Malware Classification using Machine Learning

Sarath Kumar Mannam
College of Engineering and Computer Science
University of Central Florida
Orlando, USA
sarath.mannam13@

Joseph Filipek
College of Engineering and Computer Science
University of Central Florida
Orlando, USA
joefilipek@knights.ucf.edu

Abhishek Maruturi
College of Engineering and Computer Science
University of Central Florida
Orlando, USA
abhishekmaruturi@knights.ucf.edu

Dariel Tenf
College of Engineering and Computer Science
University of Central Florida
Orlando, USA
dtenf816@knights.ucf.edu

***Abstract*-Malware is a software that aims to bypass a computer's security for a variety of reasons, such as accessing sensitive data or ruining a computer's functionality. It is a portmanteau from the phrase "malicious software". The malware industry has seen massive growth since computers became almost entirely intertwined with the first world economy. With this rapid growth, anti-malware communities have had to continuously improve anti-malware to be more robust in detecting and terminating malware in the event of a cyber attack. In this work, we have used Machine Learning Algorithms to efficiently classify the types of malware into its respective categories. Counter-attacking the malware is dependent upon the class of malware it belongs to. Therefore, our algorithms can be used to efficiently categorize the malware.**

I. Introduction

Malware is a type of software that seeks to get beyond a computer's security for a number of objectives, including gaining access to sensitive data or destroying the computer's functionality. It's a combination of the words "malicious software" and "portmanteau". Different types of malwares can be introduced into the systems when we are on the Internet. The moment a user accesses the internet, the risk of getting malware is there. It is important to keep computers safe to avoid data loss.

A quick glance back at the history of malicious software reminds us that malware has been there since the inception of computing. During the 1970s, the first virus was discovered. It was known as the Creeper Worm, and it was an experimental self-replicating program that flashed the message "I'm the creeper, catch me if you can" on remote systems. After that many malicious software have been discovered and this fight turned out to be a never

ending and cyclical arms race. As security analysts and researchers improve their defences, malware developers continue to innovate, find new infection vectors, and enhance their obfuscation techniques.

One of the most essential elements being Endpoint protection. It provides a suite of security programs including, but not limited to, firewall, URL filtering, email protection, anti-spam and sandboxing. Traditional malware detection systems is an algorithm or hash that uniquely identifies a specific malware and heuristic based detection is a set of rules determined by experts after analysing the behaviour of malware.

A. Importance of Malware Detection Systems

As the number of devices and people accessing the Internet grows on a daily basis, it is very vital to keep our devices and information secure. According to a security firm, Norton, cybercrime victims lost a total of $19.4 billion USD in 2017 alone. During the COVID-19 pandemic, the first-world economy has become even more dependent on computers, as more and more people have begun to work from home. Accordingly, cybercrime has gone up to 600%.

Malicious files belonging to the same malware "family" (share the same malicious behaviour) are constantly being modified to look like different files to bypass security, so it is increasingly important that anti-malware products can keep up in real-time.

Our main aim in this project is to make use of machine learning algorithms to classify malwares by extracting the important features from the malware files. We'll be dealing with the .asm and .byte files for feature extraction.

B. Existing Literature and Related Work

By presenting certain feature types and feature selection strategies utilized in the literature,Shabtai et al. (2009) establish a taxonomy for malware detection using machine learning algorithms. They primarily concentrate on feature selection approaches (Gain ratio, Fisher score, document frequency, and hierarchical feature selection) as well as classification algorithms (Artificial Neural Networks, Bayesian Networks, Naive Bayes, K-Nearest

Neighbour, and so on). They also go over how ensemble methods may be used to integrate several different classifiers.

Bazrafshan et al. (2013) identify three main methods for detecting malicious software: (1) signature-based methods, (2) heuristic-based methods and behaviour-based methods. In addition, they investigate some features for malware detection and discuss concealment techniques used by malware to evade detection.

Souri et al. (2018) present a survey of malware detection approaches divided into two categories: (1) signature-based methods and (2) behaviour-based methods. However, the survey does not provide either a review of the most recent deep learning approaches or a taxonomy of the types of features used in data mining techniques for malware detection and classification.

In this project, we are focusing on extracting features using bytes and opcode N-grams. This is the most common type of feature for malware detection and classification. An n-gram is a contiguous sequence of n items from a given sequence of text. N-grams can be extracted from the bytes sequences representing the malware's binary content and from the assembly language source code. By treating a file as a sequence of bytes, byte n-grams are extracted by looking at the unique combination of every n consecutive bytes as an individual feature.

On the other hand, the sequence of assembly language instructions can also be extracted from the assembly language source code. In this case, only the mnemonic of the instruction, i.e. "ADD", "MUL", "PUSH", etc., are retained. Thus, opcode or mnemonic n-grams refer to the unique combination of every n consecutive opcodes as an individual feature.

Moskovitch et al. (2008) presented a method for classifying malware based on text categorization techniques. First, they extracted all n-grams from the training data, with n ranging from 3 to 6. Second, they selected the top 5500 features according to their Document Frequency (DF) score, to which the Fisher Score feature selection technique was later applied. Afterwards, using the resulting features as input they trained various algorithms such as an Artificial Neural Network (ANN), a Support Vector Machine (SVM), Naïve Bayes (NB) and Decision Trees (DT).

Jain and Meena (2011) proposed a method to extract bytes n-gram features, with n ranging from 1 to 8, from known malicious samples to assist in classification of unknown executables. As the number of unique n-grams is extremely large, they used a technique called class wise document frequency to reduce the feature space. Finally, different N-gram models were prepared using various classifiers like Naïve Bayes, Instance-based Learner, Decision Trees, Adaboost and Random Forests.

Fuyong et al. (2017) proposed a method that calculates the information gain of each byte n-gram in the training samples and selected K n-grams with the maximum information gain as features. Afterwards, they calculated the averages of each attribute of the feature vectors from the malware and benign samples separately. Lastly, a new piece of software was assigned to one of the two categories according to the similarity between the feature vector of the unknown sample and the average vectors of the two categories.

Santos et al. (2013) proposed a technique for malware detection based on the frequency of appearance of opcode sequences and its relevance. Each program was represented as a vector of features where each feature corresponds to a distinct 1-g or 2-g. To reduce the number of 2-g features, they applied Information Gain to select the top 1000 features. Their approach was validated on 17000 malicious and 1000 benign programs, and results show that the higher accuracy was achieved by a Support Vector Machine classifier with Pearson VII as kernel.

C. System Overview

We start with accessing the dataset which we have downloaded from Kaggle. The dataset consists of 150GB .asm files and 50GB .bytes files. We produce that data into the ML pipeline and extract features like bytes and opcodes n-grams. We have used several machine learning algorithms such as Random Forest, K-Means etc.

D. Data Collection

The Malware Detection Dataset consists of .byte files and .asm files. There are 10,868 files of .asm and .byte files each. We have used simple data preparation techniques with pandas and numpy to access the data from the downloaded dataset. The data contains a total of 9 classes such as Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator.ACY, Gatak.

E. Components of ML System

This project involves data preparation, data cleaning, Exploratory Data Analysis, Data Transformation, Data Modeling, Model Evaluation & Hyper Parameter Tuning. Each of these components will be explained briefly in sections 3 & 4.

II. Important definitions and problem statement

A. definitions

In the dataset, each file is classified into 9 families. They contain their ID, classification, and raw data that contains the hexadecimal representation of the file's binary content without the PE header and the .asm file.

Classes of Malware				
Ramnit	Lollipop	Kelihos_ver3	Vundo	Simda
Tracur	Kelihos_ver1	Gatak	Obfuscator.ACY	

Malware - Malware is software that aims to bypass a computer's security for a variety of reasons, such as accessing sensitive data or ruining a computer's functionality.

Byte files - Byte files contain binary code in hexadecimal format. The starting column represents the address, and the rest are machine instructions.

Asm files - Asm files are assembly source code file that contains the information of headers in the file like .text, .data, etc ; It shows information of opcodes (pop, mov), registers (edx, exi) and Keywords (std:) etc in the file.

Bag of Words - One feature to be extracted is gathering a bag of words from the byte files using SVD. Bag of Words is the frequency of each word(hex values) in a file. The number of features with uni-gram would yield 256 features.

B. PROBLEM STATEMENT

The task is to develop the best mechanism for classifying files in the test set into their respective family affiliations. The constraints are the huge data size and their lack of raw features. Since the dataset is huge (150GB), we have to optimize the training on a single box.

Some of the featurizations require the entire dataset to be on the RAM. We have to do software engineering to make the training possible. 1-gram would get 256 new features, while 2-gram would get 256^2 features. This ended up not allowing us to do n-gram features on .byte and .asm files.

III. OVERVIEW OF PROPOSED APPROACH/SYSTEM

A. COMPONENTS OF ML SYSTEM

Our ML system consists of Data Preparation and Cleaning, Exploratory Data Analysis and Data transformation, Data modeling, and Model Evaluation and Hyper Parameter Tuning.

B. DATA PREPARATION AND CLEANING

We gain raw data. Separate to byte files and asm files which ended up with 50GB of byte files and 150GB of asm files. In each byte file, the first column that represents the address of the instruction is ignored.

C. EXPLORATORY DATA ANALYSIS & DATA TRANSFORMATION

Distribution of the classes as well as the variance of the feature space will be taken into consideration to accurately distinguish classes. For byte files, file size will be taken as features, and uni-gram BOW of hexadecimal values will be taken as features. This results in 257 features. For asm files, file size will be taken as features as well, along with BOW of selected 51 keywords. This results in 52 features. Multivariate analysis on the extracted features will be performed to understand the correlation between the features.

D. Data Modeling

Machine learning models will be fitted to the transformed data and optimizing the classification parameters. Random model was run first to figure out the upper bound for byte files and asm files. K-Nearest Neighbor, Logistic Regression, Random Forest, and XgBoost were the selected models.

