Ransomware Detection Using Binary Classification

Kazi Samiul Kader[1], Md Tareque Hasan Tahsin[1], Md Shohrab Hossain[1], Husnu S. Narman[2]
[1]Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Bangladesh
[2]Department of Computer Sciences and Electrical Engineering, Marshall University, Huntington, WV, USA
Email:kazisami62@gmail.com, tarequehasan2@gmail.com, mshohrabhossain@cse.buet.ac.bd, narman@marshall.edu

***Abstract*—Nowadays ransomware attack is one of the most widely used tactics for cyber attacks. It is computationally infeasible to revert the damage done by a ransomware attack. Therefore, it is of utmost importance to identify a program to be ransomware during installation time. In this paper, machine learning binary classification algorithms have been used to identify ransomware through dynamic analysis of several features of ransomware. At first, manual selection of features is analyzed, and later on, we have used the automatic feature selection process using the K best algorithm. Results show that in both cases (manual and automatic selection), we achieved a significant percentage of accuracy to detect ransomware at runtime.**

Keywords: Ransomware, Machine Learning, Dataset, Classification, Feature Selection, K Best

I. Introduction

Ransomware is a type of malicious software designed to block access to a computer system until a sum of money is paid. It is nowadays a very common form of cyber attack. Ransomware attacks are very difficult to overturn because they usually implement state-of-the-art encryption. Hence, it is prudent to identify whether a program is a ransomware or not at installation time. This is why binary classification is necessary to detect a program to be ransomware or not. Binary classification is the task of classifying the elements of a given set into two groups (predicting which group each one belongs to) on the basis of a classification rule. In this paper, we tried to detect whether a program is ransomware or not using some machine learning classifications.

Our *motivation* comes from the fact that cyber attack has become rampant these days with the widespread use of the internet throughout the world. There are various means of attacking, and ransomware is one of the most prominent modes of attack nowadays[1]. It is to demand ransom by taking control of the resources being attacked. It is extremely expensive to revert the ransomware attack process to get the resources back[2]. If we try to classify the different types of ransomware attacks, we would find that there are two types of such attacks. One is called the locker ransomware which blocks the attacked entity from accessing the resources. Hence it is called *locker-ransomware*. Another is to encrypt the resources. Hence, the user is unable the access the files or resources. This is called *crypto-ransomware*. In both cases, the objective of the attacker is to demand money from the attacked entity.

There have been many attempts to prevent a ransomware attack. Several of them used machine learning techniques [3], [4], [5] in order to classify ransomware. Deep learning has also been used in this regard [6]. However, their results show a relatively lower success rate. They have not used to all the features that a program usually interacts with while being installed [7]. We have taken into account all those features. We used six machine learning classification algorithms with both manual and automatic feature selection. We have seen our techniques improve the successful ransomware detection rate.

Our *objective* is to dynamically analyze methods to detect ransomware. This is where machine learning techniques comes in handy to classify a program to be goodware or ransomware[8],[9],[7]. We aim to do so at the time when a program is being installed on a computer. This is because once a malicious program has been installed it is difficult to undone the damage if it is indeed ransomware.

Our *contributions* to address to detect ransomware is as follows:

- We have applied 6 machine learning classifications algorithms which are Stochastic Gradient Descent Classifier[10], Random Forest Classifier[11], K Neighbors Classifier[12], Gaussian Process Classifier[13], Multi-Layer Perceptron Classifier[14], and Ada Boost Classifier[15]
- We have applied them on the dataset as explained in [7]. In the dataset, we applied these algorithms on several features which were of mainly 7 categories.

The *novelty* of our approach lies in several folds. We have used *chi-2* test while selecting relevant features. We have analyzed both manually and automatically selected features while applying the above-mentioned six algorithms.

Our experimental *results* is mainly organized in two folds. One is for manual feature Selection. Another is for automatic feature selections based on K best algorithm. In both cases, we achieved a significant percentage of accuracy to detect ransomware at runtime. The details are mentioned in the result section.

