Malware Detection System

Zihan Qu (zqu6@jh.edu)
Zhen Liang (zliang30@jh.edu)
Kepeng Zhou (kzhou15@jh.edu)

1 Background

1.1 Malware Detection

Malware stands for malicious software, which is designed to infiltrate, damage, or exploit computer systems, networks, or devices without the user's consent. It continues to evolve, adopting increasingly sophisticated techniques to evade detection and removal, and covers a wide range of threats. The three main types of malware are as follows:

Ransomware: It has become increasingly prevalent in recent years, targeting not only individuals but also large organizations, including healthcare providers, educational institutions, and government agencies. Its ability to encrypt crucial data and demand ransom payments can cause significant financial and operational disruptions. To counter this threat, it is essential to regularly back up data, use strong security measures such as encryption and multi-factor authentication, and keep software up-to-date.

Trojans: It exploit users' trust in seemingly legitimate applications to infiltrate systems and perform malicious activities, often remaining undetected for extended periods. To protect against trojans, users should exercise caution when downloading software, only install applications from reputable sources, and use robust antivirus and anti-malware software that can detect and remove these deceptive threats.

Spyware: It poses a significant risk to personal privacy, as it can capture sensitive information, such as login credentials, financial data, and browsing habits, without the user's knowledge.

To mitigate the risk of spyware, users should be cautious when clicking on links or downloading attachments from unknown sources, employ privacy-enhancing browser extensions, and utilize reliable security software that can detect and eliminate spyware.

Malware detection is crucial for individuals and organizations to protect their digital assets and maintain security and privacy. Given the constantly evolving nature of cyber threats, a multi-layered approach to malware detection is essential. This involves combining various detection techniques. Analyzing software activity is one crucial technique.

1.2 CIC-MalMem-2022 Dataset

For the goal of detecting malware through software activity, we chose CIC-MalMem-2022 as the dataset to develop our solution. The CIC-MalMem-2022 dataset is a comprehensive collection of obfuscated malware samples. This dataset is collected based on the memory

dump of software. Some of the features are generated by analysis tools like Volatility Framework. The dumps are generated utilizing debug mode, to minimize the impact of the dumping process on the memory dumps. And part of the data is made up by simulating the dump of malware to create a well-balanced dataset.

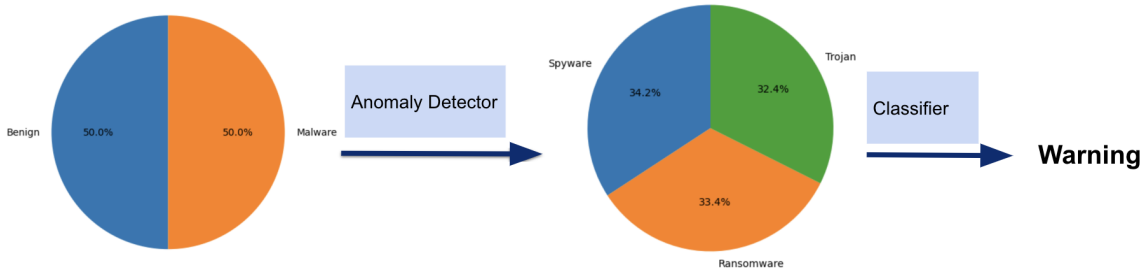
The dataset contains a total of 58,596 records with 29,298 benign and 29,298 malicious. For malicious cases, they are divided into three categories (Trojan Horse, Spyware, and Ransomware). And each Category is divided into 5 families. The detailed breakdown is as follows.

Malware category	Malware families
Trojan Horse	• Zeus • Emotet • Refroso • scar • Reconyc
Spyware	• 180Solutions • Coolwebsearch • Gator • Transponder • TIBS
Ransomware	• Conti • MAZE • Pysa • Ako • Shade

1.3 Problem Definition

Since multi-layered approaches are adopted for the most advanced malware detection system, and the dataset has labels for benign and malicious classes and also malicious categories, we should build a system monitoring software utilizing fewer resources and rise warnings with malware types once the malware was detected.

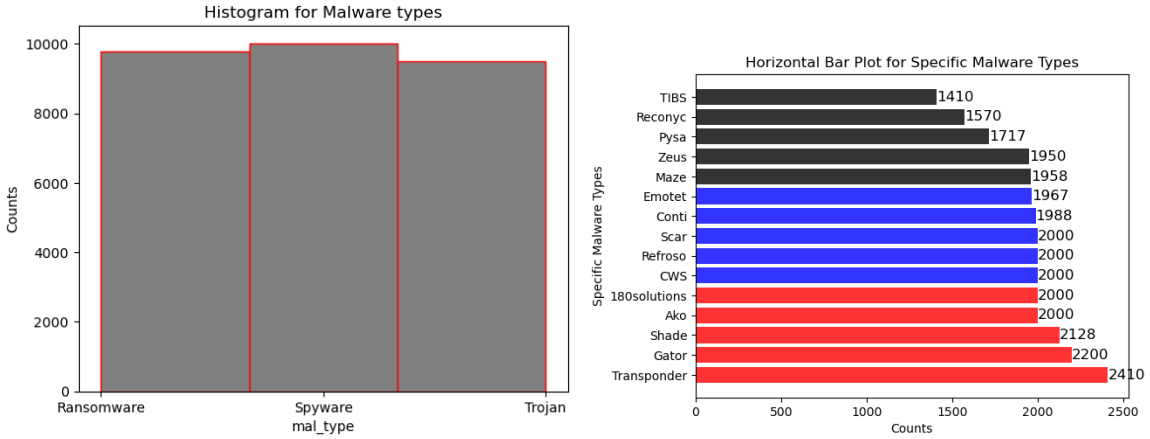
However, on the classification task, models with less computational complexity do not perform well, and the larger models, like LSTM XGBoost, are too complex for monitoring tasks. And the performance when the benign cases are included is worse than just classifying malicious cases. Thus, our solution is to build an abnormal detection model with less complexity as the monitor model. And we also build a classifier to classify the detected malware that only runs when malware is detected.