E. Model Evaluation & Hyper Parameter Tuning

We'll be using Model Accuracy and Multi-Class Log Loss. Logistic regression loss will evaluate the probability outputs of a classifier instead of its discrete predictions.

$$L_{\log}(Y, P) = -\log \Pr(Y|P) = -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{k=0}^{K-1} y_{i,k} \log p_{i,k}$$

The lower bound for the log loss is 0, which is what we want. The upper bound will be calculated by running a random model that randomly predicts malware classification in accordance to their population density(?). Model Accuracy is the percentage of malware correctly classified.

Confusion matrix, Precision matrix, and Recall matrix will be used to analyze the model effectiveness. In a Confusion matrix, each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. Precision matrix takes the confusion matrix and divides each value by the total sum of its column. This normalizes the matrix where we can compare the value as percentages of the instances in the predicted class. In a similar manner, the Recall matrix takes the confusion matrix and divides each value by the total sum of its row. The values now represent the percentages of the instances in the actual class. In both of them we want to see the diagonal values to be 1.0 where all the instances in the predicted class are in the original class. The derived Precision and Recall matrix will allow us to understand the confusion matrix easier.

IV. Technical Details of Proposed Approaches/Systems

While researching related work and other attempts at this project on kaggle, we found that modelling this data is surprisingly easy despite our team having no experience in cybersecurity. With each of the file types, we were able to clean the raw data rather easily and create a dataset that worked well with our proposed models.

Although simple, file size proved to be a valuable feature for use in our machine learning models. We generated boxplots for both file types as shown in figure 1 and 2 below. Although there is some overlap, it is clear that there is enough variance to provide a valuable feature for classification.

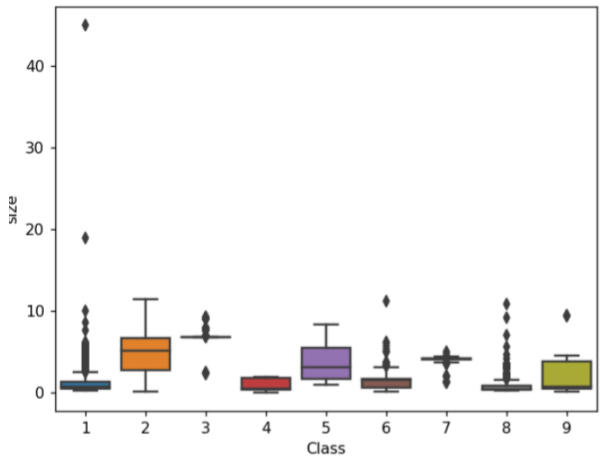

Figure 1. Class Distribution for File Size of Byte Files

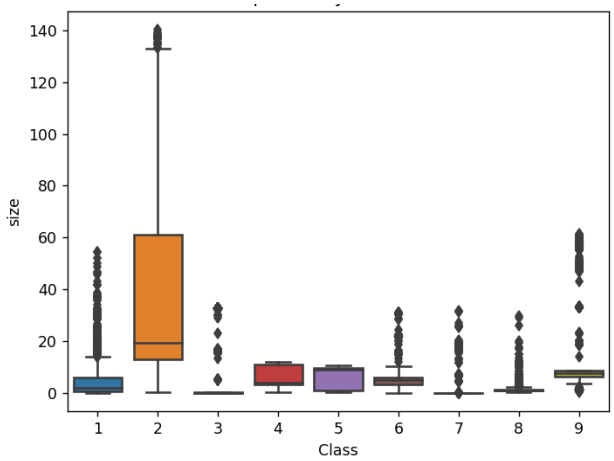

Figure 2. Class Distribution for File Size of .asm Files

To generate the rest of the features, there was a concern that it would be difficult for our team to determine what could be considered a valuable feature for our classification problem without a background in cybersecurity. What we found instead is that we can use the bag of words technique to accomplish this with no experience in cybersecurity and it proves to be extremely effective in separating the classes. In our ML system implementation, we decided to use unigram bag of words for the sake of simplicity and proof of concept, however, for a more robust model, it is also possible to use bigram and k-gram bag of words on the byte files to further increase accuracy. In the byte files, each word is represented by a hex value representation of a byte (e.g. 00, 01, …, ff), so with unigram bag

of words, we generated 256 new features, which including file size, gives each of our byte file samples a dimensionality of 257 features. In the asm files, the words are a bit less straightforward to extract as it requires more familiarity with the structure of the asm files themselves. Ultimately, we were able to extract 51 new features using the bag of words technique with the prefixes, opcodes, keywords, and registers as the words, which including file size, gives each of our asm file samples a dimensionality of 52 features. In figures 3, 4, 5, and 6, we performed multivariate analysis on both the cleaned byte file data and the cleaned asm data with t-SNE visualization with perplexities 50 and 30 on each.

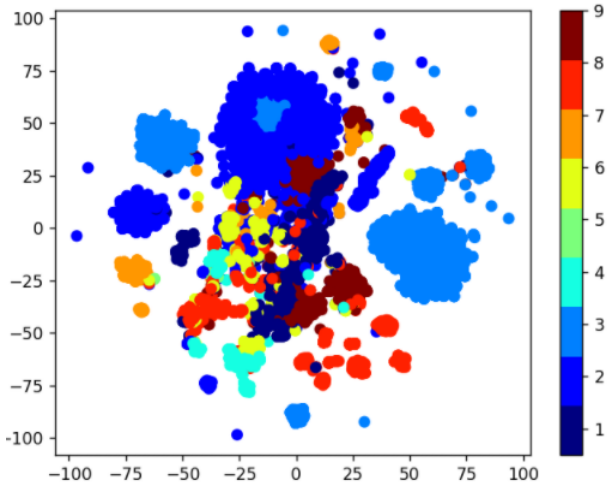

Figure 3. t-SNE Visualization for Byte Files (perplexity = 50)

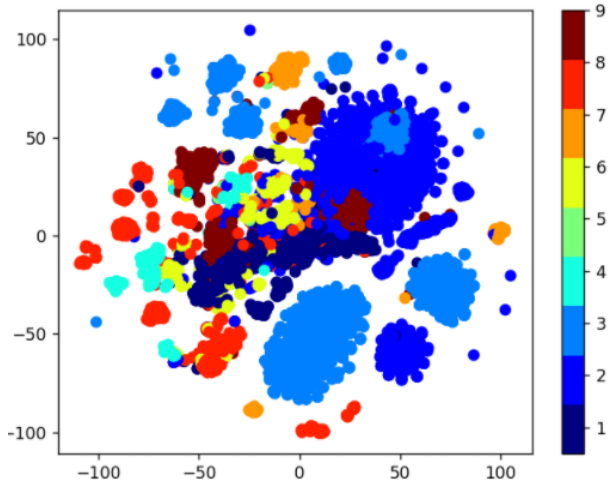

Figure 4. t-SNE Visualization for Byte Files (perplexity = 30)

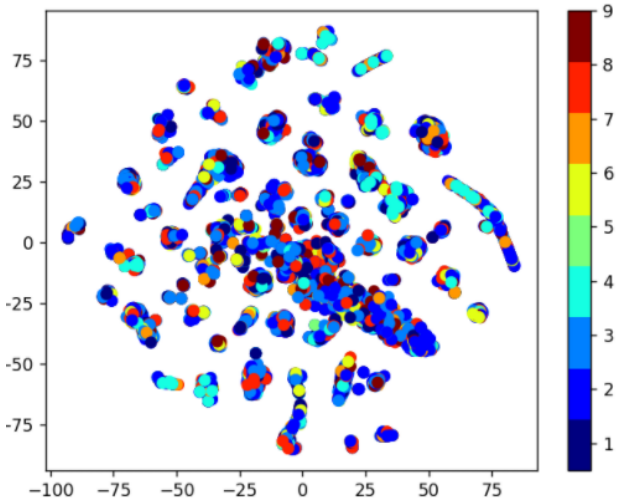

Figure 5. t-SNE Visualization for ASM Files (perplexity = 50)

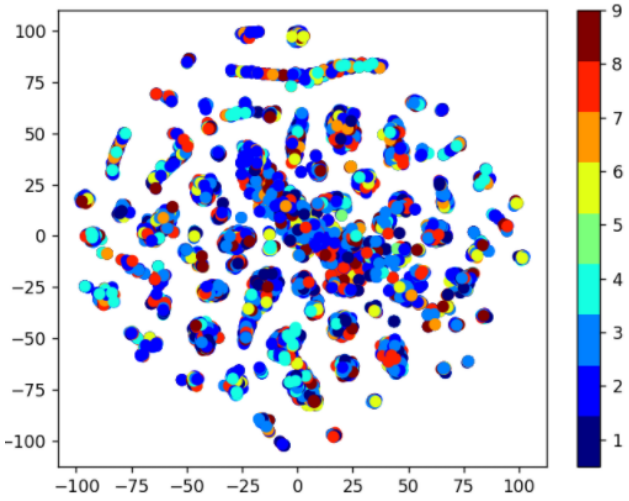

Figure 6. t-SNE Visualization for ASM Files (perplexity = 30)

After having cleaned the raw data, our data set was ready to be evaluated in a model, however, the information yielded from our models would be useless without first defining our evaluation metrics. For our evaluation metrics, we decided to use multi-class log loss as well as model accuracy accompanied by confusion, precision, and recall matrices, in order to obtain a clear observation on the weaknesses of each model. This allowed us to optimize our model selection from the set of models we applied to our dataset.

The first model we decided to explore was the random model, so we could define a worst case for any model we explore afterwards. After the random model, we explored four different multi-class classification models. The four models being K-Nearest Neighbor, Logistic Regression, Random Forest, and XGBoost.

V. Experiments

When conducting our experiments, we first set our attention on the byte files dataset as it was much smaller. So we began with applying the random model on the byte files dataset, which yielded a cross validation log loss of 2.46, a test log loss of 2.49, and a model accuracy of 11.5%. We also generated the model's confusion, precision, and recall matrices, which are shown below in figures 7, 8, and 9. With these results, we were able to define our worst case, and we knew that if any model were to yield worse results, something erroneous was occurring in our model.