The rest of the paper is organized as follows. In Section II, we give our proposed scheme. In Section III, we discuss the results that we have obtained. Finally, Section IV has the concluding remarks of this paper.

II. Proposed Scheme

Our proposed scheme is detailed in this section. From the observation that ransomware runtime behavior has a different pattern than benign applications. To detect the pattern, a machine learning scheme has been proposed.

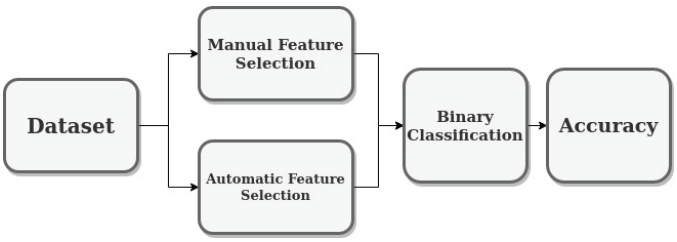

Figure 1: Proposed scheme.

As shown in Fig. 1, the proposed scheme is divided into two parts, manual feature selection, and automatic feature selection. In manual feature selection, the features in the dataset have been selected by applying domain-specific knowledge, and then the features have been fed into six binary classifiers to observe the output. In automatic feature selection, K Best (Chi 2) feature selection algorithm has been applied to extract 100 - 300 features automatically from the dataset, and then those extracted features were fed into six binary classifiers to observe the output.

For each classifier, baseline parameters have been selected to test parameter sensitivity afterward.

Binary Classifier	Baseline Parameters
Stochastic Gradient Descent	loss = hinge
Random Forest	criterion = gini
KN	weights = uniform algorithm = auto
Gaussian Process	max_iter_predict = 100
Multi- Layer Perceptron	batch_size = 200 to N tol = $1e^{-4}$ shuffle = True learning_rate_init = 0.001 power_t = 0.5
AdaBoost	base_estimator = None algorithm = SAMME.R

Table I: Baseline Parameters for Binary Classifiers.

Table I lists all the baseline parameters for all six binary classifiers used. For the Stochastic Gradient Descent classifier, the loss parameter represents a function. It is set to hinge which gives us a linear Support Vector Machine [16]. For the Random Forest classifier, the criterion parameter represents how we want to measure the purity of the decision trees inside our random forest. We used gini as our criterion [16]. For the K Neighbors classifier, the weights parameter is set to uniform which weights each neighbor equally, and the algorithm is set to auto which will choose the appropriate algorithm to select neighbors based on our given data [16]. For the Gaussian Process classifier, the max_iter_predict parameter is used to specify what is the maximum number of iterations for calculating posterior [16].

In Multi-layer perceptron, the batch size parameter is used to specify how many minibatches to use for the solver. The initial learning rate is set to 0.001. The power_t parameter is used when the solver is SGD for inverse scaling the learning rate. By setting the shuffle parameter to true, we are indicating that samples should be shuffled in each iteration if the solver is set to adam or SGD. The tol parameter sets the optimization tolerance [16]. In Ada Boost classifier, base_estimator is set to None which selects decision tree classifier, and the algorithm is set to the SAMME.R boosting algorithm [16].

A. Dataset

The dataset[3] used has 1524 rows and 30970 columns. Each row represents an application (either ransomware or goodware). Starting from the fourth column, each column represents individual operations performed by an application on the operating system and each of these columns contains binary values. As for the first three columns, the first column has a unique ID for each application row, the second column is the binary label column (1 to indicate ransomware and 0 to indicate goodware) and the third column represents a multi-class classification label (1-11 for ransomware and 0 for goodware). For binary classification purposes, the first and third columns have been removed before using the dataset for training.