3 Methodology

3.1 Data Analysis

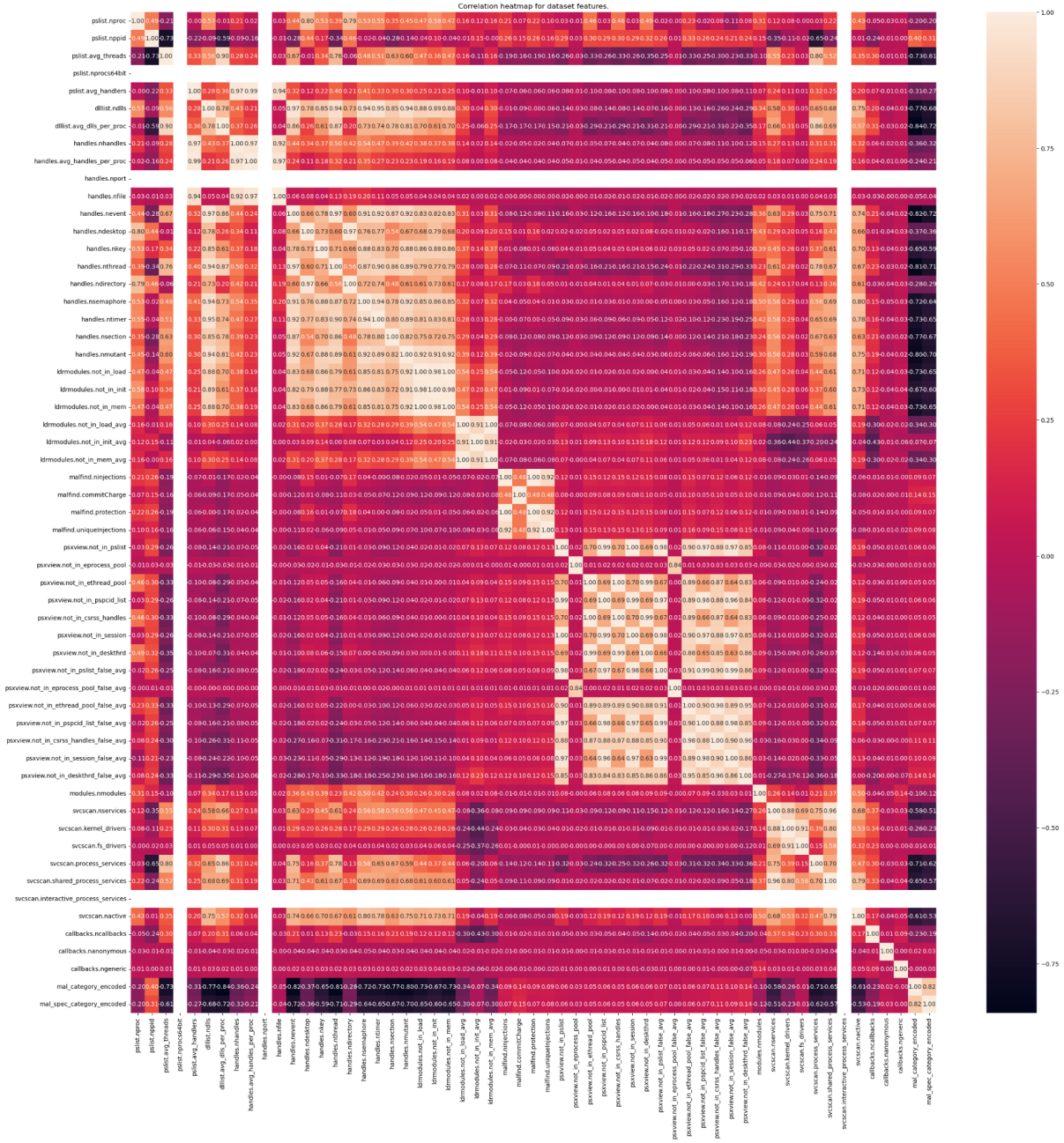
Preparation: The very first step in our data analysis is to get the dataset ready for further processes such as anomaly detection, malware classification, and result visualization. To complete this part, we can take a quick look at the data distribution.



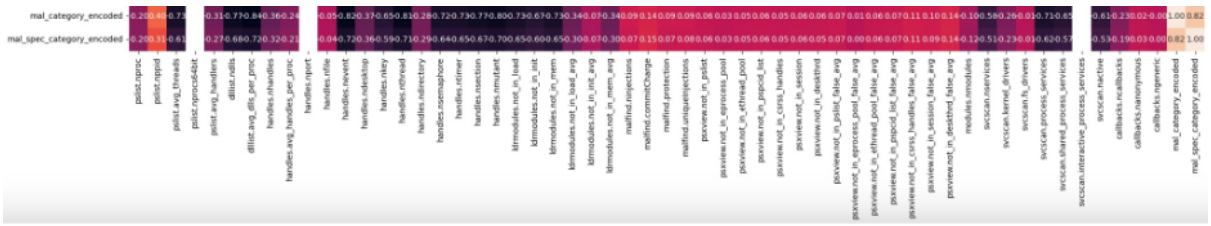
As we mentioned above, the dataset provides a label for each row of data, which indicates whether it is ‘Benign’ data or ‘Malware’ data. However, each ‘Malware’ data has a unique label. The first two words in the ‘Malware’ label indicate what type of malware it belongs to. (e.g., Ransomware-Shade-f0…) After checking the value of ‘Malware’ label data, we notice that there are 3 different values of the malware category and 15 values of the specific malware category that belongs to the previous malware category. Therefore, we split the malware code into mal_category and mal_spec_category which represent malware category and specific malware category respectively. And for better performance of the future classification process, we decide to encode mal_category and mal_spec_category values, since many classification models can only handle values in numerical form.

Feature selection: we apply two methods to feature selection, the first is based on correlation between features and target vectors using heatmap, the other method is using CFS algorithm. The reason for feature selection is that for each row of data in our dataset, there are too many features. Some of these features have low connection and correlation with our target vector, which could lead to overfitting and increased computational complexity if we use too many of them.

Below is the heatmap that represents the correlation between each feature column in the dataset and the two target vectors that we choose (mal_category and mal_spec_category).



According to the heatmap, we notice most of the features do not have highly positive correlation with chosen target vectors. But there are several features that have highly negative correlation with the two target vectors. Hence, we can create a new dataset with only these important features (Correlation coefficient value between -0.5 and -1.0. The darker the color is, the higher negative correlation it has with target vectors) for future classification. Notice that some of the row and columns are blank, which is because these values are constant throughout the dataset and make it impossible to compute the correlation coefficient.



As for CFS algorithm, the mathematical formula is like this:

$$Merit_s = \frac{k\overline{r_{cf}}}{\sqrt{k+k(k-1)\overline{r_{ff}}}}.$$

In the paper [1], the algorithm analyzes the correlation coefficient between feature and feature, and the relationship between feature and classification. It analyzes 2^{56} combinations of feature subsets. In this formula, *k* is the number of features. The output of this algorithm gives the top best features, and we use 7 here such that it indicates the top 7 best features.

3.2 Anomaly Detection

3.2.1 Model Selection

As discussed in the 3.1 Data Analysis section, the benign data and the malicious data have equal amounts. So unlike the typical abnormal detection scenario, this dataset is a well-balanced dataset with well-labeled anormaly cases. So if we use abnormally detection models like GMM and HMM to fit the normal cases, the abnormal cases will be wasted. So, we chose to perform binary classification on this dataset.

Besides, the abnormal detector will be running as a monitor service in the real scenario. So we assumed that the detector has to have low computational complexity.

According to these two requirements, we chose a set of models with low computational requirements:

1. **Logistic Regression**: Logistic Regression is chosen for its simplicity and fast training process. It models the probability of an instance belonging to a class using a logistic function. Inference complexity is O(n), where n is the number of features.
2. **Random Forest**: Random Forest is an ensemble method based on decision trees, which provides good accuracy and handles high dimensional data well. However, it may not be ideal for this case due to its higher computational complexity during inference (O(T*depth), where T is the number of trees and depth is the maximum depth of the trees).
3. **Support Vector Machines (SVM)**: Support Vector Machines (SVM) can perform well in binary classification tasks, but may not be suitable for the requirements as it has higher computational complexity during training and inference (O(n * m), where n is the number of support vectors and m is the number of features).

4. **AdaBoost**: AdaBoost is an ensemble method that builds a strong classifier from a series of weak learners. It has relatively low complexity during inference (O(T*m), where T is the number of weak learners and m is the number of features).
5. **k-Nearest Neighbors (k-NN)**: k-Nearest Neighbors (k-NN) is a non-parametric, lazy learning method that can provide good performance in binary classification tasks. Its inference complexity is O(n * m), where n is the number of training samples and m is the number of features. k-NN's complexity may be higher than other models.
6. **Naive Bayes**: Naive Bayes is a simple probabilistic classifier based on applying Bayes' theorem, assuming independence between features. It is chosen for its simplicity and low inference complexity (O(n*m), where n is the number of classes and m is the number of features).
7. **Decision Tree**: Decision Trees are easy-to-understand, interpretable models that perform well in binary classification tasks. Their inference complexity is O(depth), where depth is the maximum depth of the tree. Decision Tree might be a suitable model since it is relatively computationally efficient.