Figure 7. Random Model Confusion Matrix for Byte Files

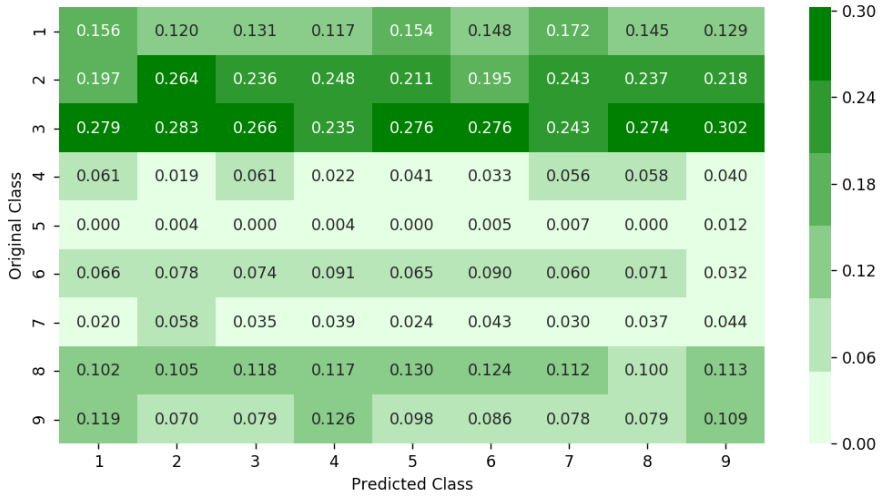

Figure 8. Random Model Precision Matrix for Byte Files

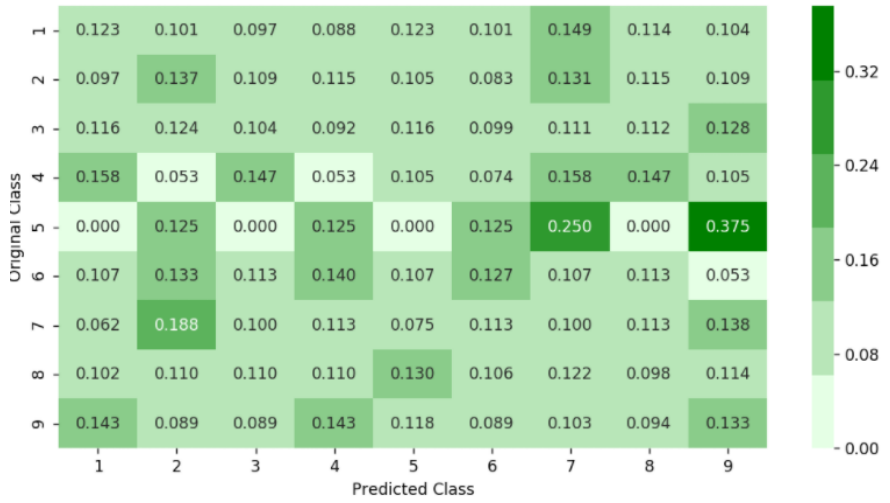

Figure 9. Random Model Recall Matrix for Byte Files

We then transitioned to the K-Nearest Neighbor (KNN) model, where before we calculated model accuracy and generated the confusion, precision, and recall matrices, we first calculated the cross validation log loss for several different values of k, so we could observe which k would be optimal for our model. This can be seen below in figure 10.

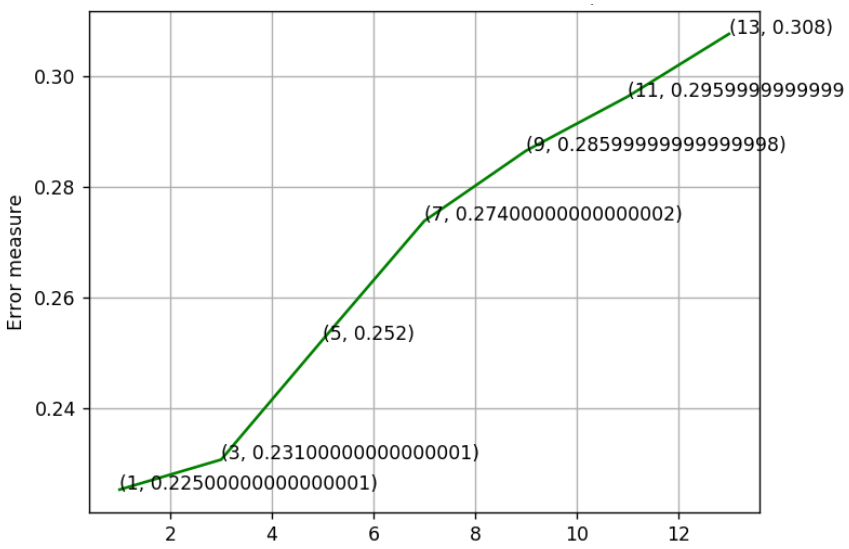

Figure 10. K vs Cross Validation Error for Byte Files

As shown in figure 10, the optimal k for this model happened to be k = 1. When we used this k, we yielded a cross validation log loss of .225, a test log loss of .242, and a model accuracy of 95.49%. The confusion, precision, and recall matrices are shown below.

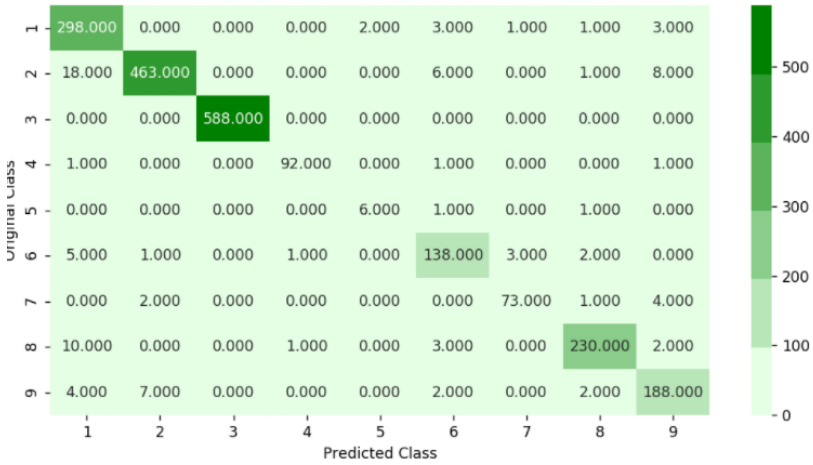

Figure 11. KNN Model Confusion Matrix for Byte Files

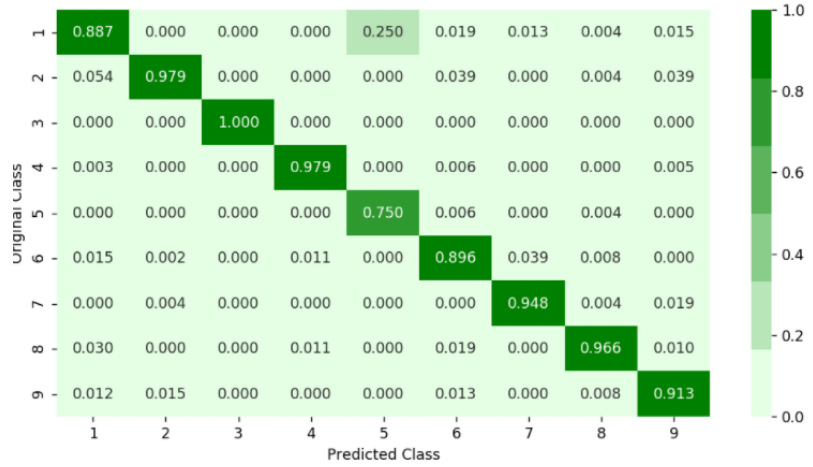

Figure 12. KNN Model Precision Matrix for Byte Files

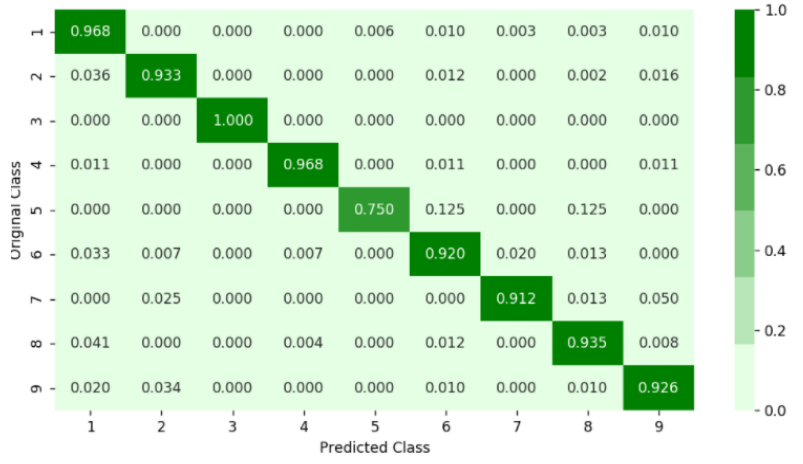

Figure 13. KNN Model Recall Matrix for Byte Files

After KNN, we explored Logistic Regression, however, just as what was done with KNN, we first optimized the hyperparameter c, before generating our data for the logistic regression model. The hyperparameter c represents the inverse of the regularization strength of the model. The larger the c, the smaller the regularization. The optimal c can be seen in the figure below.