Features in the dataset are divided into 7 distinct categories according to different operations an application performs on the operating system. The categories are listed below:

- API invocations
- Extension of the dropped files
- Registry key operations
- File operations
- Extensions of the file involved in file operations
- File directory operation
- Embedded strings

B. Classification on Manual Feature Selection

Six classification algorithms were fitted on the above-mentioned feature categories separately. The classification was also performed on the whole dataset.

The algorithms used for classification are listed below:

- Stochastic Gradient Descent Classifier[10]
- Random Forest Classifier[11]
- K Neighbors Classifier[12]
- Gaussian Process Classifier[13]
- Multi-Layer Perceptron Classifier[14]
- Ada Boost Classifier[15]

The widely accepted Python library scikit-learn[16] has been used for running all the classifiers.

C. Classification on Automatic Feature Selection

The focus of this stage was to reduce the dataset features for below-mentioned reasons:

- In the dataset, no of samples « no of features. It has 1524 samples and 30970 features. This will cause overfitting in data as it will be affected by the curse of dimensionality[17].
- When using the model in real-time to detect ransomware, extracting all 30970 features will not be feasible on a client machine in terms of both time and resources.
- Not all the features are necessary for classification purposes.

- Training time can be reduced significantly if features are removed which in turn enables the possibility to perform online learning in the future.

K Best algorithm from the scikit-learn library is used to reduce the number of features from 30970 to 100 – 300.

The K Best algorithm ranks all the features based on the chi-squared function [18]. This function works by computing the chi-square stats between each non-negative feature and class. By applying this algorithm, independent features can be removed which are not related to the classification task at hand. 100 – 300 highest valued features were extracted by using mentioned feature selection algorithm to feed the six binary classifiers.

D. Performance Metric

We have chosen accuracy as our performance metric. In this experiment, we want to detect whether the software is ransomware or not. From the experiments, we expect to find out what is the probability that our trained model will be able to find out if a given software is ransomware or not. Accuracy is a reliable performance metric in this scenario as it provides us with the measure of success in our test data which gives us some indication of what we can expect our model to perform in the real world.

III. Results

In this section, obtained results by applying the proposed scheme of the experiment are discussed.

A. Manual Feature Selection Result

In the first stage, six algorithms were executed on the manually selected 7 categories and also on the whole dataset.

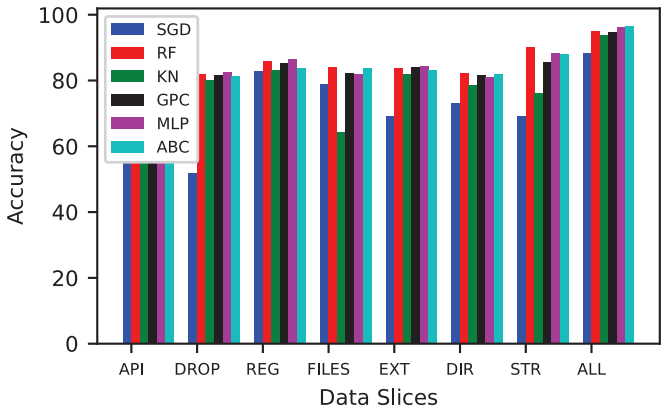

Figure 2: Results of manual feature selection.

Fig. 2 shows that the x-axis corresponds to the 7 data slices (categories) manually selected from the dataset including the whole dataset at very last. In the y-axis, the accuracy for different classifiers is shown. Colored lines indicate the six classifiers used for the experiment.

For example, it can be observed from the graph that, for API and the whole dataset, all six algorithms provide an accuracy of around 95%. For all the other categories, the average seems to be 80% accuracy.

The result over the whole dataset is indicative of overfitting by the curse of dimensionality which follows to the next stage of experimentation where the number of features is reduced to achieve a more accurate and uniform result.

B. Automatic Feature Selection Result

In the second stage, the K Best algorithm is applied to select 100 – 300 features.