3.2.2 Training

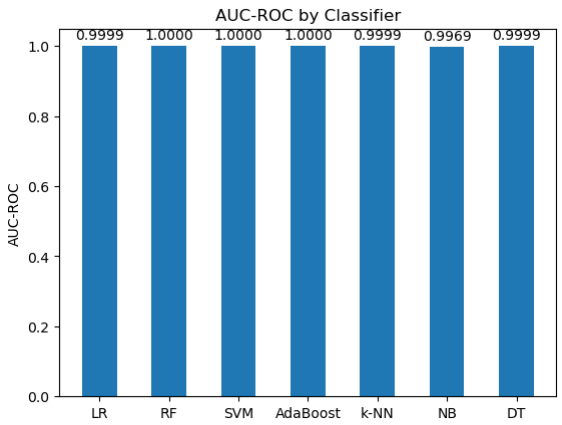
Since the dataset is balanced, the three-way cross-validation will not suit this dataset. So we split the trainset and testset as 7:3. And fit every model on the trainset. For the random forest and decision tree models, we fine-tuned the model with different parameters(n_estimator, criterion("gini" and "entropy"), and max_depth for random forest, and max_depth and criterion("gini" and "entropy") for decision trees).

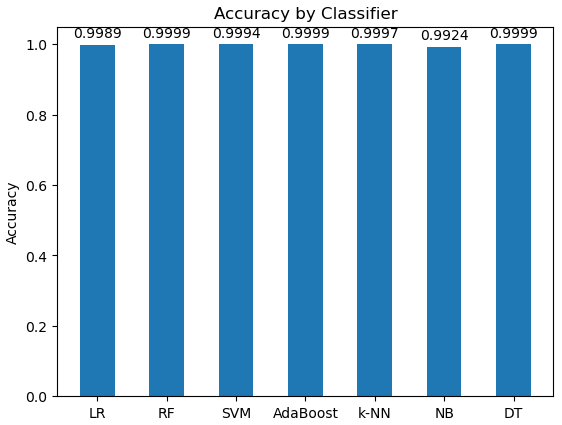
3.2.3 Evaluation

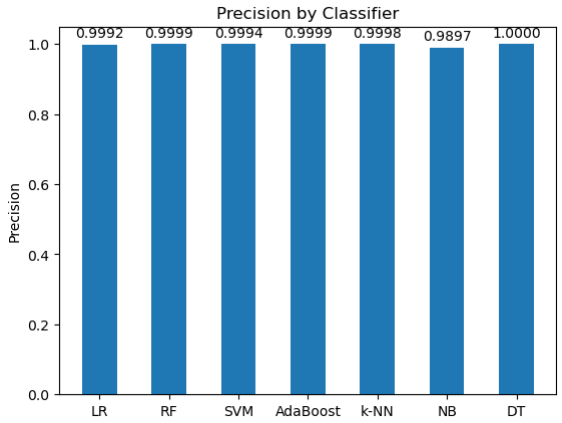
For the metrics, we use a set of metrics we learned in class and also use Cohen's Kappa and Matthews Correlation Coefficient to compare models better. Cohen's Kappa Coefficient evaluates classification agreement, accounting for chance agreement. It's useful when dealing with imbalanced datasets or high accuracy, as it provides a more robust performance measure compared to accuracy alone, which can be misleading in such cases. The Matthews Correlation Coefficient (MCC) is a performance metric for binary classification that considers true and false positives and negatives. It ranges from -1 to 1, where 1 indicates perfect prediction, 0 suggests no better than random chance, and -1 represents complete disagreement. MCC provides a balanced measure even for imbalanced datasets, making it more informative than accuracy in cases where class distribution is skewed.
The metrics we adopted are as follows:

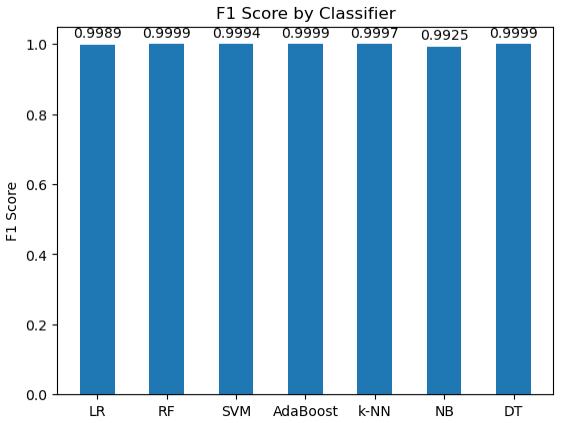
1. Accuracy
2. AUC-ROC
3. Precision
4. Recall
5. F1 score
6. Matthews Correlation Coefficient
7. Cohen's Kappa Coefficient.

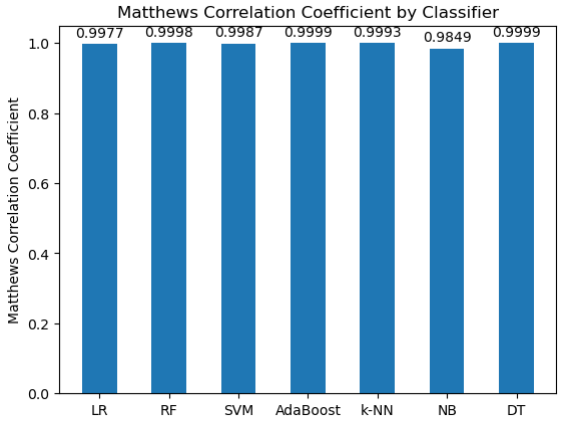
Also, we evaluate whether the models are efficient by their Inference Time. It is reflected by the test time on the test set(
Here are the evaluation results for each model:

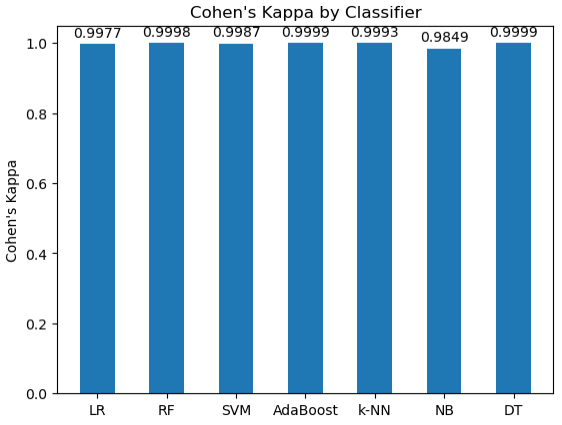

AUC-ROC by Classifier
AUC-ROC
0.9999
1.0000
1.0000
1.0000
0.9999
0.9969
0.9999
LR
RF
SVM
AdaBoost
k-NN
NB
DT