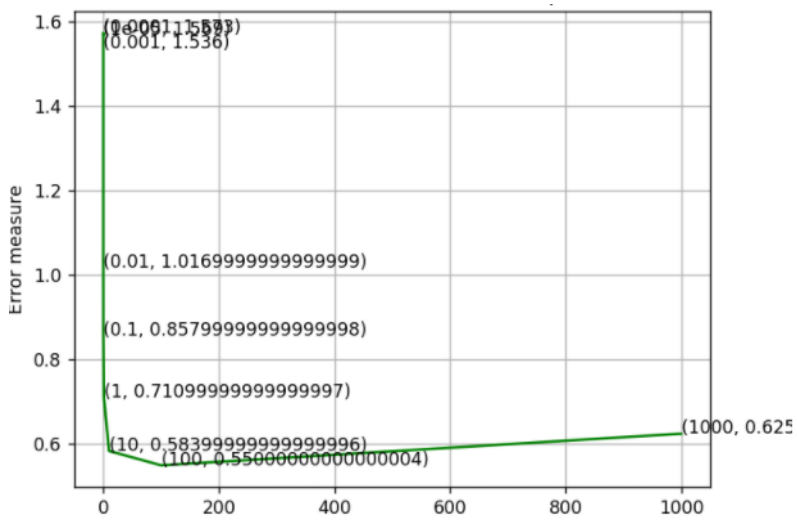

Figure 14. c vs Cross Validation Error for Byte Files

The optimal c for this model was c = 100, and with this value, we yielded a cross validation log loss of .55, a test log loss of .528, and a model accuracy of 87.67%. The confusion, precision, and recall matrices are shown below.

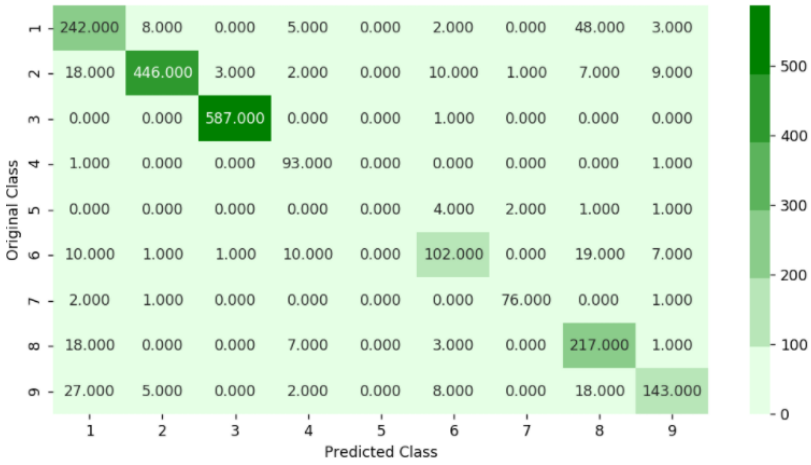

Figure 15. Logistic Regression Model Confusion Matrix for Byte Files

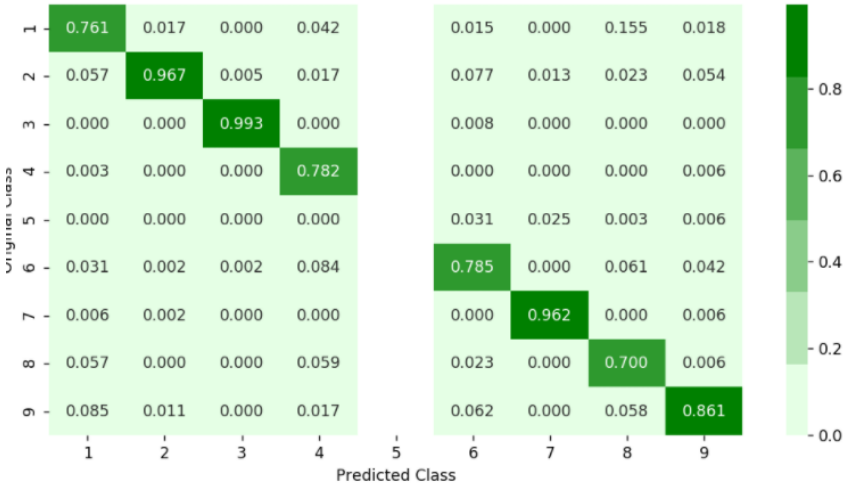

Figure 16. Logistic Regression Model Precision Matrix for Byte Files

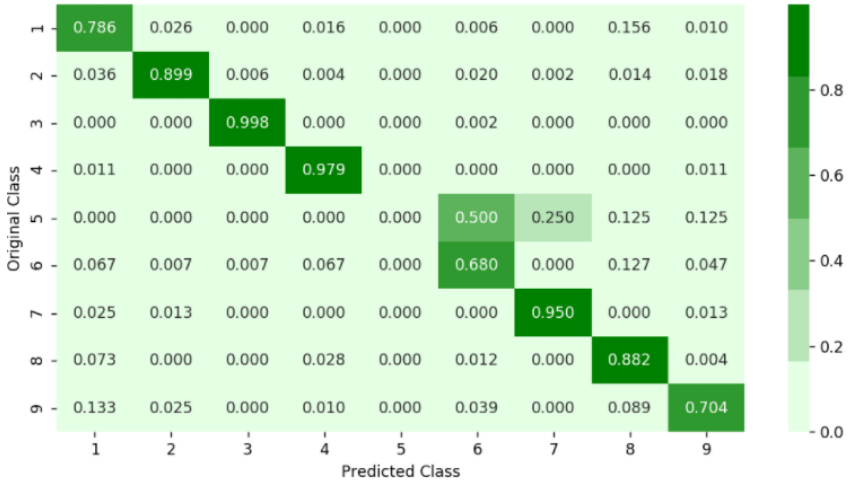

Figure 17. Logistic Regression Model Recall Matrix for Byte Files

The next model we explored was Random Forest with the number of estimators being the hyperparameter to be optimized. The optimal number of estimators can be found in the figure below.

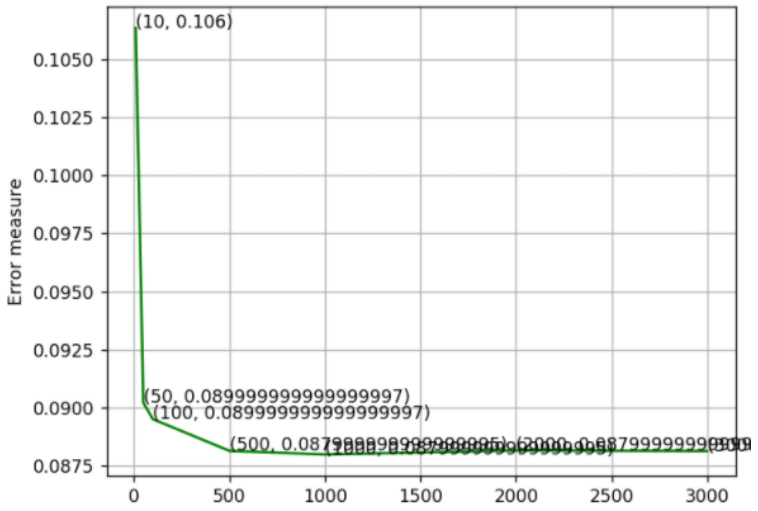

Figure 18. N_estimators vs Cross Validation Error for Byte Files
(Random Forest)

The optimal number of estimators for this model was n_estimators = 1000. With this number of estimators, we yielded a cross validation log loss of .088, a test log loss of .086, and a classification accuracy of 97.98%. The confusion, precision, and recall matrices are shown below.

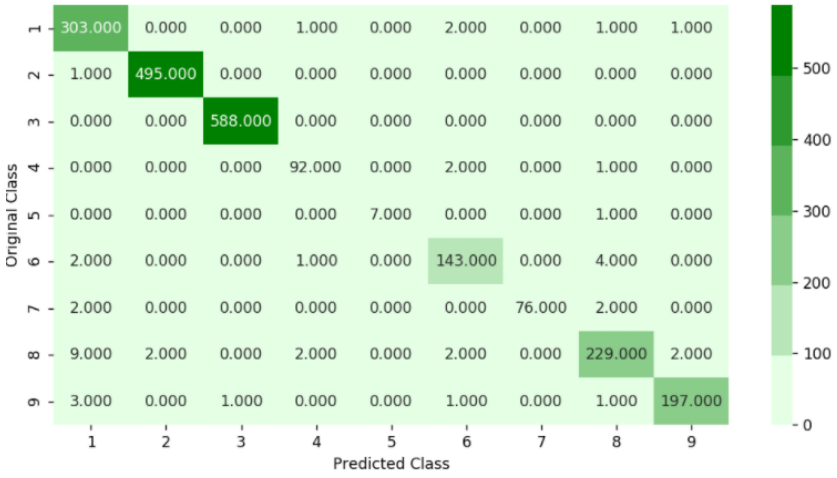

Figure 19. Random Forest Model Confusion Matrix for Byte Files

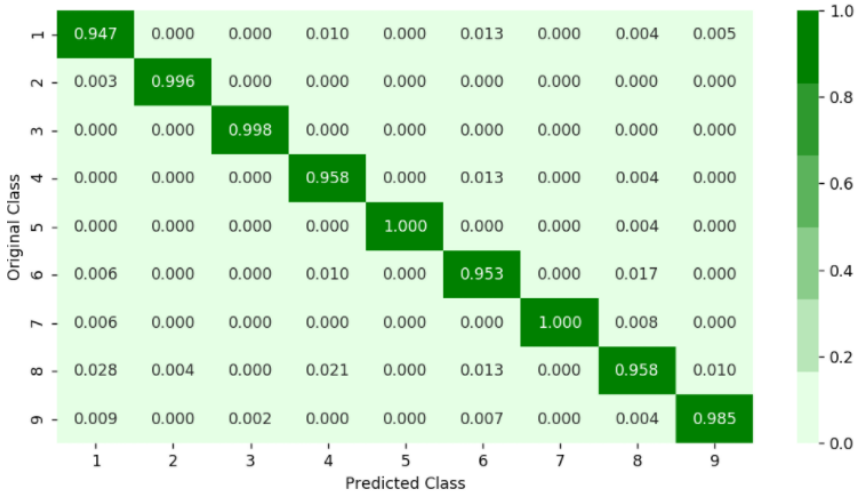

Figure 20. Random Forest Model Precision Matrix for Byte Files

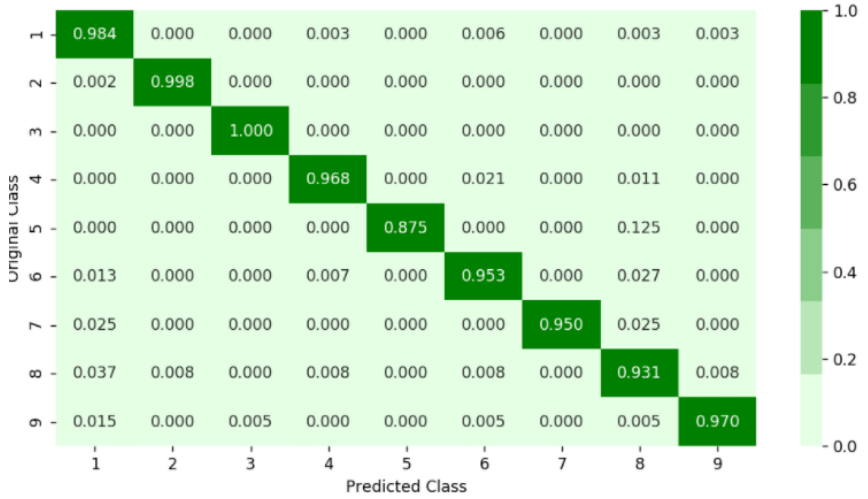

Figure 21. Random Forest Model Recall Matrix for Byte Files

The last model we applied on the byte file dataset was XGBoost with hyperparameter tuning on the number of estimators. The optimal number of estimators can be found in the figure below.

