Predicting Malware Attacks

Akshara Nair (MT22008) **Bhanu Nautiyal** (MT22138) **Pratik Chauhan** (MT22118) **Shubham Agrawal** (MT22124)

Abstract

Malware is a file or code that interferes with the normal functioning of a computer with the malicious intention of corrupting a system using its mutation properties by which it not only creates copies of itself but also creates new variants, hence making it necessary to classify each sample based on its representation and behavior. Promoting this idea, Microsoft launched a challenge in 2015 in the Kaggle platform, encouraging it to build a model that classifies malware into different categories using half a terabyte of data containing more than 20,000 malware samples in byte and asm file format. In this project, the approach mainly emphasizes on tackling the huge dataset, extracting and creating novel features from both byte and asm files and finally fitting machine learning models to classify different types of malware. The proposed method achieved a log loss of 0.29

1 Introduction

With the growing necessity of the internet, there has been exponential growth in the numbers of Internet users worldwide and so did the number of Internet users with malicious intent. Modern Malwares is an umbrella term for viruses, bots, ransomware, or anything that if present in a system can launch attack to do wide range of malicious activities threatening the security of a system by sharing confidential information, bringing down servers, penetrate network.[1] One of the main reason for the extensive presence of malwares are because of the use of metamorphic and polymorphic techniques build by malware developers, which creates modified copies of itself. The difference between polymorphic malware and metamorphic malware is that in polymorphic malware the mutation is static that only aims to encrypt and decrypt the code, on the other hand metamorphic malwares modify the code every time.

Since such malware attacks have the potential to pose serious threats it is important to detect malwares early to build defense mechanisms to prevent such attacks. And studies have shown that the success of any anti-malware industry directly depends on the detection as the encrypted actions are different for different viruses.

There have been many attempts to apply machine learning in the malware detection domain starting in 2001 [2] whereby the researchers introduced the problem by saying, “Eight to ten malicious programs are created every day, and most cannot be accurately detected until signatures have been generated for them.” For classification of malwares, firstly it is important to analyze if a content exhibits any malware. After this step, malware needs to be categorized into groups. The malware could be analyzed by extracting the characteristics of the syntax and semantics of a program such as opcodes, signatures, byte code n-grams from hex code.

In this project, a system is proposed which primarily extracts malware characteristics to effectively assign malware samples to a family.This project was inspired by the work of Team say no to overfitting who won malware classification competition hosted by kaggle and also by the work of Rohan Paul.[6]

The overall task is divided into two sub-tasks:
Feature Engineering: - New features including file size, unigram, bigram , trigram counts of bytes and instructions in bytes and asm files are extracted, additionally asm file pixels are also added.
Modeling : - Final data was trained on machine learning algorithms including KNN, logistic regression, Random forest and Xgboost and evaluated. Along with feature engineering and modeling, cross validation played an important role to tackle overfitting.

2 Related Works

In Bugra Cakir et al.[3] feature extraction was done using shallow deep-network. An accuracy of 96% was achieved using gradient boosting algorithm and whose performance was evaluated using k-fold cross validation split.Peng et. al[7], used attention mechanism and proposed an approach using CNN and LSTM to detect phishy url. The focus was on extracting features from urls like texture information,lexical information and host information. Finally url classification was done using softmax classifier which resulted in accuracy of 98.26%.

Cui et al[4] used a model which identifies mobile malware behavior using data mining techniques.They used bunching technique called withdrawal group was made which utilizes earlier learning to reduce measure of the dataset. The model was then trained on Naive Bayes and Decision Tree.Matilda Rhode et al.[8] model generated an accuracy of 94% using RNN on the VirusTotal dataset which consisted of thousand malicious and six hundred trusted samples of windows 7.DI XUE et al[9] Proposed a system based on probability score where CNN with spatial Pyramid Pooling used mainly to analyze static features in phase 1 and dynamic features in phase 2. To get probability scoring phase 1 and phase 2 were concatenated. Surprisingly their result achieved 98.82% accuracy and 100% recall and precision for 52 malware families out of 63. Huan Zhou used deep learning network using combination of static and dynamic features for file representation[5].