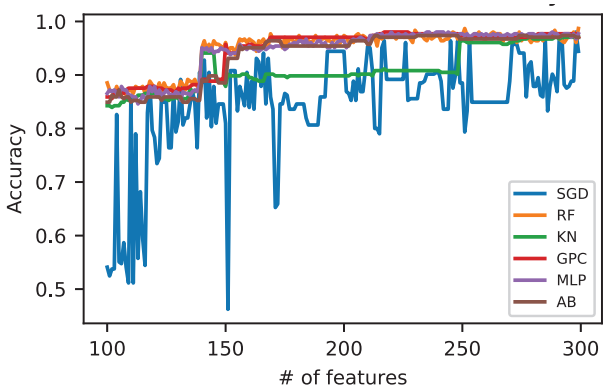

Figure 3: Results of automatic feature selection.

In Fig. 3, the x-axis corresponds to different numbers of selected features. The y-axis indicates accuracy, and the colored lines represent the six classification algorithms used for the experiment.

For example- the blue line represents the Stochastic Gradient Descent Classifier. It fluctuates from around 45% to 97% in the 100-300 feature range. Like, in around 150 features it drops down to 45% and in 298 features it achieves the accuracy of 98%. Also, the fluctuation is regular with respect to feature numbers. With feature number increase, it gains more accuracy.

The orange line represents the Random Forest Classifier. This classifier has a regular fluctuation compared to SGD classifier as its accuracy is proportional to feature increase. In the 100-300 feature range, its accuracy stays within 88-97%.

K Neighbours Classifier is indicated by the green line. From feature range 100-135, it stays around 85% accuracy. It rises in accuracy shortly around feature range 135-145 and then drops to around 88%. It stays around 88% till 275 features, then from 275-300 feature range, it achieves consistent accuracy of 97%.

The Gaussian Process Classifier starts at 84% accuracy and with feature increase, its accuracy level starts to increase as well. From 175 features to 300 features, it maintains its accuracy level of around 97%. This classifier achieves its highest accuracy at 217 features which is 98.03%.

The purple line represents Multi-layer Perceptron Classifier. This classifier starts at around 87% accuracy and jumps to 95% accuracy level at around 135 feature range. It maintains this accuracy level throughout the rest of the feature range till 300. It achieves the highest accuracy of 98% at feature 234.

The chocolate-colored line represents Ada Boost Classifier. This classifier starts at 85% accuracy and picks up its accuracy at around 160 features. It maintains accuracy of over 93% from 160 to 300 feature range. The highest accuracy of this classifier occurs at 213 features which is 97.38%.

C. AUC Over Features and Parameter Sensitivity

In this section, the AUC (Area Under Curve) value of ROC (Receiver Operating Characteristics) is tested on an automatically extracted feature range for all six binary classifiers with regards to baseline parameter sensitivity.

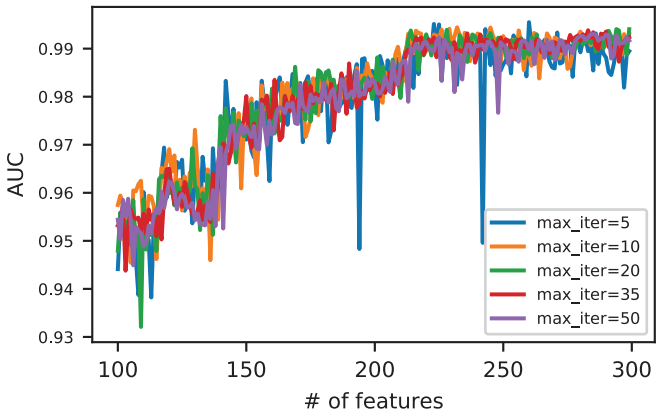

Figure 4: SGD AUC Over Features w.r.t. max_iter.

