malware are created daily and 63% of data breaches are financially motivated. Most popular malicious attacks are in form of (.zip) or (.jar)

1 Introduction

In this day and age, there are still lots of companies that lack infrastructure and deploy traditional malware detection systems to counteract malware threats, and hence fall prey to unscrupulous actors. Not only are these systems expensive but also have latency problems. On the other hand, Microsoft has actively built anti-malware products over the years and collected data from millions of computers around the world. In order to effectively analyze and classify such large amounts of data, we propose using machine learning models to predict how susceptible a machine is to malware threat. In this project, we will apply different Machine Learning algorithms for this prediction task, and compare their accuracy on the basis of multiple metrics like Accuracy, Precision, Recall amongst others.

2 Related Work

Anti-malware applications face many challenges today, One of the major challenges is the need to analyze large volumes of data for possible malicious intent.[2] It is estimated that more than 2.5 quintillion bytes of data is generated and captured every year. With increasing focus on malware detection, more than 90% of such data was generated in last two years, and it amounts to nearly 40 trillion gigabytes of Data.[3] It is worth mentioning that Microsoft's anti-malware detection application runs real-time on over on 600 million computers worldwide.[4]

Microsoft's Malware Prediction Challenge on Kaggle is a popular Kaggle Challenge. One of the competitors adopted a method using Naıve Transfer Learning. They trained a Gradient Boosting Machine (GBM) model on the training data, and fine-tuned it to achieve better accuracy on the test set. In order to minimize the marginal distribution gap between target and source domains, they identified the key features for domain adaptation. They then changed the results of their predictions based on the general statistical regularities that they extracted from the training set. They used a GBM to estimate the importance ratings of all the features, and picked only 20 of the most important category features for training their model. This helped simplify the simplify the problem and helped reduce computation cost. Using only 20 of the most important features, they achieved an accuracy of 63.7% on the test set. For another model in which they removed columns with maximum mean discrepancy, they achieved an accuracy of 64.3% on the test set.[5]

3 Dataset and Features

3.1 Dataset

For this project, we utilized 'Microsoft Malware Prediction' Dataset shared by Microsoft as a part of the Kaggle Competition.[6] The goal of this competition is to predict a Windows machine's probability of getting infected by various families of malware, based on different properties of that machine. This data was generated by combining heartbeat and threat reports collected by Microsoft's Windows Defender. The dataset contains data for over 7 million machines.

3.2 Features

The dataset consists of 80 features. Each row in this dataset corresponds to a machine, uniquely identified by a MachineIdentifier. The target variable,HasDetections, is a binary variable and is the ground truth and indicates that Malware was detected on the machine.

3.3 Missing Values

Eight columns have 90 percent of the data missing. We removed those columns from the training as those columns were not adding value to the model accuracy. Thirty percent of the data has null values, as part of data cleaning exercise, we removed that data from the dataset.

3.4 High Cardinality

The dataset has very high cardinality. To deal with cardinality, a customized function is created which formulates and counts the no of occurrences of the particular value. The function is based on the cumulative sum and captures up to 75 percent of the most frequent values in the column. Low frequent values are assigned as "Other" in the dataset.

3.5 Class Imbalance

Classes are balanced in the dataset provided. Roughly 50 percent of the datapoints correspond to having Malware and 50 percent correspond to having no Malware.

3.6 Feature Dimensions

After data cleaning and pre-processing, there are 72 features, 25 of which are categorical features. To avoid explosion of the feature space, we use hashing [7] to transform the categorical features instead of one-hot encoding. In addition, we applied sparse principal component analysis (PCA)[8] to reduce the number of features to ten. In Experiment section, we experimented with both the full set of 72 features and the reduced feature space.