3 Methodology

The main focus was on selecting features used to represent each sample. Therefore, the rationale guided the aim that the final set of rich content-based features integrates different types of components to attain the most accurate results.

Figure 1: Overview of the project.

1.) Data:
The data was provided by Kaggle and for each malware, two types of files are present: hexadecimal byte contents and asm files produced by a commercial disassembler IDA pro. Also, a CSV file mentioning the class of each malware with a 20-character hash ID is provided. The nine different types of malware are Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator.ACY, Gatak.
The original dataset published was of around 500GB uncompressed, and the data was skewed and therefore these two were the big constraints to work upon. This was tackled by extracting 42 random samples belonging to each class label as the least frequent class was analyzed to be class 5 containing only 42 samples in total. In essence, approximately 5GB of workable and balanced dataset was generated on which further analysis was done.
Hence, we did all our analysis on a 5.6 GB balanced dataset.

2.) Feature Engineering:

a. Features extracted from byte files : Hexadecimal digit can take 16 values each , making in total 256 possible values and hence 256 different unigram bag of words were created. For each byte file the frequency of the hexadecimal values were counted. These values were then normalized using Min-Max Scaler. Along with unigram of byte Files, top 2000 Bi-gram of Byte files were selected.The selection was done using SelectKBest which selects highest k features using a function and here the function was kept as Chi-Square Test and k equal to 2000. Byte file size was added as another feature. Additionally, we merged unigram byte counts and size of each byte file was merged on ID to create a new feature whose values were then normalized.

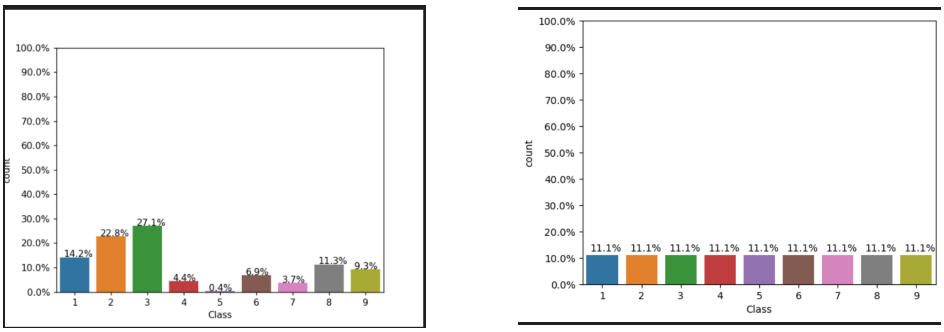
b. Features extracted from asm files: Asm files were a combination of address,segments,opcodes,registers,function class and APIs. Univariate analysis is shown in table . Based on further analysis, the following features were added. Size of ASM files, top 52 unigram of ASM files, top 500 bigram of opcodes, top 800 trigram of opcodes. Additionally, first 800 pixels data from ASM file images were added by first converting asm files to images and then converting it to numpy array and then extracting pixels from each image of each asm file.

Finally all the features of both the files were were merged into a single dataframe.

3.1 Modelling

Both byte feature dataframe and asm file dataframe were trained on four machine learning algorithms namely KNN,Logistic Regression, Random Forest Classifier and XgBoost Classifier and the final dataframe generated after merging all the feature of both byte and asm files was trained on Xgboost with the best hyperparameter got from RandomizedSearchCv to mitigate overfitting.

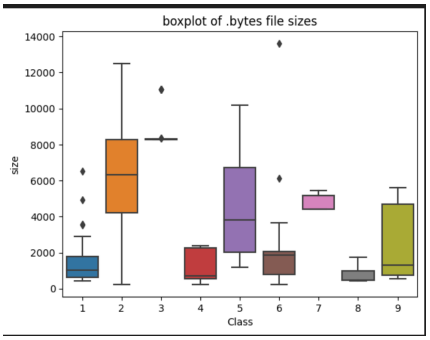
4 Data Analysis



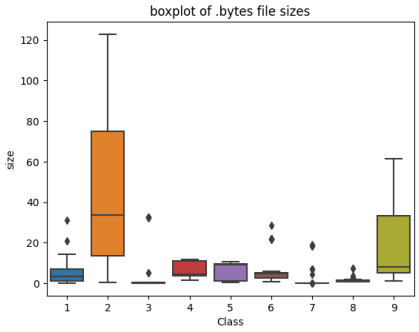
(a) Counts of malwares before sampling.