1) SGD Classifier: Fig. 4 shows the impact of increasing the max_iter parameter of the Stochastic Gradient Descent classifier on the AUC value over feature range 100-300. This parameter indicates the maximum number of passes over the training data which is sometimes referred to as epochs. We observed that for a lower number of maximum iteration like max_iter = 5 (blue line in the graph), accuracy starts over 94% and increases as features keep increasing. There is a noticeable increase in the accuracy from 94% to 97% around 125 features which drops slightly and then increases again at around 150 features.

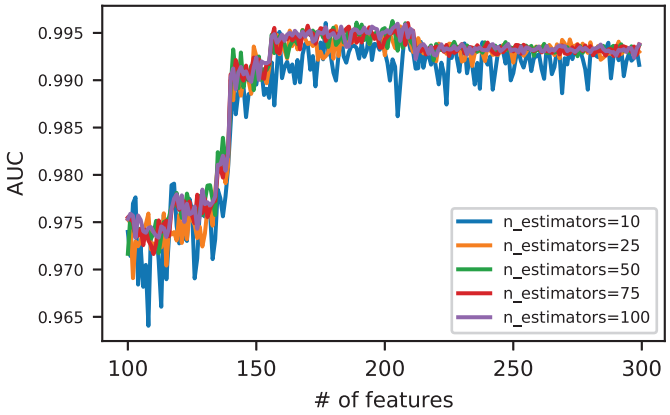

Figure 5: RF AUC Over Features w.r.t. n_estimators.

2) Random Forest Classifier: Fig. 5 shows the impact of varying the n_estimators parameter of the Random Forest classifier. This parameter indicates how many decision trees are present in the random forest. We found that for 10 decision trees, the accuracy starts slightly below 96.5% and then increases with respect to features. We observed a large increase in accuracy around 150 features which increases the accuracy close to 99.5%.

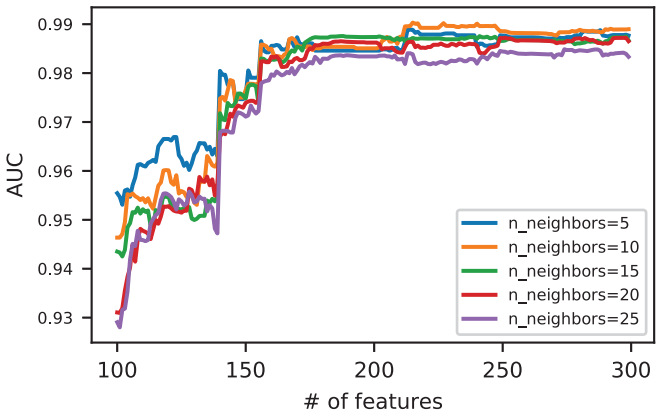

Figure 6: KN AUC Over Features w.r.t. n_neighbors.

3) KNN Classifier: Fig. 6 shows the impact of the n_neighbors parameter of the K Nearest Neighbor classifier. The n_neighbors parameter indicates the number of neighbors chosen for majority voting. This is the value of K in the K Nearest Neighbor classifier. The choice of K is very important as it will have a major impact on accuracy. We observed that for K = 5, accuracy starts above 95% and remains there till 150 features. Around 150 features, accuracy jumps to slightly over 98% and stabilizes. We observed that increasing the number of neighbors in this experiment slightly decreases accuracy overall. The impact is fairly evident at the start where we observed that K = 5 has 95.5% accuracy whereas K = 25 has 93% accuracy. This behavior may be caused by the fact that increasing K can blur the decision boundary which may impact performance negatively.