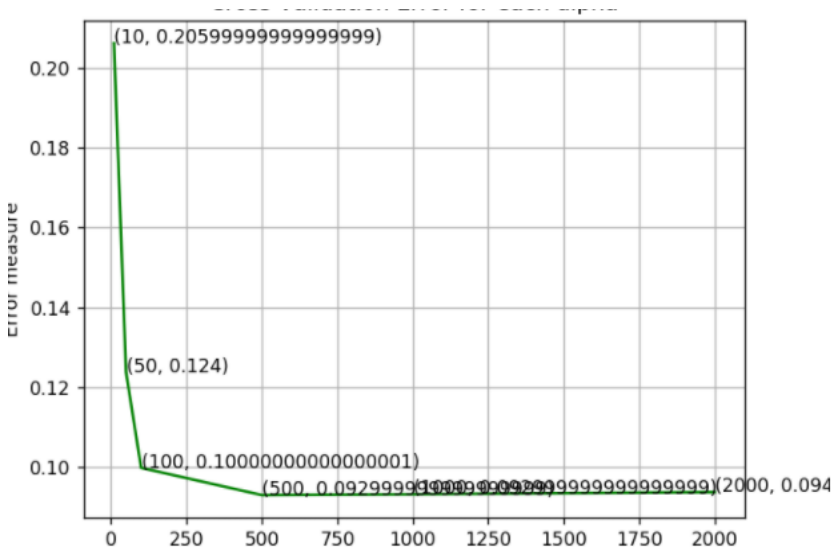

Figure 22. N_estimators vs Cross Validation Error for Byte Files (XGBoost)

The optimal number of estimators for this model was n_estimators = 500. With this number of estimators, we yielded a cross validation log loss of .093, a test log loss of .079, and a model accuracy of 98.76%. The confusion, precision, and recall matrices are shown below.

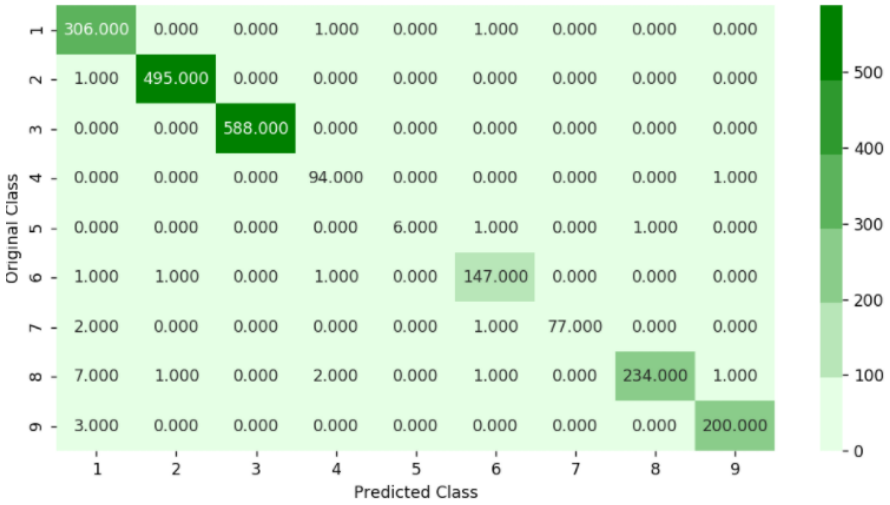

Figure 23. XGBoost Model Confusion Matrix for Byte Files

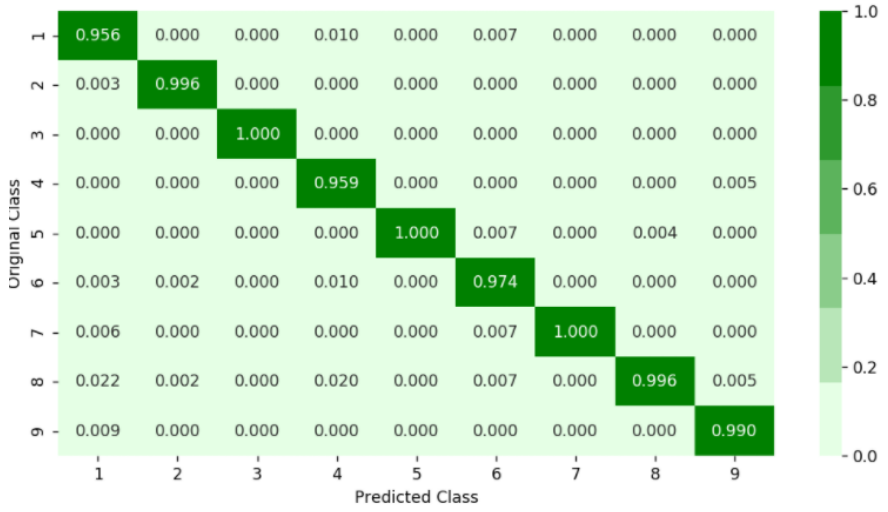

Figure 24. XGBoost Model Precision Matrix for Byte Files

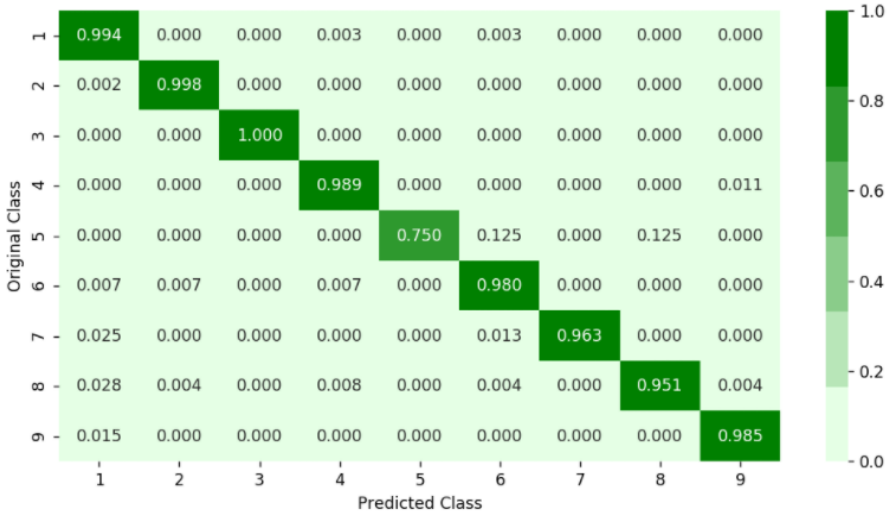

Figure 25. XGBoost Model Recall Matrix for Byte Files

After examining the data extrapolated from each model, we reached the conclusion that XGBoost provided the best model as it provided the highest accuracy, which for real world application, is the most important when trying to detect malware. Because of this, we optimized the model further using RandomSearch, which only provides a slight improvement, but an improvement nonetheless. With the fully optimized XGBoost, cross validation log loss was .092, and test log loss was .078.

After completing our analysis on the byte files dataset, we then transitioned to the asm file dataset, where we first applied the K-Nearest Neighbor (KNN) model. We began again with optimizing the number of k used in our model before generating the rest of the data . This can be seen below in figure 26.

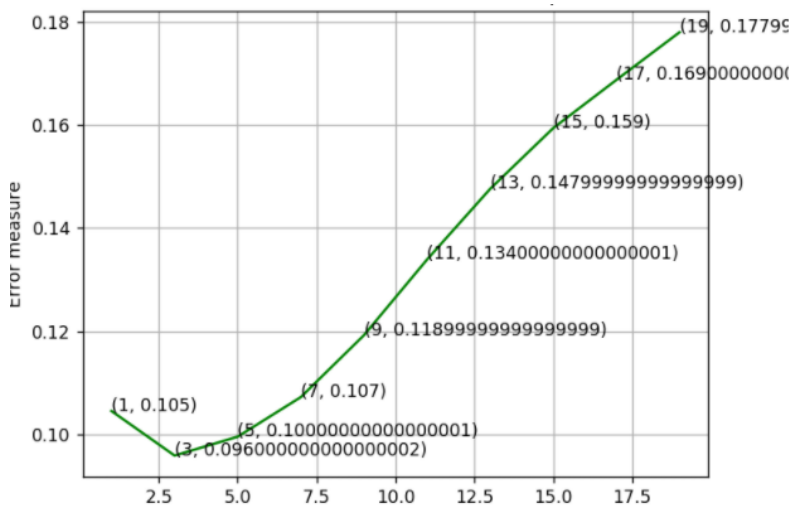

Figure 26. K vs Cross Validation Error for ASM Files

As shown in figure 26, the optimal k for this model happened to be k = 3. When we used this k, we yielded a cross validation log loss of .069, a test log loss of .089, and a model accuracy of 97.98%. The confusion, precision, and recall matrices are shown below.