(b) Count after sampling.

Figure 2: Figures showing before and after sampling.



(a) Classwise file size distribution of byte files.



(b) Classwise file size distribution of asm files.

Figure 3: Classwise size distributions in both bytes and asm files.

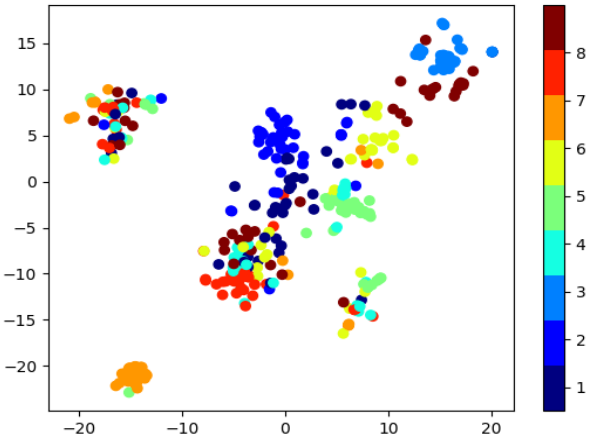
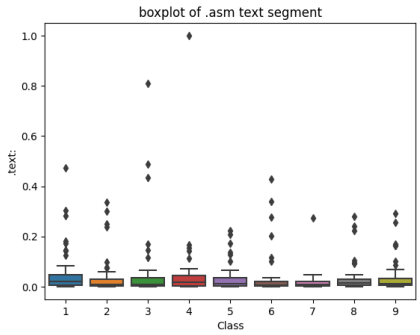
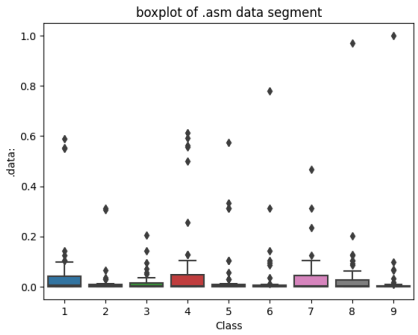


Figure 4: TSNE plot on combined features of both asm and byte files (file sizes + unigram counts)

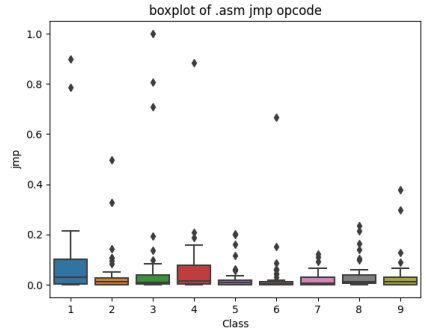
We have taken only 52 features from asm files (after reading through many blogs and research papers) The univariate analysis was done only on few features. Take-aways: Each feature has its unique importance in separating the Class labels.



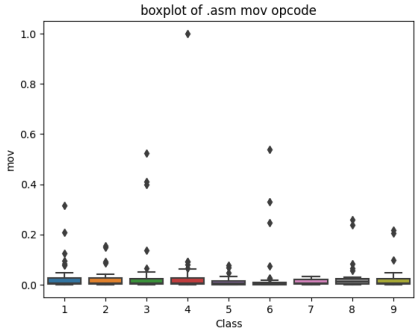
(a) Classes 1,2,3,4,5 have higher counts of the feature .text



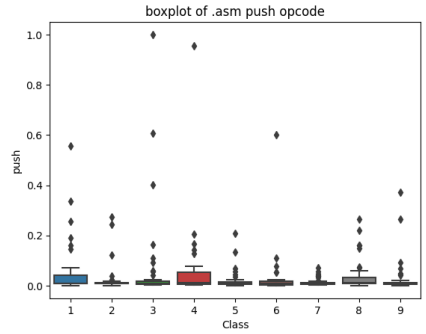
(b) Classes 1, 4. 7 and 8 have higher counts of the feature .data