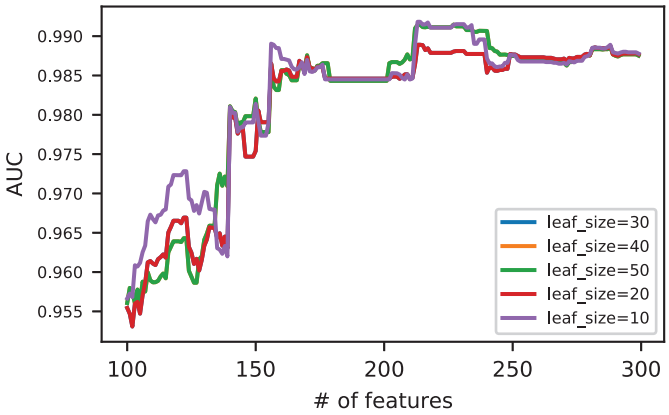

Figure 7: KN AUC Over Features w.r.t. leaf_size.

Fig. 7 shows the impact of leaf_size parameter in the K Nearest Neighbor classifier. This parameter indicates the leaf

size for the constructed tree of the model. This is an important parameter as it has an impact on the speed and query of the model. It also has an impact on how much memory is required to store the tree. For all the values of leaf size, we observed that the accuracy starts at 95.5% and increases above 97% until 130 features. Then the accuracy drops slightly above 96% and then jumps to 98% at around 150 features.

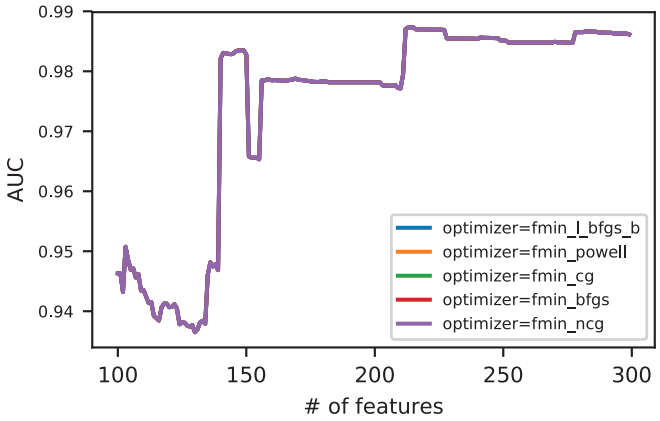

Figure 8: GPC AUC Over Features w.r.t optimizer.

4) Gaussian Process Classifier: Fig. 8 shows the impact of choosing five different optimizers on accuracy in the Gaussian Process Classifier. This parameter represents different functions that optimize the internal parameters of the GPC model. We have observed that all the functions produce the same accuracy. All of them start above 94% accuracy and the accuracy declines until around 140 features. After 140 features the accuracy jumps above 98% and steadily increases with respect to features and finally stabilizes at 99.5%.

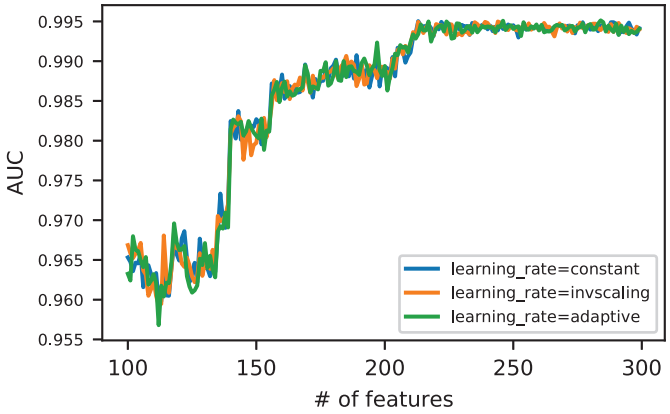

Figure 9: MLP AUC Over Features w.r.t. learning_rate.

5) MLP Classifier: Fig. 9 shows the impact of learning rate on accuracy across features. This parameter schedules the initial learning rate. There are three types of schedule provided by the scikit-learn library [16]. Constant uses the initial given learning rate at each time step. Invscaling gradually decreases the learning rate and adaptive changes the learning rate based on training loss. We observed that each of these learning rate schedulers has the same impact on accuracy. For each one of them, accuracy remains from 95.5% to 96.5% until 140 features. After that accuracy jumps around 98.5% and then slowly increases with features until getting stabilized at 99.5% accuracy.