4 Metrics

We used multiple metrics like Accuracy Rate, F-1 score, Recall and Precision to measure the performance of the models.
We decided to use Prediction Accuracy as our Primary metric. Based on the Confusion Matrix, the Accuracy Rate is defined as:

$$\text{Accuracy} = \frac{(TP+NP)}{(TP+FP+TN+FN)}$$

In addition to Accuracy, we used Precision, Recall and F-1 score as our Secondary metrics. These can also be estimated based on the the confusion matrix. Specifically,

$$\text{Precision(Specificity)} = \frac{TP}{(TP+FP)} \qquad \text{Recall(True Positive Rate, Sensitivity)} = \frac{TP}{(TP+FN)}$$

$$\text{F1-score} = \frac{2*(Precision*Recall)}{(Precision+Recall)}$$

5 Experiments

We tested different algorithms that are used commonly for binary classification, as described below:

Baseline: Ordinary Logistic Regression: Logistic regression is part of generalized linear regression family. It's easy to implement, interpret, and very efficient to train[9]. The logistic regression classifier can be expressed as:

$$h_\theta(x) = \mathrm{g}(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

Where g is the Sigmoid function, and θ is the parameter. As a baseline, we experimented with Ordinary Logistic Regression with all 72 features.

Logistic Regression with Regularization and PCA: After establishing a baseline, we also experimented with Logistic Regression with Regularization, and Logistic Regression with sparse feature space obtained using Principal Component Analysis (PCA).

Decision Tree: A decision tree is a flowchart-like tree structure where an internal node represents feature over which a split is made, the branch represents a decision rule, and each leaf node represents the outcome[10]. The decision rule is constructed by splitting the training samples using the features from the data that work best for the specific task. This is done by evaluating certain metrics, like the Gini index or the Entropy. We set the maximum depth to five to avoid over-fitting.

Random Forest: Random forecast ensembles multiple decision trees trained in parallel with bagging technique[11]. Bagging allows individual tree to be trained on random subsets of training data and subsets of features that are sampled with replacement. The randomness makes each tree unique. Thus the aggregation of these trees makes the random forecast more robust to noise in the training data than a decision tree. We set the maximum dept for each tree to five and number of trees to 100 in the initial experiment.

AdaBoost: AdaBoost is short for Adaptive Boosting. It fits a sequence of weak learners on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted sum to produce the final prediction[12]. Specifically, $\hat{y} = sign(\sum_{t=1}^{T} W_t f_t(x))$.

Initially, those weights a_i are all set equally to 1/N, where N is sample size. As a result, first step simply trains a weak learner on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the re-weighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. Specifically,

$$a_i = a_i e^{-W_t} \text{ if } f_t(x_i) = y_i$$

$$a_i = a_i e^{W_t} \text{ if } f_t(x_i) \neq y_i$$

Then the coefficient W_t is updated by $0.5 \ln(\frac{1-weightederror(f_t)}{weightederror(f_t)})$

In the initial experiment, we used the default setting from scikit-learn in which number of weak learner = 50 and learning rate = 1.

XGBoost: XGBoost is one of the most popular and efficient implementations of the Gradient Boosted Trees algorithm, a supervised learning method that is based on function approximation by optimizing specific loss functions as well as applying several regularization techniques. The most important factor behind the success of the XGBoost implementation of gradient boosting trees is its scalability and computational efficiency, due to several systems and algorithmic optimizations[13]. These innovations include: a novel tree learning algorithm for handling sparse data; a theoretically justified weighted quantile sketch procedure that enables handling instance weights in approximate tree learning. Parallel and distributed computing makes learning faster which enables quicker model exploration and effective cache-aware block structure for out-of-core tree learning. The goal of

XGBoost is to minimize the following regularized objective:

$$L(\phi) = \sum_t l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\Omega(f_k) = YT + \frac{1}{2}\lambda||\omega||^2$$

$L(\phi)$ is the loss function measuring difference between prediction and target for each instance.
Ω penalizes the complexity of the model
Each f_k corresponds to an independent tree structure q and leaf weights ω

The tree ensemble model in the above equation includes functions as parameters and cannot be optimized using traditional optimization methods in Euclidean space. Instead, the model is trained in an additive manner. Formally, let $\hat{y}_i$ be the prediction of the i^{th} instance at the t^{th} iteration, we will need to add f_t to minimize the following objective.