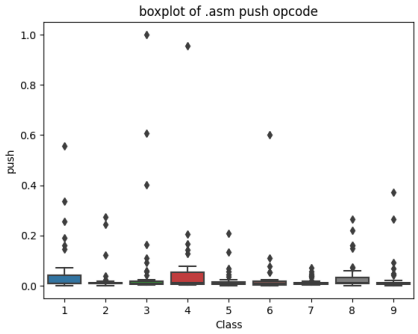
(a) Classes 1 and 4 have significantly larger counts of the feature jmp



(b) Classes 5 and 6 have lower counts of mov instruction



(a) Classes 1 and 4 have higher counts of push instruction



(b) Some op codes like .Pav: occur very rarely in any class

5 Evaluation

5.1 KNN algo on normalised features (byte + size)

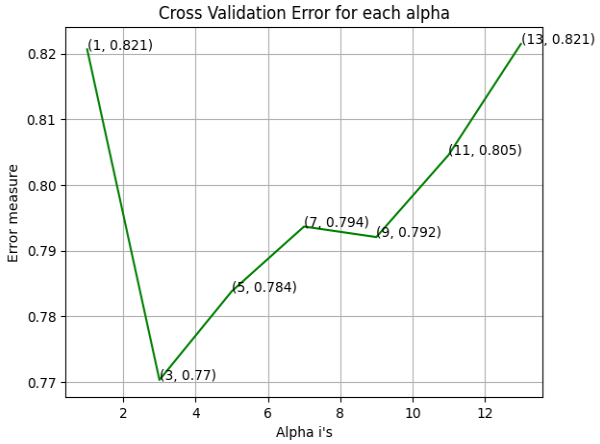


Figure 8: Number of Nearest neighbour vs Error

Alpha represents number of nearest neighbours

For values of best alpha = 3. The train log loss is: 0.4772889187221725

For values of best alpha = 3. The cross validation log loss is: 0.7703240260694246

For values of best alpha = 3. The test log loss is: 0.9494538400604402

Accuracy : 77.63157894736842 (on Test data)

5.2 Logistic Regression on normalised features (byte + size)

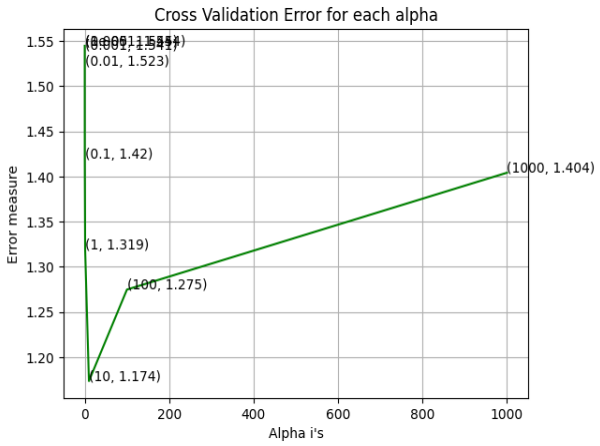


Figure 9: Inverse regularisation parameter vs Error

Alpha denotes a hyperparameter called Inverse regularisation parameter

log loss for train data 1.0830314385746629

log loss for cv data 1.1735267524155137

log loss for test data 1.2980097759009468

Accuracy : 68.42105263157895 (Test)

5.3 Random Forest on normalised features (byte + size)

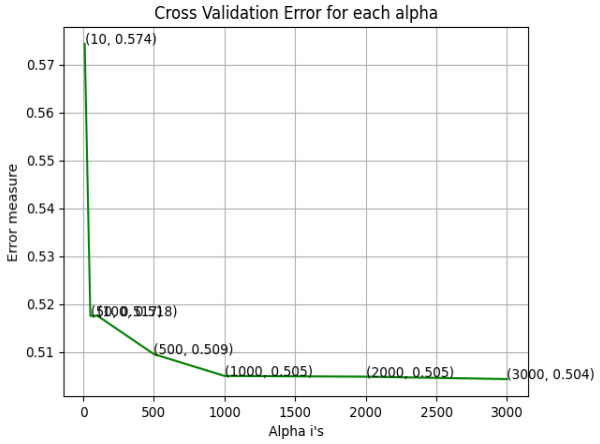


Figure 10: Number of estimators vs Error

Alpha denotes a hyperparameter called number of estimators

For values of best alpha = 3000. The train log loss is: 0.2550139559929393

For values of best alpha = 3000. The cross validation log loss is: 0.5042914547494448