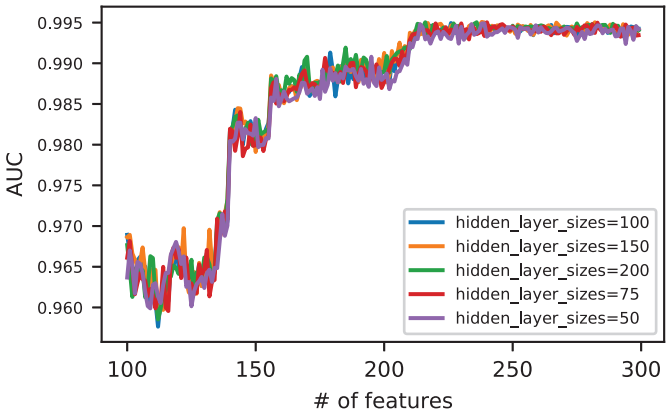

Figure 10: MLP AUC Over Features w.r.t. hidden_layer_sizes.

Fig. 10 shows the impact of hidden layer sizes on accuracy across features. A neural network typically consists of one hidden layer and the size of the hidden layer has an important impact on accuracy. There are many conventional rules of thumb on deciding the size of the hidden layer. One such rule is that the hidden layer size should be the mean of input and output layer and another such rule states that hidden layer size should be two-thirds of the input layer. Our input layer consists of around 900 samples and we have 2 neurons in the output layer. So according to the first rule, our hidden layer size should be 450 and according to the second rule, our hidden layer size should be 600. In this experiment, we wanted to find out if we can achieve high accuracy by using a lesser number of neurons in the hidden layer than the suggested conventional number and thus saving space. We have used five different hidden layer sizes, like - 50, 75, 100, 150, 200. For each one of them, accuracy remains from 97% to 96% until 140 features. After that accuracy jumps around 98.5% and then slowly increases with features until getting stabilized at 99.5% accuracy.

6) Ada Boost Classifier: Fig. 11 shows the effect of the learning rate parameter in the Ada Boost classifier. Learning rate assigns the weight to each classifier. We observed that the impact of this parameter consists of sharp increases in accuracy score and then plateaus for some time. In its steady growth, it maintains this trend of sharp jumps and short plateaus. It has a very promising AUC score of over 100-300 features as it always stays from AUC value 0.94 - 0.99 and in the end crosses 0.99 around 200 features. With respect to different learning rates at around 300 features a learning rate of 0.75 triumphs other learning rates.

Fig. 12 shows the impact of n_estimators parameter on accuracy. This parameter determines how many weak learners or stumps (decision tree with only one node and two leafs) are

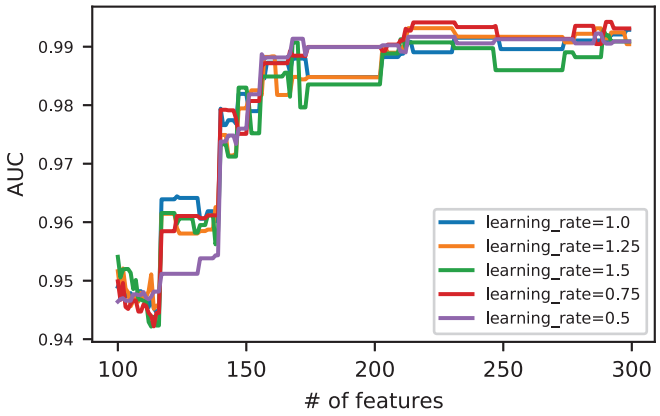

Figure 11: AB AUC Over Features w.r.t. learning_rate.