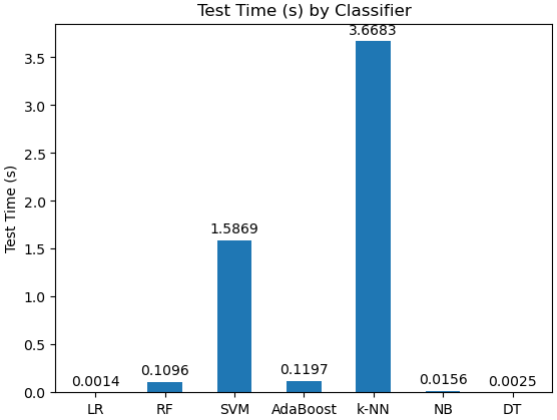

Accuracy by Classifier
Accuracy
0.9989
0.9999
0.9994
0.9999
0.9997
0.9924
0.9999
LR
RF
SVM
AdaBoost
k-NN
NB
DT


Precision by Classifier
Precision
0.9992
0.9999
0.9994
0.9999
0.9998
0.9897
1.0000
LR
RF
SVM
AdaBoost
k-NN
NB
DT


F1 Score by Classifier
F1 Score
0.9989
0.9999
0.9994
0.9999
0.9997
0.9925
0.9999
LR
RF
SVM
AdaBoost
k-NN
NB
DT


Matthews Correlation Coefficient by Classifier
Matthews Correlation Coefficient
0.9977
0.9998
0.9987
0.9999
0.9993
0.9849
0.9999
LR
RF
SVM
AdaBoost
k-NN
NB
DT


Cohen's Kappa by Classifier
Cohen's Kappa
0.9977
0.9998
0.9987
0.9999
0.9993
0.9849
0.9999
LR
RF
SVM
AdaBoost
k-NN
NB
DT

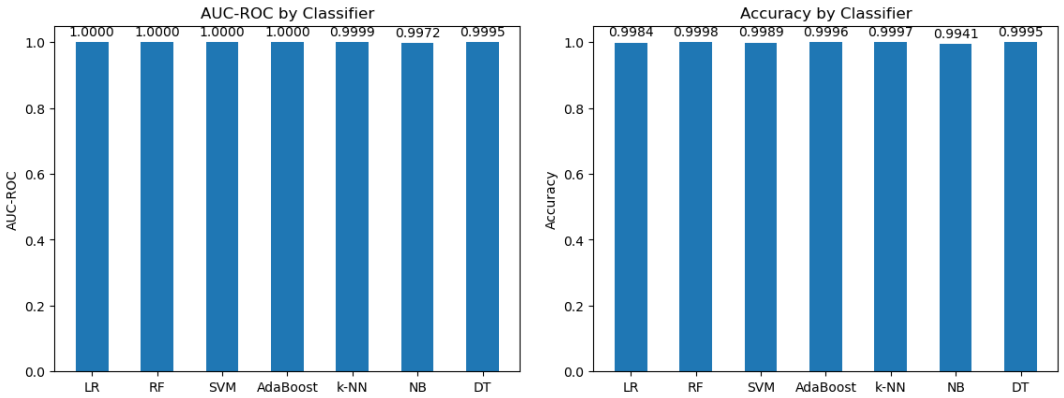

Test Time (s) by Classifier
Test Time (s)
0.0014
0.1096
1.5869
0.1197
3.6683
0.0156
0.0025
LR
RF
SVM
AdaBoost
k-NN
NB
DT

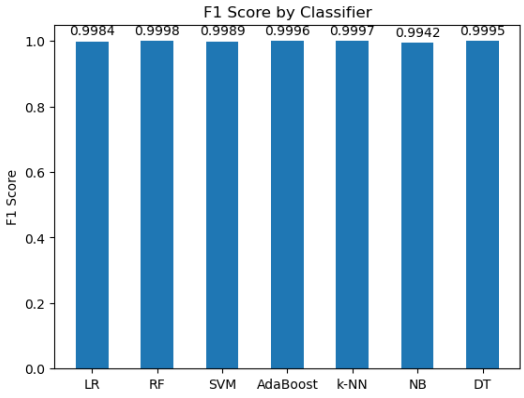
In the evaluation results, for most metrics, the models achieved great results(over 99.9%). So, we can not choose the model based on the slight difference in performance.
In the last chart, the inference time of models varies a lot. The models that achieved the best performance are logistic regression and decision tree.

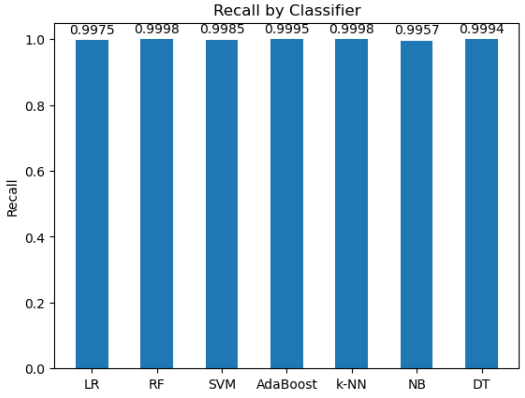
3.2.4 Selected Feature

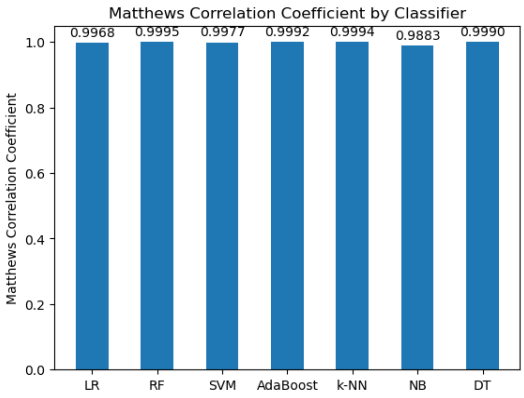
Since the models' performance is good enough on all features, we try to use the top seven selected features to reduce the model complexity. The feature selection process has been explained in the Data Analysis section.

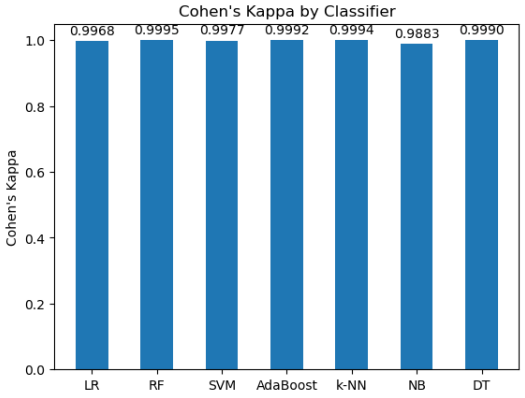
The performance of models does not drop a lot after reducing the features, meanwhile, the speed of inference improved significantly.
The results are as follows:

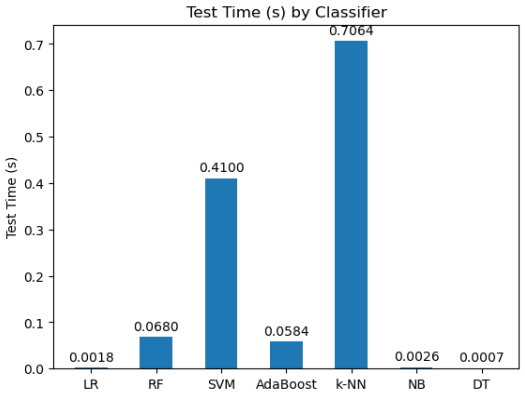












3.2.5 Conclusion: Decision Tree

After evaluation of models on all features and 7 selected features, we chose the Decision Tree as the best model for this task for two reasons:

1. Low computational conception.
2. High performance on selected features.

3.3 Malware Classification

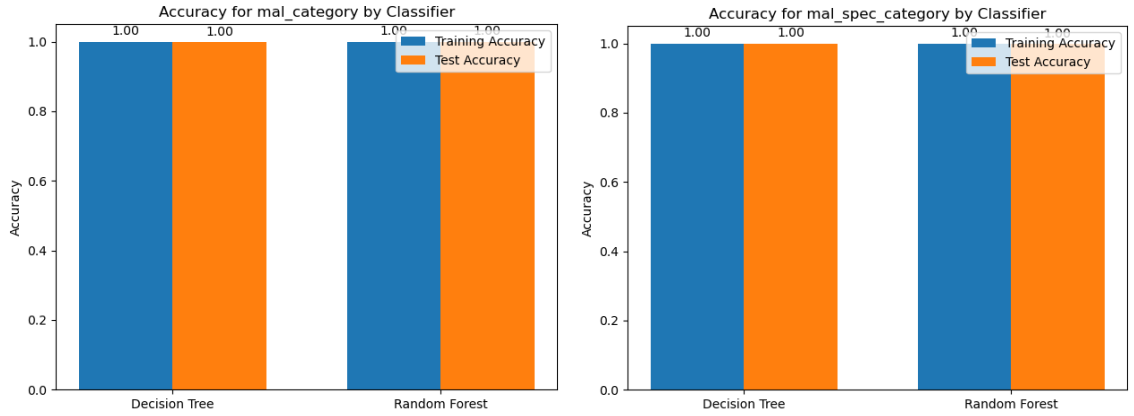
In this section, we apply various classification and compare their results in classification accuracy. As for the feature selection in this section, we mainly use two sets of features, which are extracted by the two methods we mentioned above. We use the feature selection mechanism based on CFS, correlation coefficient index, and full feature sets. The details of which columns we use are listed in the appendix[1].

3.3.1 DT and RF

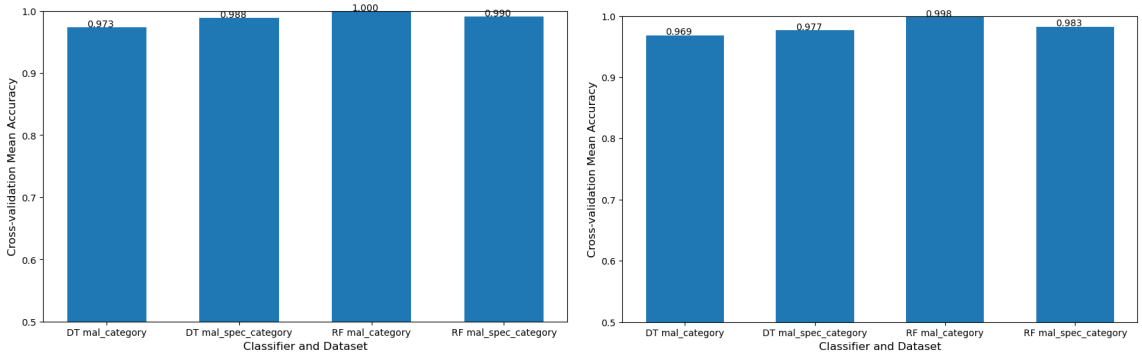
We applied Decision Tree and Random Forest to Malware classification to categorize malware and specific malware categories. Before the application of these two methods, we thought they could achieve intermediate accuracy in prediction of mal_category and mal_spec_category, and possibly finish with higher accuracy in mal_category prediction. And we thought Random Forest might have better performance on accuracy prediction than Decision Tree. We use these two classification methods in several different but related scenarios, such as use the entire dataset and all features, use only malware dataset and all features, and use only malware dataset and selected features. When we use all features, we tend to get very similar results in prediction accuracy even with different datasets. However, when we set our dataset to be the malware dataset, and alter the selected features, prediction results fluctuate with the corresponding features we selected. We use two selected feature groups in this section. One is features that have highly negative correlation with target vectors, the other is calculated by CFS algorithm. The two feature groups have different numbers of features, and in later processes, they lead to very various accuracy

[1] The second appendix is about feature selection in malware classification and anomaly detection part

outputs. When we use all features from the dataset, the prediction accuracy in Training and Testing sets are both close to 100%.

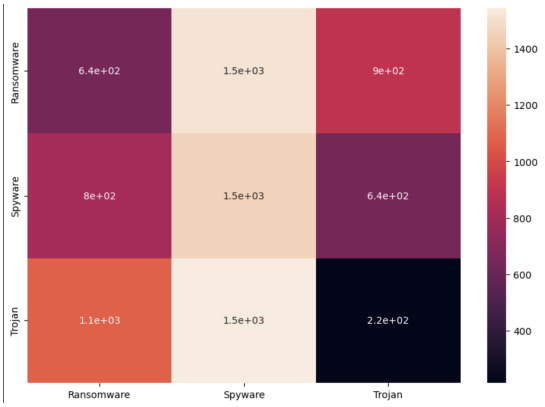


We used cross validation to double check the result for above results.



3.3.2 LSTM

We implemented LSTM on the malware classification only for the malware general classification among spyware, ransomware, and trojan. We thought that the deep learning model might have better performance specially in the long sequence, since we have 15 different types need to classify. However, we tried several combinations of feature selection, including but not limited to the CFS feature selection, all feature sets, top 16 correlation-coefficient from heatmap, etc, the classification output is not as good as we expected. The probable reason that led to the poor performance should be that even though we have about 24k training data, it is still far from “sufficient level” to feed the LSTM layers. So no matter what we change our feature selection, the accuracy will not increase a lot.



3.3.3 CART (Classification and Regression Tree)

The CART algorithm is a classification technique used to construct a decision tree based on Gini's impurity index. It is a fundamental machine learning algorithm with many applications. It consists of two categories: the classification tree and the regression tree. Classification trees are used to discover the class that is most likely to fall when the target variable is continuous, and regression trees are used to predict these variable's value.

In this section, we use the cart algorithm to classify both malware types (i.e. ransomware, trojan, and spyware), and also the specific malware types (i.e. ako, cws, etc.). Also, we use all feature sets, the correlation-coefficient feature selection according to heatmap, and feature selection of the CFS algorithm. So we include three versions of code which represent how the model performs in all feature sets and in the selected features.