For values of best alpha = 3000. The test log loss is: 0.6542908815029106

Accuracy : 84.21052631578947 (test)

5.4 XGBoost on normalised features (byte + size)

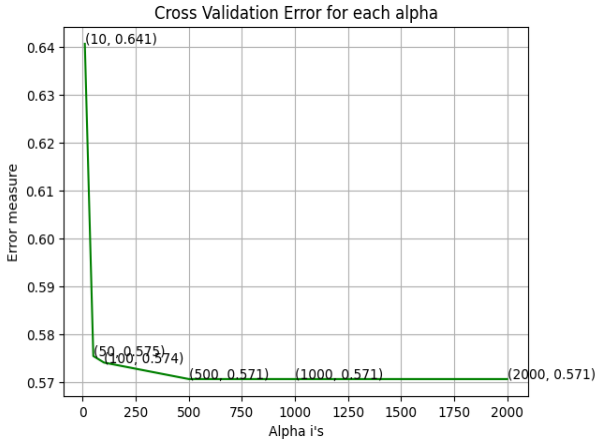


Figure 11: Number of estimators vs Error

alpha denotes a hyperparameter called number of estimators

For values of best alpha = 2000 The train log loss is: 0.2862226490547623

For values of best alpha = 2000 The cross validation log loss is: 0.5706082301800358

For values of best alpha = 2000 The test log loss is: 0.7322269342006231

Accuracy : 85.52631578947368

Log losses after hyperparameter search :- 'subsample': 1, 'n_estimators': 1000, 'max_depth': 10, 'learning_rate': 0.05, 'colsample_bytree': 0.1