$$L^{(t)} = \sum_{i=1}^{n} l(\hat{y}_i^{t-1}, y_i + f_t(x_i)) + \Omega(f_t)$$

f_t is greedily added so as to make maximum improvement to the objective function. In the initial experiment, we used the default settings from the XGBoost library, in which learning rate =0.3, maximum depth =6, L_2 penalty = 1 and L_1 penalty = 0

Neural Network: A neural network consists of multiple layers of neurons, which are highly interconnected nodes that can process information in a highly efficient manner[14]. Each neuron is a mathematical function which first computes a weighted sum of its inputs and then return a value based on its activation function and the weighted sum. The input weights for each neuron are optimized by back propagation and stochastic gradient descent. We used a shallow network where the input layer has 72 neurons, the hidden layer has 512 neurons and the output layer has one neuron. Relu activation function was used in the hidden layer and ADAM was used as the optimizer.

Voting Classifier: Voting classifier combines different machine learning classifier and use a majority vote or the average predicted probability to predict the class labels. Linear SVM, XGboost and Random Forest were chosen as part of voting classifier.

5.1 Results

The performance metrics for the experiments are shown in the Table below:

Algorithm	Train				Test			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
Logistic Regression	0.603	0.612	0.531	0.569	0.603	0.614	0.532	0.570
Logistic Regression PCA	0.580	0.599	0.447	0.512	0.580	0.601	0.443	0.510
Regularized Logistic Reg	0.602	0.596	0.671	0.631	0.600	0.596	0.672	0.632
Decision Tree	0.602	0.601	0.572	0.586	0.602	0.600	0.573	0.587
Random Forest	0.605	0.619	0.519	0.564	0.606	0.619	0.521	0.566
AdaBoost	0.614	0.629	0.529	0.576	0.612	0.626	0.529	0.573
XGBoost	0.648	0.657	0.596	0.625	0.635	0.644	0.583	0.614
Neural Network	0.627	0.626	0.601	0.617	0.622	0.618	0.605	0.612
Voting Classifier	0.715	0.737	0.658	0.695	0.634	0.640	0.581	0.609

We can see that the ensemble methods and Neural Network have higher Accuracy than logistic regression and decision tree. Logistic regression with PCA transformation has lower Accuracy than logistic regression with full set of features. **XGBoost and Voting Classifier have the highest Prediction Accuracy.**
Voting Classifier achieves accuracy rate of 0.715 in training data and 0.634 in test data. The performance gap indicates an over-fitting issue. XGBoost has more consistent performance with

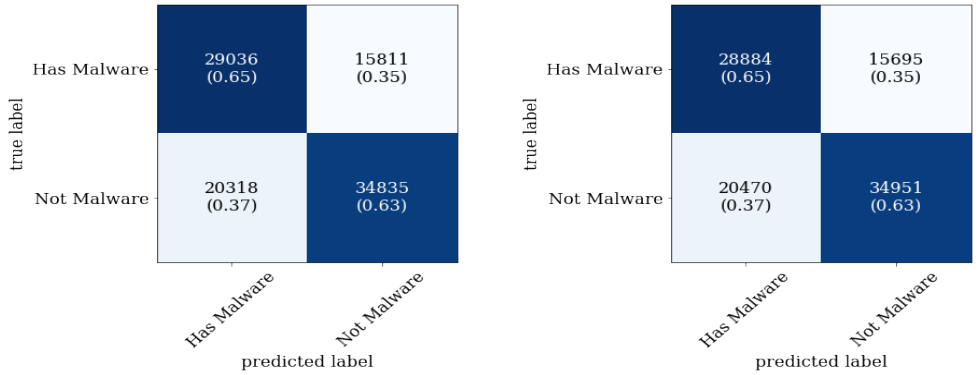
accuracy rate of 0.648 in training data and 0.635 in test data. In addition, XGBoost has better performance than Voting Classifier in test data, with Accuracy 0.635 vs 0.634 and Recall 0.583 vs 0.581.