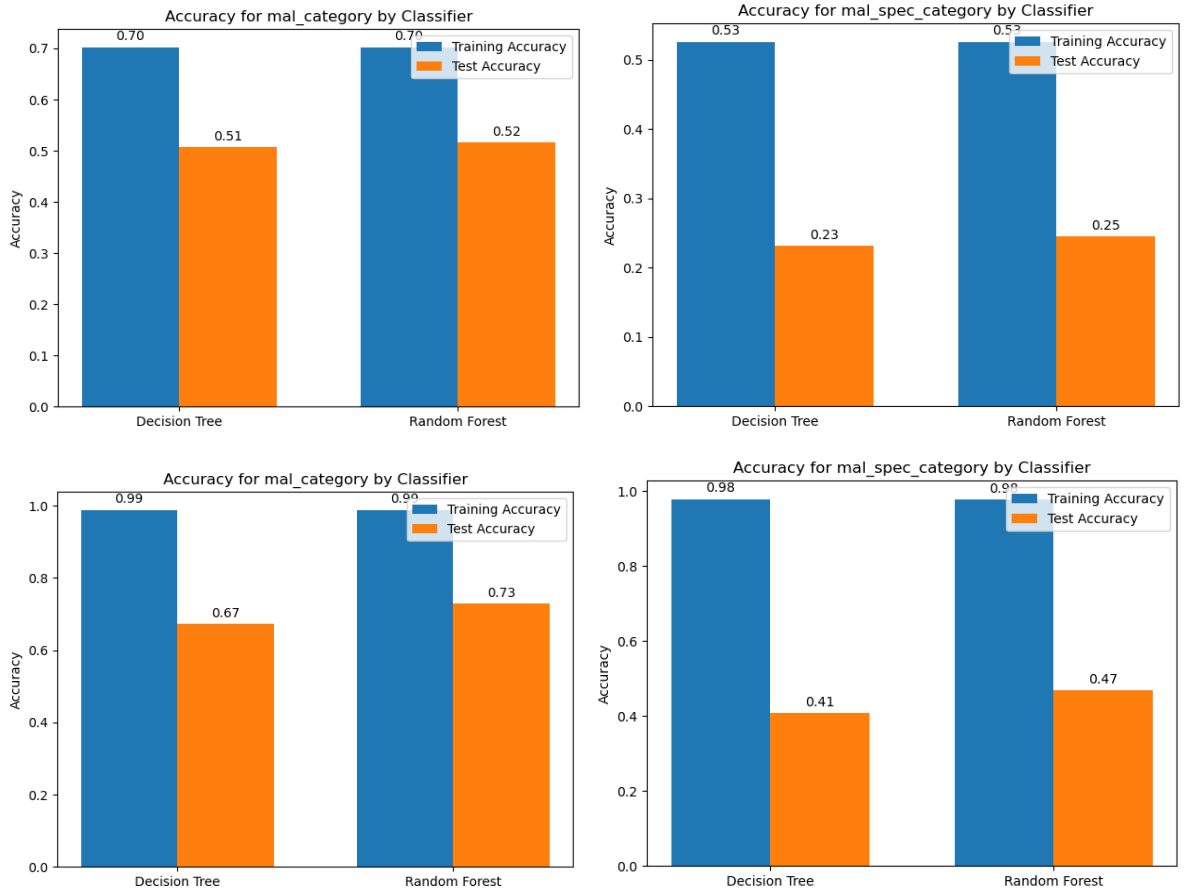
3.3.4 XGBoost

However, the CART algorithm has a poor performance on both specific and non-specific classification, both accuracy (best accuracy regards of feature selection) is less than 60%. Since the disadvantages of the CART algorithm have problems about the overfitting, sparse predictions, and high variance. XGBoost, in contrast, explicitly adds a regular term to control the complexity of the model, which helps prevent overfitting and thus improves the generalization ability of the model. XGBoost builds an ensemble of decision trees in a sequential manner, with each tree learning to correct the mistakes of its predecessors. This aims to minimize a loss function, which measures the difference between the predicted and true labels. Under this circumstance, we want to use the XGBoost model to overcome disadvantages in the CART algorithm. Same as above, we use the same feature selection mechanism to train the model as in CART.

4 Discussion and Results

4.1 DT and RF

As mentioned before, we notice differences between prediction accuracy under various situations. And when we switch to selected features, the results are not even close to 100%. The consequence applied to both Decision Tree and Random Forest.

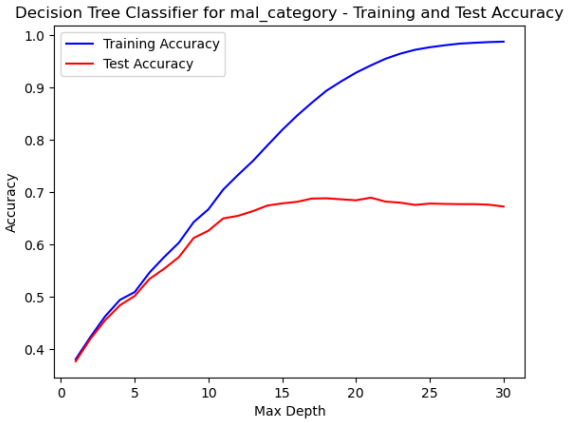


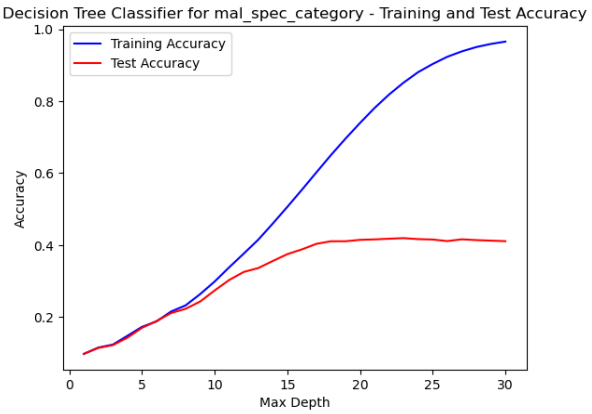
As shown in the pictures, both Decision Tree and Random Forest have better performances on mal_category prediction than mal_spec_category prediction accuracy. Random Forest has a slight advantage over Decision Tree in accuracy prediction.
Despite that, when we use malware dataset and selected features, the prediction result was obviously overfitting, as the training accuracies are significantly higher than testing accuracies. We set different parameters for Decision Tree and Random Forest, especially max_depth, which control the longest path between the root node and the leaf node.

```
----------------------------------------
Max depth: 6
Accuracy:  0.5340159271899886
Training Accuracy DT for mal_category (selected features / malware dataset):  0.5460795787009948
Testing Accuracy DT for mal_category (selected features / malware dataset):  0.5340159271899886
----------------------------------------
Max depth: 7
Accuracy:  0.5534698521046644
Training Accuracy DT for mal_category (selected features / malware dataset):  0.5758728301150771
Testing Accuracy DT for mal_category (selected features / malware dataset):  0.5534698521046644
----------------------------------------
Max depth: 8
Accuracy:  0.575881683731513
Training Accuracy DT for mal_category (selected features / malware dataset):  0.6033742929588454
Testing Accuracy DT for mal_category (selected features / malware dataset):  0.575881683731513
----------------------------------------
Max depth: 9
Accuracy:  0.6122866894197952
Training Accuracy DT for mal_category (selected features / malware dataset):  0.6426272674078408
Testing Accuracy DT for mal_category (selected features / malware dataset):  0.6122866894197952
----------------------------------------
Max depth: 10
Accuracy:  0.6261660978384528
Training Accuracy DT for mal_category (selected features / malware dataset):  0.6671055197971524
Testing Accuracy DT for mal_category (selected features / malware dataset):  0.6261660978384528
----------------------------------------
Max depth: 11
Accuracy:  0.6496018202502845
Training Accuracy DT for mal_category (selected features / malware dataset):  0.704651843183148
Testing Accuracy DT for mal_category (selected features / malware dataset):  0.6496018202502845
----------------------------------------
Max depth: 12
Accuracy:  0.6544937428896473
Training Accuracy DT for mal_category (selected features / malware dataset):  0.7325433976984591
Testing Accuracy DT for mal_category (selected features / malware dataset):  0.6544937428896473
----------------------------------------
Max depth: 13
Accuracy:  0.6633674630261661
Training Accuracy DT for mal_category (selected features / malware dataset):  0.7589721084454847
Testing Accuracy DT for mal_category (selected features / malware dataset):  0.6633674630261661
```