train loss 0.24537333596291822

cv loss 0.4977482685080632

test loss 0.6648262099273531

5.5 KNN algo on normalised features (prefixes + keywords + instructions + registers + size)

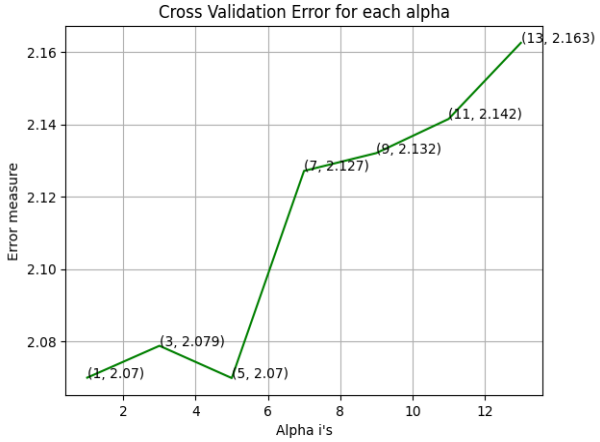


Figure 12: Nearest neighbours vs Error

alpha denotes number of nearest neighbours

For values of best alpha = 5 The train log loss is: 1.92482807563511

For values of best alpha = 5 The cross validation log loss is: 2.069873530421539

For values of best alpha = 5 The test log loss is: 2.135518986511267

Accuracy : 15.789473684210526

5.6 Logistic regression on normalised features (prefixes + keywords + instructions + registers + size)

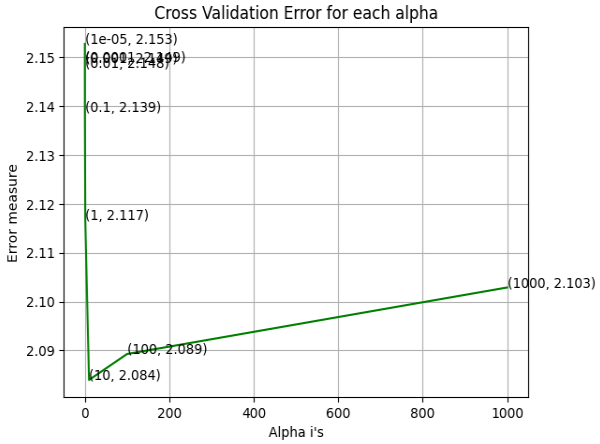


Figure 13: Inverse Regularisation Parameter vs Log-Loss

log loss for train data 2.0436865211887065

log loss for cv data 2.0839637984419923

log loss for test data 2.0723245708396854

Accuracy : 23.684210526315788

5.7 Random Forest on normalised features (prefixes + keywords + instructions + registers + size)

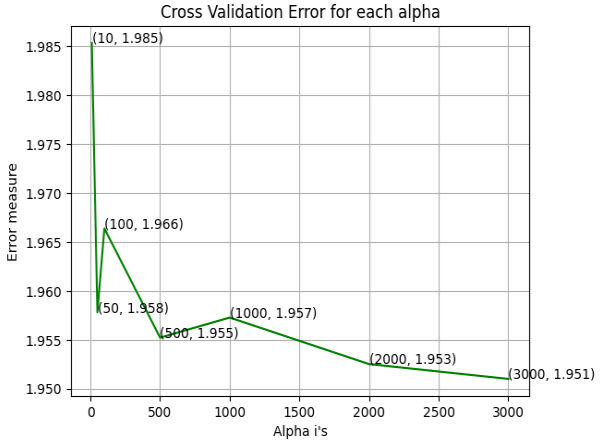


Figure 14: Number of estimators vs Error

For values of best alpha = 3000 The train log loss is: 1.1776127169292536

For values of best alpha = 3000 The cross validation log loss is: 1.9510056978218426

For values of best alpha = 3000 The test log loss is: 2.0171271337013064

Accuracy : 25.0

5.8 XGBoost on normalised features (prefixes + keywords + instructions + registers + size)

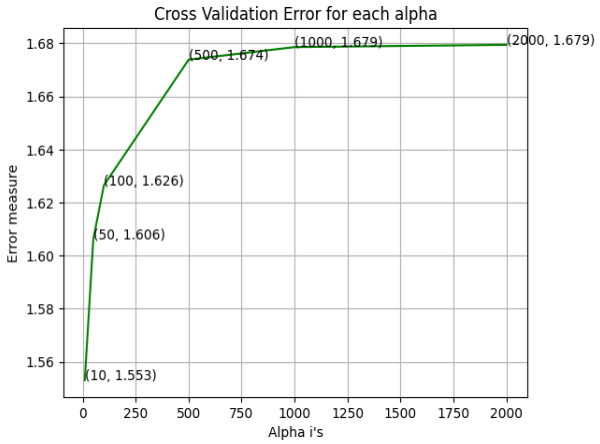


Figure 15: Number of estimators vs Error

For values of best alpha = 10 The train log loss is: 0.8119842553141021

For values of best alpha = 10 The cross validation log loss is: 1.5531367274658754

For values of best alpha = 10 The test log loss is: 1.5939354595548565

Accuracy : 51.31578947368421

After hyperparameter search: 'subsample': 0.5, 'n_estimators': 10, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1

train log loss is: 0.8119842553141021

cross validation log loss is: 1.5531367274658754

test log loss is: 1.5939354595548565

Accuracy : 51.31578947368421

Clusters can be seen clearly, but the inter cluster distances are very low.