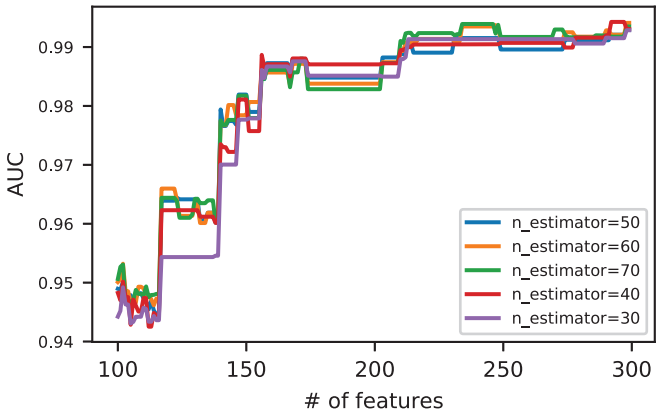

Figure 12: AB AUC Over Features w.r.t. n_estimator.

there in the model. The AB classifier shows steady growth with sharp jumps and plateau trend just like Fig. 11. With regards to the number of estimators, it does not show much change in accuracy.

IV. Conclusion and Future Works

In this paper, we have shown that it is possible to detect ransomware using machine learning algorithms with a fair amount of accuracy. As the feature number is limited to below 300 features, it would take a very short amount of time to detect ransomware using this technique in client machines. We have used various classifiers and feature selection algorithms to detect ransomware applications and evaluate their effectiveness by using an established dataset. The accuracy level was improved by 1% by using a different feature selection algorithm (chi-square test) instead of the Mutual Information criterion.

In the future, we plan to use deep learning classification such as CNN to improve this result further. Also, new features could be considered such as network communication performed by an application during their runtime to predict ransomware more accurately.

References

[1] N. Biasini, J. Esler, *et al.*, "Threat spotlight: Cisco talos thwarts access to massive international exploit kit generating 60m annually from ransomware alone." http://www.talosintel.com/angler-exposed, 2015.

[2] T. Group, "Teslacrypt - decrypt it yourself." http://blogs.cisco.com/security/talos/teslacrypt, 2015.

[3] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu, "Automated dynamic analysis of ransomware: Benefits, limitations and use for detection," *arXiv preprint arXiv:1609.03020*, 2016.

[4] R. Vinayakumar, K. Soman, K. S. Velan, and S. Ganorkar, "Evaluating shallow and deep networks for ransomware detection and classification," in *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, (Karnataka, India), pp. 259–265, IEEE, 2017.

[5] K. Cabaj, M. Gregorczyk, and W. Mazurczyk, "Software-defined networking-based crypto ransomware detection using http traffic characteristics," *Computers & Electrical Engineering*, vol. 66, pp. 353–368, 2018.

[6] D. V. Sang, D. M. Cuong, and L. T. B. Cuong, "An effective ensemble deep learning framework for malware detection," in *the Ninth International Symposium on Information and Communication Technology*, (Danang, Vietnam), pp. 192–199, ACM, 2018.

[7] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[8] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *Journal in computer Virology*, vol. 7, no. 4, pp. 247–258, 2011.

[9] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, vol. 7, no. Dec, pp. 2721–2744, 2006.

[10] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *COMPSTAT*, pp. 177–186, Springer, 2010.

[11] T. K. Ho, "Random decision forests," in *3rd international conference on document analysis and recognition*, vol. 1, (Montreal, Quebec, Canada), pp. 278–282, IEEE, 1995.

[12] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[13] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced lectures on machine learning*, pp. 63–71, Springer, 2004.

[14] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[15] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine learning*, vol. 37, no. 3, pp. 297–336, 1999.

[16] F. Pedregosa, G. Varoquaux, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[17] R. E. Bellman, *Adaptive control processes: a guided tour*, vol. 2045. Princeton university press, 2015.

[18] H. Liu and R. Setiono, "Chi2: Feature selection and discretization of numeric attributes," in *7th IEEE International Conference on Tools with Artificial Intelligence*, (Herndon, VA, USA), pp. 388–391, IEEE, 1995.