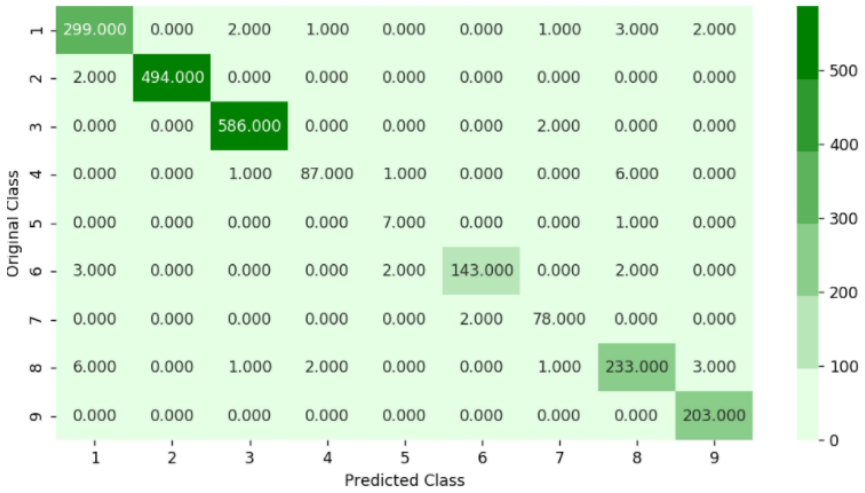

Figure 27. KNN Model Confusion Matrix for ASM Files

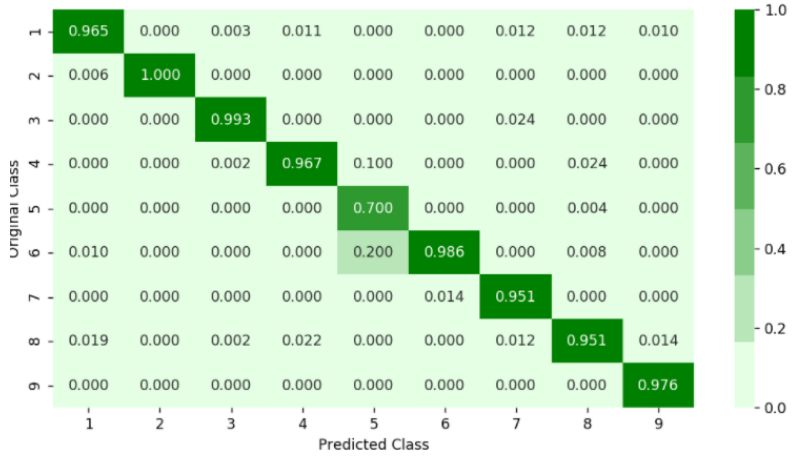

Figure 28. KNN Model Precision Matrix for ASM Files

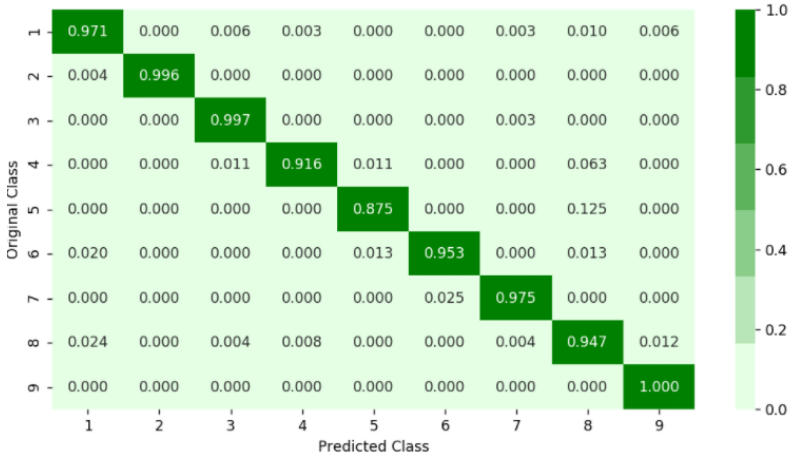

Figure 29. KNN Model Recall Matrix for ASM Files

After KNN, we explored Logistic Regression, and again first optimized the hyper parameter c before generating our model data. The optimal c can be seen in the figure below.

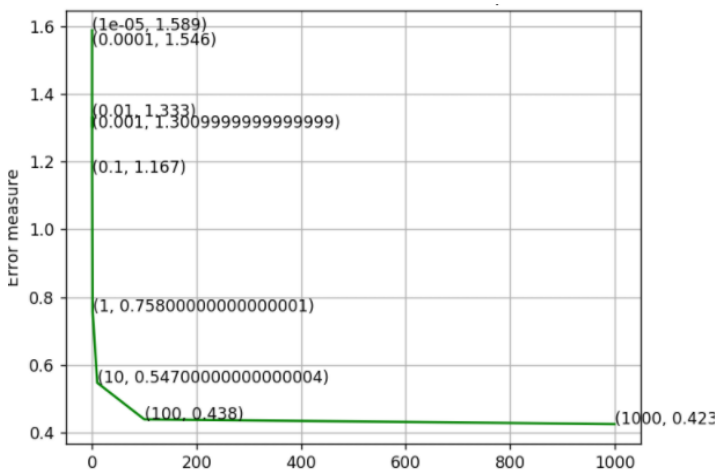

Figure 30. c vs Cross Validation Error for ASM Files

The optimal c for this model was c = 1000, and with this value, we yielded a cross validation log loss of .424, a test log loss of .416, and a model accuracy of 90.39%. The confusion, precision, and recall matrices are shown below.

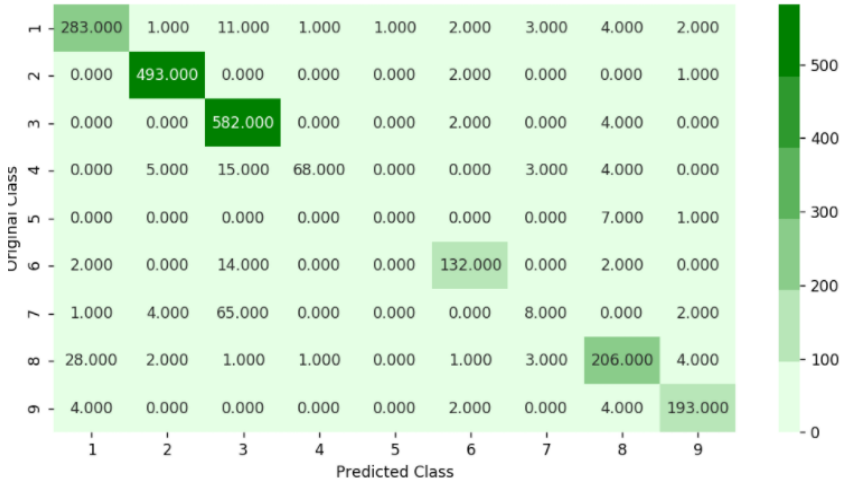

Figure 31. Logistic Regression Model Confusion Matrix for ASM Files

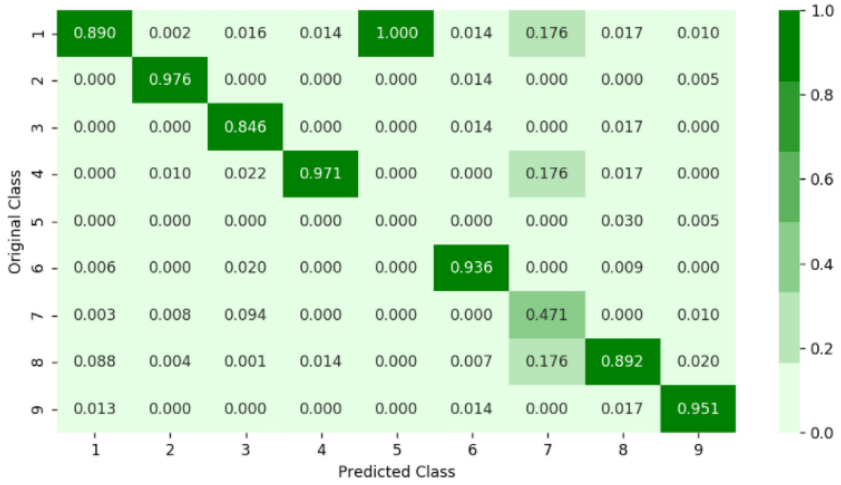

Figure 32. Logistic Regression Model Precision Matrix for ASM Files

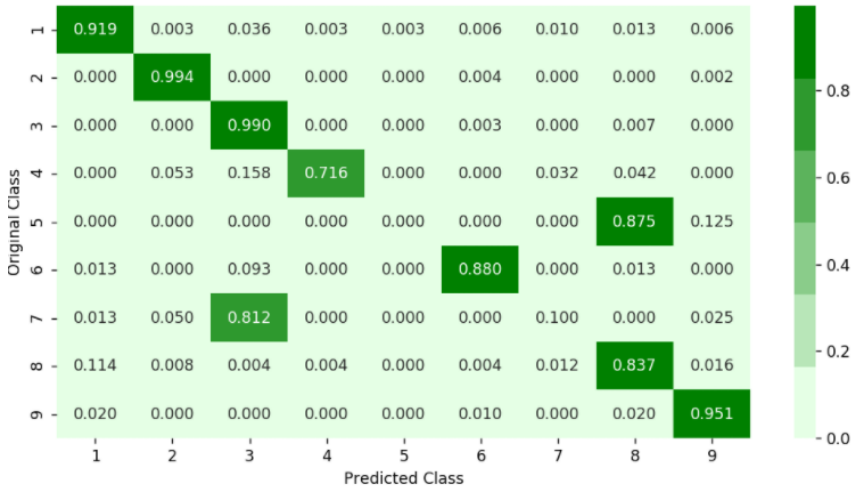

Figure 33. Logistic Regression Model Recall Matrix for ASM Files

The next model we explored was Random Forest with the number of estimators being the hyperparameter to be optimized. The optimal number of estimators can be found in the figure below.