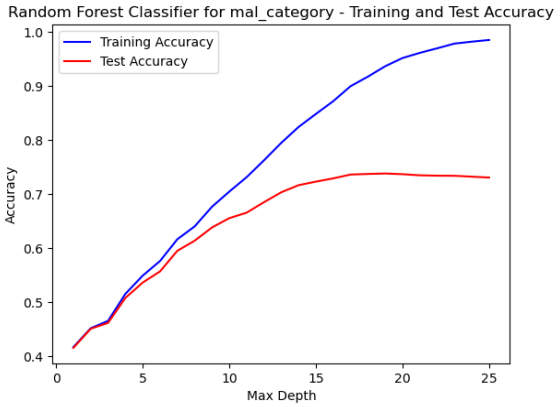
```
----------------------------------------
Max depth: 9
Accuracy:  0.24323094425483505
Training Accuracy DT for mal_spec_category (selected features / malware dataset):  0.26384825433976988
Testing Accuracy DT for mal_spec_category (selected features / malware dataset):  0.24323094425483505
----------------------------------------
Max depth: 10
Accuracy:  0.2740614334470999
Training Accuracy DT for mal_spec_category (selected features / malware dataset):  0.2989077433196801
Testing Accuracy DT for mal_spec_category (selected features / malware dataset):  0.2740614334470999
----------------------------------------
Max depth: 11
Accuracy:  0.3031854379977247
Training Accuracy DT for mal_spec_category (selected features / malware dataset):  0.3384045250633899
Testing Accuracy DT for mal_spec_category (selected features / malware dataset):  0.3031854379977247
----------------------------------------
Max depth: 12
Accuracy:  0.3254835039817975
Training Accuracy DT for mal_spec_category (selected features / malware dataset):  0.3764384630388141
Testing Accuracy DT for mal_spec_category (selected features / malware dataset):  0.3254835039817975
----------------------------------------
Max depth: 13
Accuracy:  0.33629124004550626
Training Accuracy DT for mal_spec_category (selected features / malware dataset):  0.4152038228983811
Testing Accuracy DT for mal_spec_category (selected features / malware dataset):  0.33629124004550626
----------------------------------------
Max depth: 14
Accuracy:  0.3559726962457338
Training Accuracy DT for mal_spec_category (selected features / malware dataset):  0.4600156036668617
Testing Accuracy DT for mal_spec_category (selected features / malware dataset):  0.3559726962457338
----------------------------------------
Max depth: 15
Accuracy:  0.37497155858930603
Training Accuracy DT for mal_spec_category (selected features / malware dataset):  0.5063877511215136
Testing Accuracy DT for mal_spec_category (selected features / malware dataset):  0.37497155858930603
----------------------------------------
```

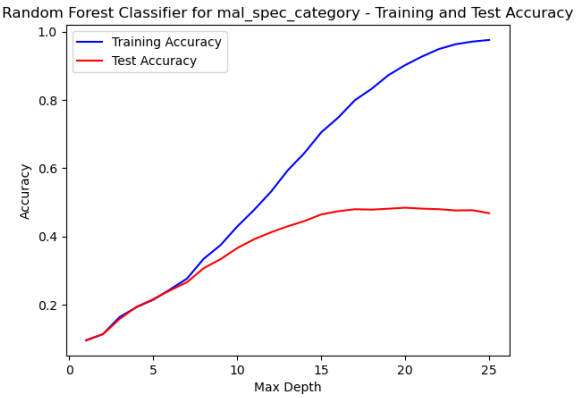
The pictures below show how max_depth (range from 0 to 30) affects prediction accuracy for Decision Tree. (With the increase of max_depth, results tend to overfitting)





The pictures below show how max_depth (range from 0 to 25) affects prediction accuracy for Random Forest. (With the increase of max_depth, results tend to overfitting)





The reason why this happens is because when the tree goes deeper, it is prone to capture more noises and becomes more specific to the training set. Therefore, choosing a reasonable max_depth will give better performance and become more useful for unknown incoming data.

4.2 CART

We use the cross validation training method and let the model determine the best parameter selection. We provide the 36 combinations of parameters chosen for both non-specific and specific category classification. Here is the parameter we choose for the CART algorithm.

criterion	gini	log-loss	
splitter	best	random	
min_samples_leaf	100	200	300
max_depth	3	5	8

In addition to these parameter settings, we also analyze the model performance under different feature selections[2].

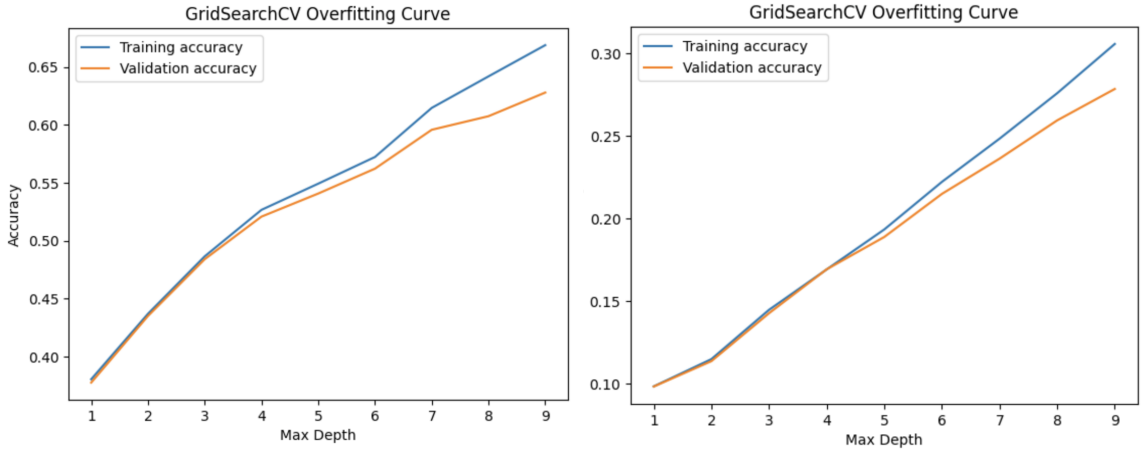
	Malware category	**Specific category**
CFS algorithm	0.44	0.18
correlation coef < -0.5	0.55	0.22
full data	0.6	0.25

We considered the same model but under different feature selection mechanisms, we found out that while increasing the feature selection number, the model performance increases. However, in the second and third row, the increase is not as obvious as previous. We suspect it might be overfitting, so we check the training accuracy and testing accuracy under full data feature selection.

	Malware category	**Specific category**
CART Training	0.66	0.26
CART Testing	0.6	0.25

To simplify checking the overfitting process, we maintain the consistency of the criterion, splitter, and min_samples_leaf as default, but change the max depth to a range of 1 to 10.

Left: *Overfitting curve of malware type over ransomware, spyware, trojan*
Right: *Overfitting curve of specific malware type over ako, etc.*



However, the training accuracy of CART in both categorizations has no noticeable difference than the testing accuracy.

4.3 XGBoost

We use the similar solution as above to let the model decide which combination of parameters is best. We use the cross validation training method and we provide the 18

[2] correlation coefficient < -0.5 has 16 features selected, CFS algorithm has 7 features selected

combinations of parameters chosen for both non-specific and specific category classification. Here is the parameter we choose for the XGBoost algorithm.

objective	multi:softmax	binary:logistic	
criterion	mse		
n_estimator	100	200	300
max_depth	3	5	8

And the model performance under different conditions.