6 Results

Algorithm Used	Log-Loss		
	Train	Validation	Test
KNN	0.977	0.77	0.94
Logistic	1.08	1.77	1.29
Random Forest	0.25	0.5	0.65
XGBoost	0.28	0.57	0.73
XGBoost with tuned params.	0.29	0.49	0.66

Table 1: Log-Loss from only byte files

Algorithm Used	Log-Loss		
	Train	Validation	Test
KNN	1.92	2.06	2.13
Logistic	2.04	2.08	2.07
Random Forest	1.17	1.95	2.01
XGBoost	0.811	1.55	1.59
XGBoost with tuned params.	0.811	1.55	1.59

Table 2: Log-Loss from only asm files

Algorithm Used	class	1	2	3	4	5	6	7	8	9
KNN	Precision	0.73	0.86	1	0.88	1	0.88	1	1	1
	Recall	1	0.75	1	0.88	1	0.86	1	1	0.78
Logistic Regression	Precision	1	1	1	0.62	1	1	0.9	0.75	1
	Recall	0.75	1	1	1	0.67	0.62	1	1	0.89
Random Forest	Precision	1	1	0.73	1	1	0.78	0.9	1	1
	Recall	1	1	1	0.88	0.89	0.88	1	1	0.67
XGBoost	Precision	0.82	0.9	1	1	1	0.75	1	0.67	1
	Recall	1	1	1	0.88	0.88	0.75	1	0.67	0.88

Table 3: Precision and Recall [only Byte Files]

Algorithm Used	class	1	2	3	4	5	6	7	8	9
KNN	Precision	0.06	0.5	0.15	0.19	0.5	0	0.67	0.9	0
	Recall	0.12	0.56	0.33	0.38	0.11	0	0.25	0.25	0
Logistic Regression	Precision	0.5	0.45	0.19	0.17	0.11	0.33	0.25	0.14	0
	Recall	0.11	0.62	0.12	0.12	0.11	0.44	0.5	0.11	0
Random Forest	Precision	0.08	0.33	0.36	0.09	0.22	0.33	0.5	0.6	0
	Recall	0.11	0.38	0.5	0.12	0.22	0.22	0.62	0.33	0
XGBoost	Precision	0.14	0.33	0.67	0.29	0.44	0.60	0.33	0.44	1
	Recall	0.22	0.62	0.75	0.25	0.44	0.33	0.25	0.44	0.25

Table 3: Precision and Recall [only Asm Files]

7 Conclusion

The proposed method develops a hybrid set of features by extracting features from both byte and ASM files. The performance of the individual datasets was evaluated on four classifiers. It was observed that XGboost outperformed KNN, logistic regression, and Random Forest models when data was trained on byte file feature and ASM feature individually. Then additional features were extracted from both files and combined to form the final dataset and modeled on Xgboost with the following hyper-parameters-subsample: 0.3, n_estimators:100, max_depth:5, learning_rate:0.1. XGboost with the above hyperparameters resulted in an accuracy of 96.052% with train set log-loss of 0.205, validation set log-loss of 0.2847, and test set log-loss of 0.296.

References

[1] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy*, pages 183–194, 2016.

[2] Ahmed Amer and Normaziah A Aziz. Malware detection through machine learning techniques. *International Journal of Advanced Trends in Computer Science and Engineering*, 8(5):2408–2413, 2019.

[3] Bugra Cakir and Erdogan Dogdu. Malware classification using deep learning methods. In *Proceedings of the ACMSE 2018 Conference*, pages 1–5, 2018.

[4] Baojiang Cui, Haifeng Jin, Giuliana Carullo, and Zheli Liu. Service-oriented mobile malware detection system based on mining strategies. *Pervasive and Mobile Computing*, 24:101–116, 2015.

[5] James Graham, Ryan Olson, and Rick Howard. *Cyber security essentials*. CRC Press, 2016.

[6] Rohan Paul. https://rohan-paul-ai.netlify.app/blog/post/microsoft-malware-detection-kaggle-challenge. 2015.

[7] Yongfang Peng, Shengwei Tian, Long Yu, Yalong Lv, and Ruijin Wang. A joint approach to detect malicious url based on attention mechanism. *International Journal of Computational Intelligence and Applications*, 18(03):1950021, 2019.

[8] Matilda Rhode, Pete Burnap, and Kevin Jones. Early-stage malware prediction using recurrent neural networks. *computers & security*, 77:578–594, 2018.

[9] Di Xue, Jingmei Li, Tu Lv, Weifei Wu, and Jiaxiang Wang. Malware classification using probability scoring and machine learning. *IEEE Access*, 7:91641–91656, 2019.