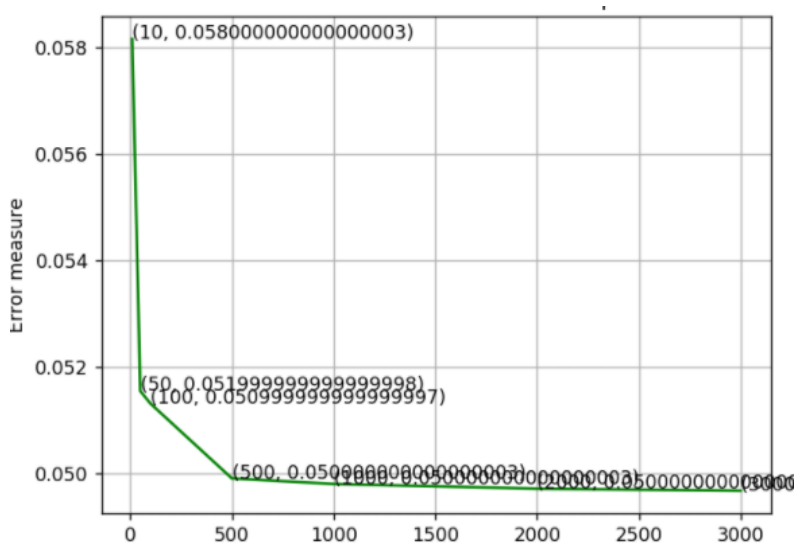

Figure 34. N_estimators vs Cross Validation Error for ASM Files
(Random Forest)

The optimal number of estimators for this model was n_estimators = 3000. With this number of estimators, we yielded a cross validation log loss of .05, a test log loss of .057, and a classification accuracy of 98.85%. The confusion, precision, and recall matrices are shown below.

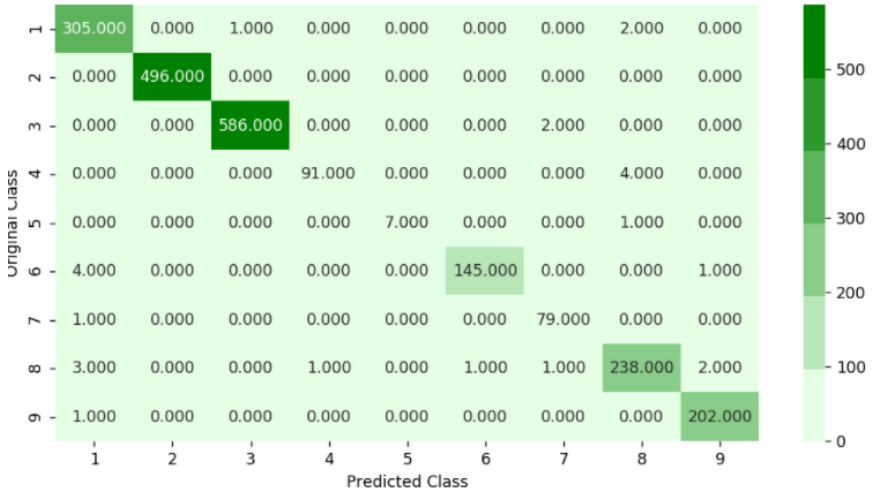

Figure 35. Random Forest Model Confusion Matrix for ASM Files

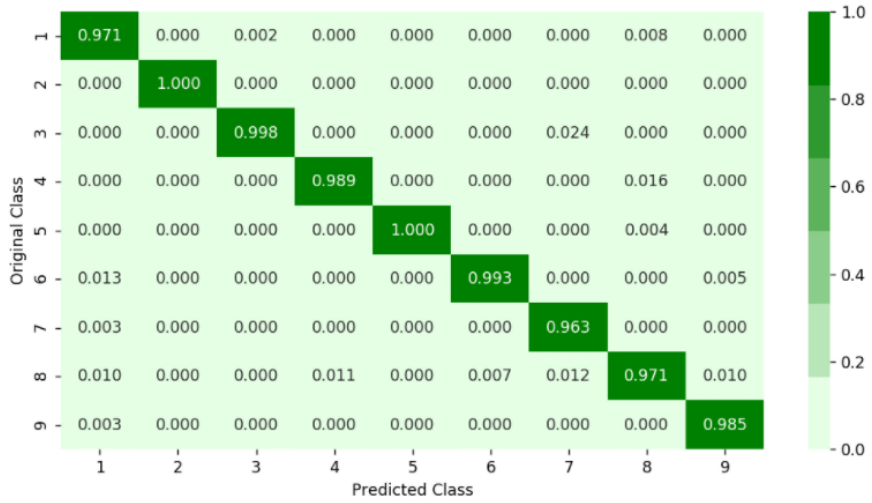

Figure 36. Random Forest Model Precision Matrix for ASM Files

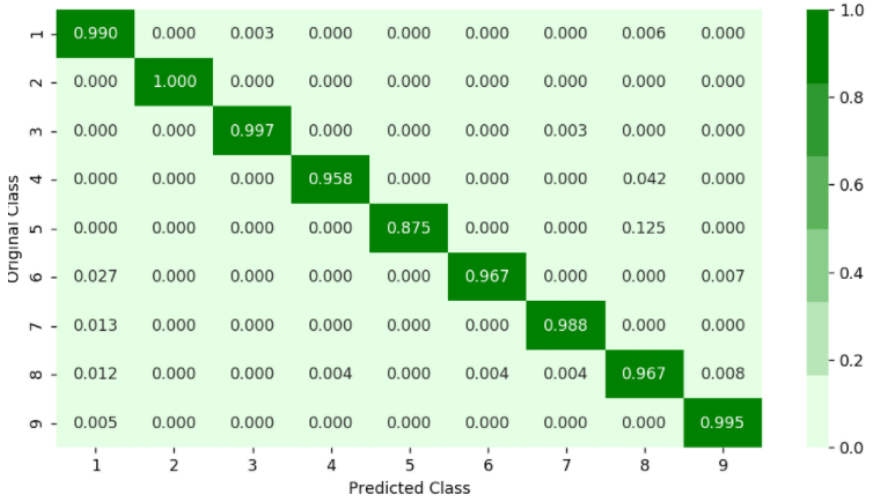

Figure 37. Random Forest Model Recall Matrix for ASM Files

The last model we applied on the asm file dataset was XGBoost with hyperparameter tuning on the number of estimators. The optimal number of estimators can be found in the figure below.

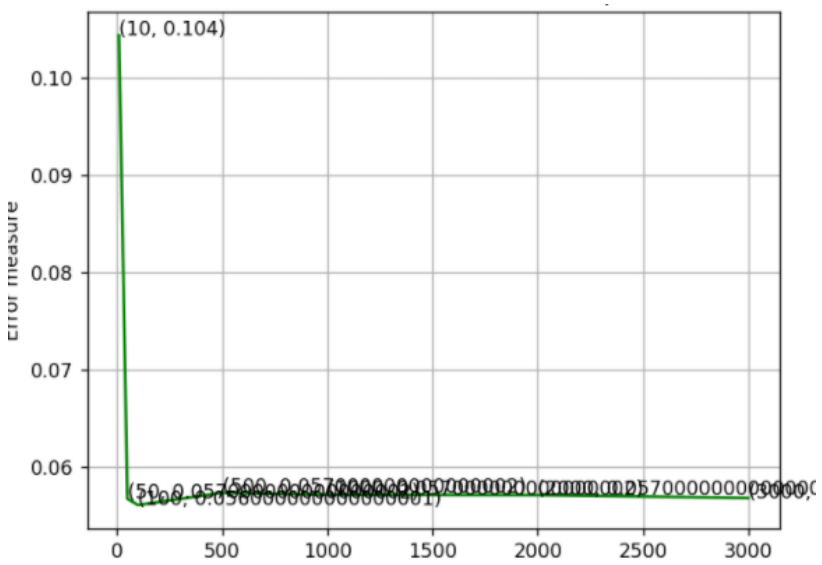

Figure 38. N_estimators vs Cross Validation Error for ASM Files (XGBoost)

The optimal number of estimators for this model was n_estimators = 100. With this number of estimators, we yielded a cross validation log loss of .056, a test log loss of .049, and a model accuracy of 99.13%. The confusion, precision, and recall matrices are shown below.