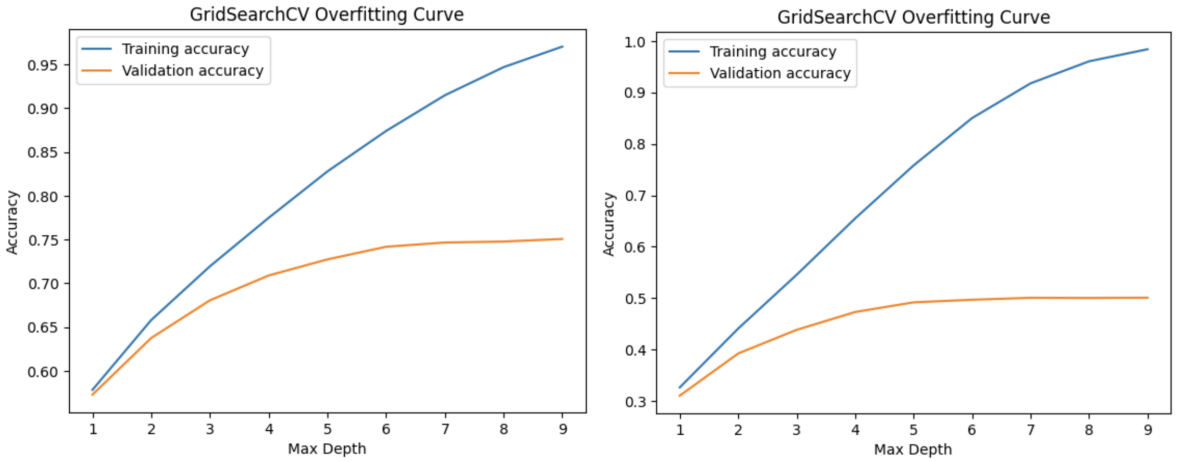
	Malware category	**Specific category**
CFS algorithm	0.51	0.25
correlation coef < -0.5	0.73	0.47
full data	0.76	0.53

Similarly, in the second and third row, the increase is not as obvious as previous. We suspect it might be overfitting, so we check the training accuracy and testing accuracy under full data feature selection.

	Malware category	**Specific category**
XGBoost Training	0.96	0.92
XGBoost Testing	0.76	0.53

We found that the XGBoost is overfitting on the full feature sets, as its training accuracy is 0.96 and 0.92 , which are way too higher than the real test accuracy. No matter what we adjust the parameters for this model under the full feature sets, the model is always overfitting. In this case, we wanted to choose the correlation coefficient method under the rationale choosing. To simplify checking the overfitting process, we maintain the consistency of the criterion, splitter, and min_samples_leaf as default (since changing them won't make too much difference on it), but change the max depth to a range of 1 to 10.

Left: *Overfitting curve of malware type over ransomware, spyware, trojan.*
Right: *Overfitting curve of specific malware type over ako, cws, etc.*



As we see, when we maintain the consistency of other parameters, the XGBoost model overfit under the full feature sets when the max depth is greater than 4, which the difference between the training accuracy and validation accuracy is larger to 10%. In this case, to maximize the model performance, the max depth of XGBoost should be set less than 4 and other parameters actually do not have too much effect on the model performance.

Reference

[1] Hall, M.A. (2000). Correlation-based feature selection of discrete and numeric class machine learning. (Working paper 00/08). Hamilton, New Zealand: University of Waikato, Department of Computer Science.

[2] Dener, M.; Ok, G.; Orman, A. Malware Detection Using Memory Analysis Data in Big Data Environment. *Appl. Sci.* 2022, *12*, 8604. https://doi.org/10.3390/app12178604

Appendix 1[3]

Feature Name	Description
pslist.nproc	Total number of processes
pslist.nppid	Total number of parent processes
pslist.avg_threads	Average number of threads for the processes
pslist.nprocs64bit	Total number of 64 bit processes
pslist.avg_handlers	Average number of handlers
dllist.ndlls	Total number of loaded libraries for every process
dllist.avg_dlls_per_proc	Average number of loaded libraries per process
handles.nhandles	Total number of opened handles
handles.avg_handles_per_proc	Average number of handles per process
handles.nport	Total number of port handles
handles.nfile	Total number of file handles
handles.nevent	Total number of event handles
handles.ndesktop	Total number of desktop handles
handles.nkey	Total number of key handles
handles.nthread	Total number of thread handles
handles.ndirectory	Total number of directory handles
handles.nsemaphore	Total number of semaphore handles
handles.ntimer	Total number of timer handles
handles.nsection	Total number of section handles
handles.nmutant	Total number of mutant handles
ldrmodules.not_in_load	Total number of modules missing from the load list
ldrmodules.not_in_init	Total number of modules missing from the init list
ldrmodules.not_in_mem	Total number of modules missing from the memory list
ldrmodules.not_in_load_avg	The average amount of modules missing from the load list
ldrmodules.not_in_init_avg	The average amount of modules missing from the init list
ldrmodules.not_in_mem_avg	The average amount of modules missing from the memory
malfind.ninjections	Total number of hidden code injections
malfind.commitCharge	Total number of Commit Charges
malfind.protection	Total number of protection
malfind.uniqueInjections	Total number of unique injections
psxview.not_in_pslist	Total number of processes not found in the pslist
psxview.not_in_eprocess_pool	Total number of processes not found in the psscan
psxview.not_in_ethread_pool	Total number of processes not found in the thrdproc
psxview.not_in_pspcid_list	Total number of processes not found in the pspcid
psxview.not_in_csrss_handles	Total number of processes not found in the csrss
psxview.not_in_session	Total number of processes not found in the session
psxview.not_in_deskthrd	Total number of processes not found in the desktrd
psxview.not_in_pslist_false_avg	Average false ratio of the process list
psxview.not_in_eprocess_pool_false_avg	Average false ratio of the process scan
psxview.not_in_ethread_pool_false_avg	Average false ratio of the third process
psxview.not_in_pspcid_list_false_avg	Average false ratio of the process id
psxview.not_in_csrss_handles_false_avg	Average false ratio of the csrss
psxview.not_in_session_false_avg	Average false ratio of the session
psxview.not_in_deskthrd_false_avg	Average false ratio of the deskthrd
modules.nmodules	Total number of modules

[3] The first appendix is feature description refers to [2] in reference

svcscan.nservices	Total number of services
svcscan.kernel_drivers	Total number of kernel drivers
svcscan.fs_drivers	Total number of file system drivers
svcscan.process_services	Total number of Windows 32 owned processes
svcscan.shared_process_services	Total number of Windows 32 shared processes
svcscan.interactive_process_services	Total number of interactive service processes
svcscan.nactive	Total number of actively running service processes
callbacks.ncallbacks	Total number of callbacks
callbacks.nanonymous	Total number of unknown processes
callbacks.ngeneric	Total number of generic processes

Appendix 2[4]

Correlation Coefficient < 0.5		
pslist.avg_threads	dlllist.avg_dlls_per_proc	dlllist.ndlls
handles.nevent	handles.nthread	svcscan.nactive
svcscan.interactive_process_services	handles.nsemaphore	handles.ntimer
handles.nsection	handles.nmutant	ldrmodules.not_in_load
ldrmodules.not_in_init	drmodules.not_in_mem	svcscan.nservices
svcscan.shared_process_services		

CFS (Ransomware, Spyware, Trojan)		
malfind.uniqueInjections	modules.nmodules	malfind.ninjections
psxview.not_in_csrss_handles_false_avg	pslist.avg_threads	malfind.protection
callbacks.nanonymous		

CFS (Benign, Malware)		
dlllist.avg_dlls_per_proc	handles.nmutant	pslist.avg_threads
ldrmodules.not_in_load	svcscan.process_services	handles.nsection
ldrmodules.not_in_mem	svcscan.shared_process_services	pslist.nppid
ldrmodules.not_in_mem_avg	handles.nevent	

[4] Feature selection