5.2 Hyper-Parameter Tuning

Based on the learning above, we conducted a grid search on XGBoost hyper-parameters: maximum depth, learning rate, number of trees, sub sample rate, feature sample rate and L_1 penalty. The tuned model was chosen based on the best three-fold cross validation average accuracy rate. The tuned model's performance is similar to the original model. The table below compares performance of the baseline XGBoost and tuned model.

Algorithm	Train					Test				
	Accuracy	Precision	Recall	F1	AUC	Accuracy	Precision	Recall	F1	AUC
XGBoost	0.648	0.657	0.596	0.625	0.647	0.635	0.644	0.583	0.614	0.635
Tuned XGB	0.646	0.656	0.591	0.622	0.646	0.636	0.648	0.582	0.613	0.636

Our AUC score of 0.636 is only slightly lower than 0.676, which occupies the first position in Kaggle leader board for this challenge.



(a) Original XGB Confusion Matrix

(b) Tuned XGB Confusion Matrix

Figure 1: Confusion Matrix on Test Data

6 Conclusion and Future Work

On this dataset of over seven million machines and 80 features, we performed data cleaning, feature engineering and experimented different machine learning and deep learning algorithms to predict the Malware. Overall, XGBoost has the stellar and stable performance because it works well with high-dimensional data, supports regularization, detects non-linear relationship and captures correlation among the features.

It's worth noting that we experimented with a relative shallow neural network. In the future, we will experiment hyper-parameter tuning on deeper neural network via Bayesian search. More comprehensive feature engineering can also be explored.

Contribution

Utkarsh Mittal, Yun Kun Zhang and Samarpreet Singh Pandher contributed equally to idea brain storming, data preparation, experimentation and final project write-up. The final code is located in this GitHub repository: https://github.com/WilsonZhang1913/CS229-Final-Project-

References

[1] Bulao, J. (2021), *How Many Cyber Attacks Happen Per Day in 2021?* https://techjury.net/blog/how-many-cyber-attacks-per-day/gref

[2]Sokolov, Mark, Herndon, Nic, (2021), *Predicting Malware Attacks using Machine Learning and AutoAI* In Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2021), pages 295-301, https://arxiv.org/pdf/1811.11440.pdf

[3]Dobre, C. and Xhafa, F. (2014). *Intelligent services for Big Data science* Future Generation Computer Systems,37:267 – 281.

[4]Caparas, M. J. (2020). Threat Protection. docs.microsoft. com/en-us/windows/security/threat-protection/.

[5]Lin, C. (2019). *Naive Transfer Learning Approaches for Suspicious Event Prediction.*2019 IEEE International Conference on Big Data (Big Data), pages 5897–5901.

[6] Microsoft (2018). *Microsoft Malware Prediction.* https://www.kaggle.com/c/microsoft-malware-prediction/data.

[7] Benardi, Lucas.(2018). *Don't be tricked by the Hashing Trick.* https://booking.ai/dont-be-tricked-by-the-hashing-trick-192a6aae3087

[8] Jenattyon, R., Obozinski, G. and Bach, F. (2010). Structured sparse principal component analysis. *Proceedings of the Thirteen International Conference on Artificial Intelligence and Statistics, pages 366-367.* JMLR Workshop and Conference Proceedings.

[9] Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society, Series B (Methodological), 20(2), 215 – 242.*

[10] Breiman L, Friedman JH, Olshen RA, Stone CJ. (1984). *Classification and Regression Trees.* Belmont California: Wadsworth, Inc.

[11] Breiman L., (2010). Random forests. *Machine Learning, 45(1), 5-32*

[12] Freund, Y. and Schapire, R. (1997). *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*

[13] Chen, T., Guestrin, C., (2016). *XGBoost: A Scalable Tree Boosting System.* https://arxiv.org/abs/1603.02754

[14] Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". *Neural Networks.* 61: 85–117. arXiv:1404.7828

[15] Project Code: https://github.com/WilsonZhang1913/CS229-Final-Project-