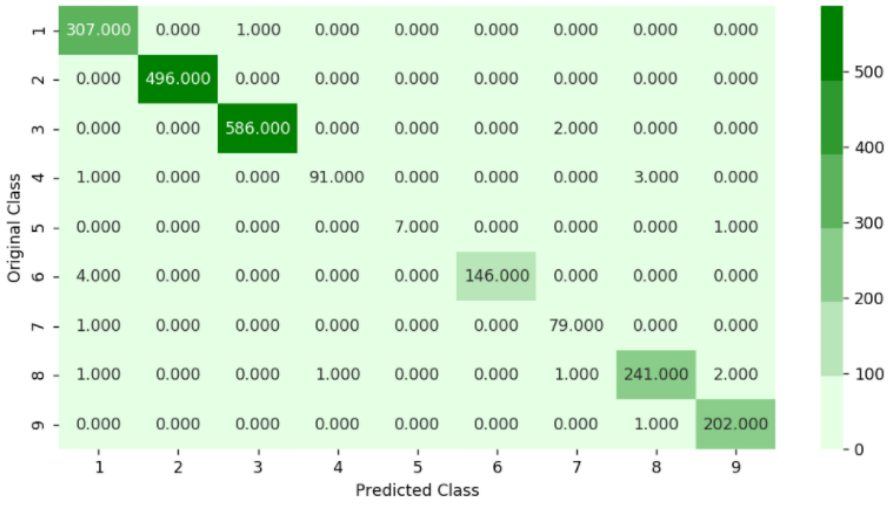

Figure 39. XGBoost Model Confusion Matrix for ASM Files

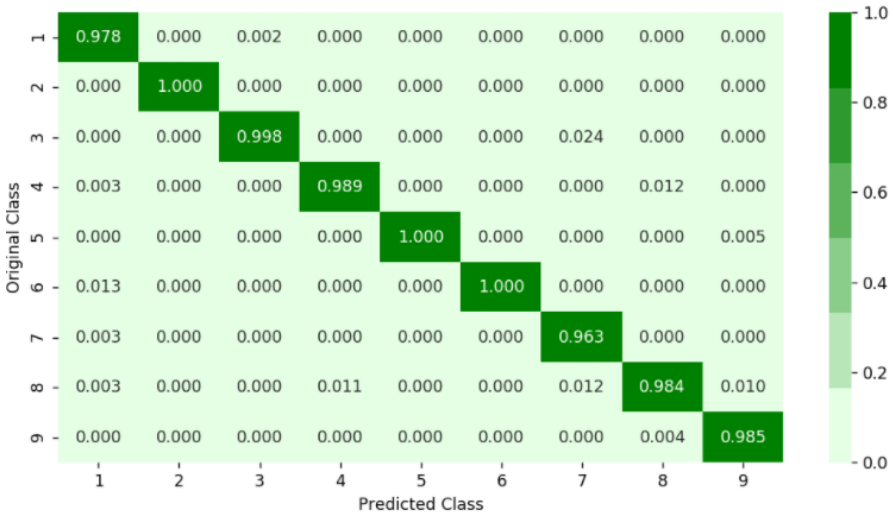

Figure 40. XGBoost Model Precision Matrix for ASM Files

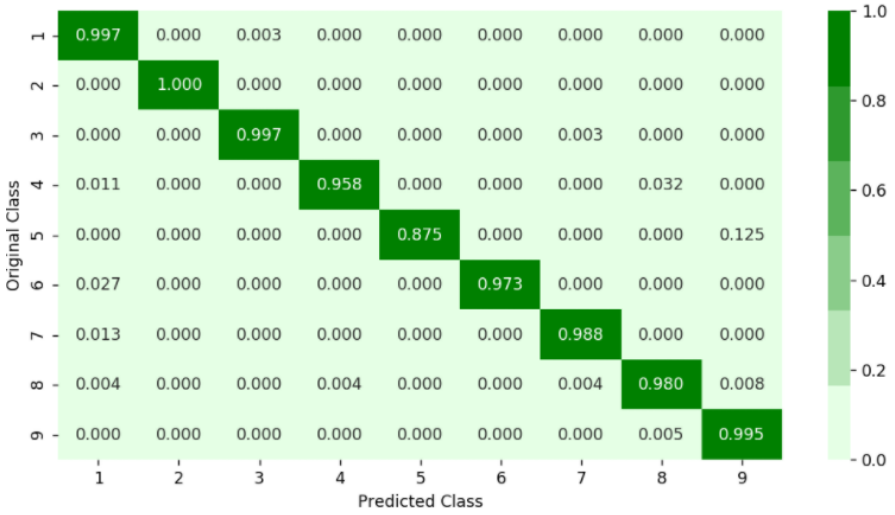

Figure 41. XGBoost Model Recall Matrix for ASM Files

After examining the data extrapolated from each model, we again reached the conclusion that XGBoost provided the best model as it provided the highest accuracy, so we again optimized the model further using RandomSearch. With the fully optimized XGBoost, cross validation log loss was .05, and test log loss was .048.

Once we completed our analysis on both the byte files and the asm files, we decided to merge both asm and byte file features together to explore if it could provide further improvement. Once we merged the features, our dataset had a dimensionality of 307, and the t-SNE multivariate analysis proved that this method could yield favorable results as seen in the figure below.

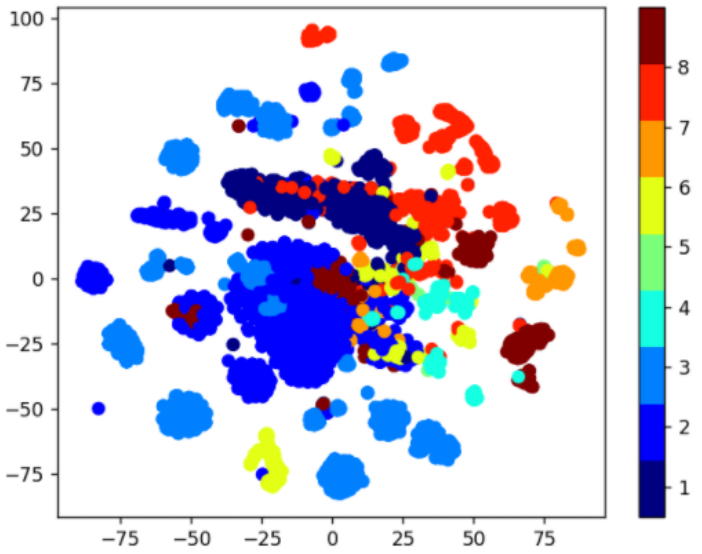

Figure 42. t-SNE Visualization for Byte+ASM Files (perplexity = 50)

As shown in the figure, the classes are substantially more distinct from each other than in the previous t-SNE visualizations. As Random Forest and XGBoost have consistently been our best models across both byte and asm files, we decided to apply both to this new dataset and find what log loss is yielded from these models. We began by finding the optimal number of estimators for Random Forest, which can be seen in the figure below.

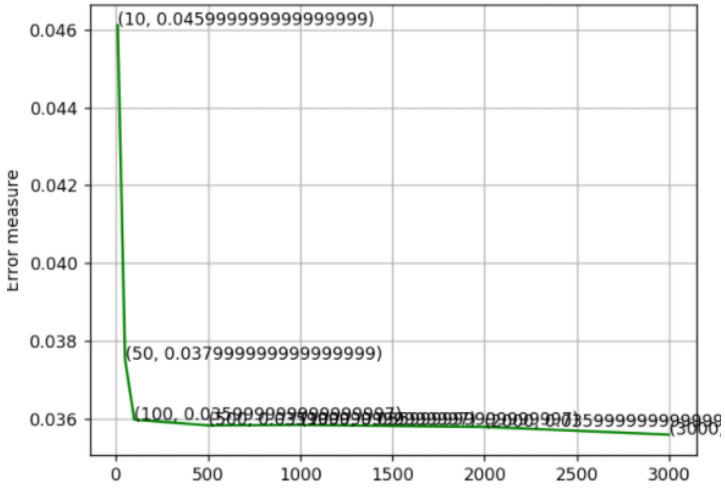

Figure 43. N_estimators vs Cross Validation Error for Byte+ASM Files (Random Forest)

Here we found that the optimal number of estimators for this model was n_estimators = 3000. With this optimal number of estimators, we yielded a cross validation log loss of .036 and a test log loss of .04.

We then applied XGBoost on this dataset, and after performing hyperparameter tuning, we yielded a cross validation log loss of .0316 and a test log loss of .0323 with the optimal number of estimators, which was also n_estimators = 3000. The figure for the hyperparameter tuning is shown below.

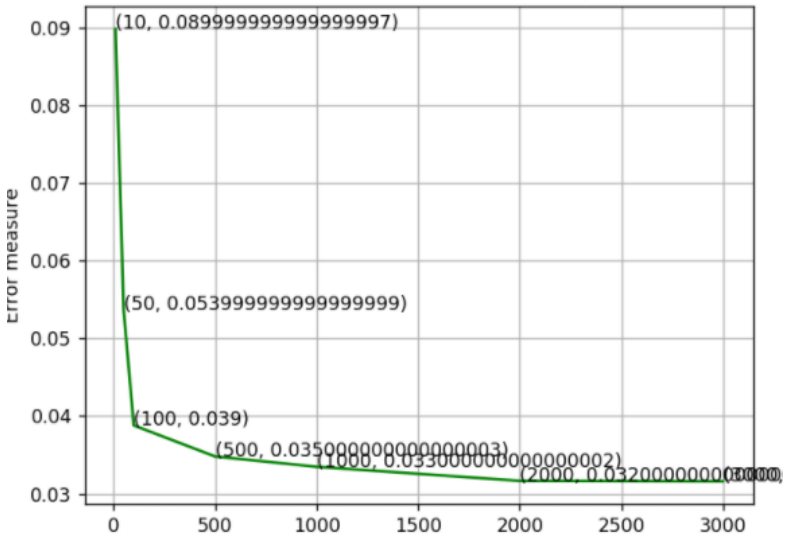

Figure 44. N_estimators vs Cross Validation Error for Byte+ASM Files (XGBoost)

We again performed RandomSearch optimization on the XGBoost model and yielded a new train loss of .0121, a new cross validation log loss of .0345 and a new test log loss of .0317.

V. Conclusion

In this work, we have seen how machine learning algorithms can be used to efficiently categorise the malware into different classes. Initially, we have used BYTE files in our dataset to build models. We have used unigram BoW to featurize the data and we have built models on this. Prior to modelling, we used a Random model as a baseline. We have used K Nearest Neighbors, Logistic Regression, Random Forest Model and XGBoost model and came to know that tree based models perform well with best log loss of 0.08 with Random Forest.

We have extended the algorithms on ASM Files by featurizing the files using a unigram Bag of Words, we then came to know that we have a best model with 0.87% of misclassified points on XgBoost and log loss 0.056. Furthermore, we have extended the modelling using ASM and BYTE file and built a feature set, afterwhich, the best model had a log loss of 0.0317.

To extend this paper, one can try bi-grams or tri-grams on the files, build a new feature set and then try XGBoost and other tree based models.

References

[1] Shabtai et al. (2009), Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey

[2] Bazrafshan et al. (2013) A survey on heuristic malware detection techniques

[3] Souri et al. (2018), A state-of-the-art survey of malware detection approaches using data mining techniques

[4] Kaggle Winner's blog http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/

[5] Mansour et al (2016) Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification

[6] First place solution in Kaggle competition: https://www.youtube.com/watch?v=VLQTRlLGz5Y

[7] Malware Detection and Classification using Machine Learning https://github.com/dchad/malware-detection

[8] Lakshmanan et al. Malware Images: Visualization and Automatic Classification http://vizsec.org/files/2011/Nataraj.pdf

[9] Lakshmanan et al. https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_pIB6ua?dl=0

Project Code

https://drive.google.com/drive/folders/1rF6VHeGRFgMV5rZN1vBMB_smuD_0fmN2