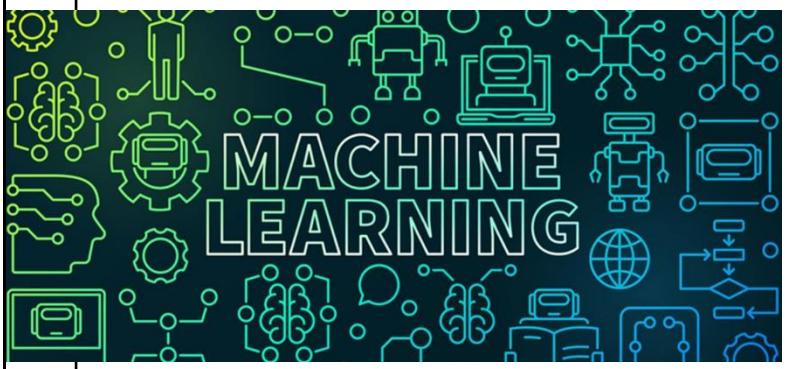
*Prof. Balaji Padmanabhan*

*Team Members:*

*Suryateja Chalapati*

# ISM6251: Malware Detection



## 1. INTRODUCTION

Malware, short for "malicious software," refers to any intrusive program created by cybercriminals (commonly referred to as "hackers") with the intent of stealing data and causing harm or destruction to computers and computer systems. Viruses, worms, Trojan horses, spyware, adware, and ransomware are all examples of prevalent malware. Data has been exfiltrated in large quantities because of recent malware attacks.

Some of the types of malwares out there are as follows:

1. **Virus:** Mostly transferred via the email to infect a device
2. **Scareware:** Alarming messages all over the internet that employ scare tactics to catch victims
3. **Spyware:** Programs installed to capture private information without consent
4. **Ransomware:** An attack where the attacker demands a ransom after hacking a person's system
5. **Worms:** Exploitive programs that can transmit from machine to machine by making countless copies
6. **Trojans:** Programs that steal personal data then use it for malicious intentions
7. **Adware:** Unwanted ads that pop-up at random to prompt unconsented install and steal personal information
8. **Fileless malware:** Software that uses legitimate programs to infect a system etc.

## 2. PRIOR WORK

We started by looking at a problem that is more prevalent today. Based on this we chose malware detection as it is one of the most pressing issues currently with cyber-attacks being at an all-time high. We wanted to build a machine learning model that can be used to flag malwares. Microsoft, Kaspersky and a lot of other big tech firms use classification-based machine learning techniques to flag malicious software. Based on our research and reading various papers on the topic, we found out that the highest accuracy achieved by the machine learning algorithms in detecting malicious software is over 98%. So, our end goal is not to improve the accuracy of the algorithms as that would

require a lot of processing power. Instead, we decided on 9 classes of malicious software's that have the most hit rate and build a model that can classify based on those 9 classes.

As discussed in class, we divided our work in two levels:

1. **Level - 1:** Build a model using 2 classes (Predict if a software is malicious or not)
2. **Level – 2,3:** Build a model using 9 classes (Predict if a software belongs to one of the 9 classes of the malicious software's)

## 3. DATASET OVERVIEW

Data Source: https://www.kaggle.com/c/malware-classification

We have two files for every malware

- .asm file
- .bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)

Total train dataset consist of 200GB data out of which 50Gb of data is .bytes files and 150GB of data is .asm files. We have a total of 10,868 .bytes files and 10,868 asm files in total 21,736 files. The dataset is unbalanced and there are 9 types of malwares (9 classes) in our data.

Types of Malwares:

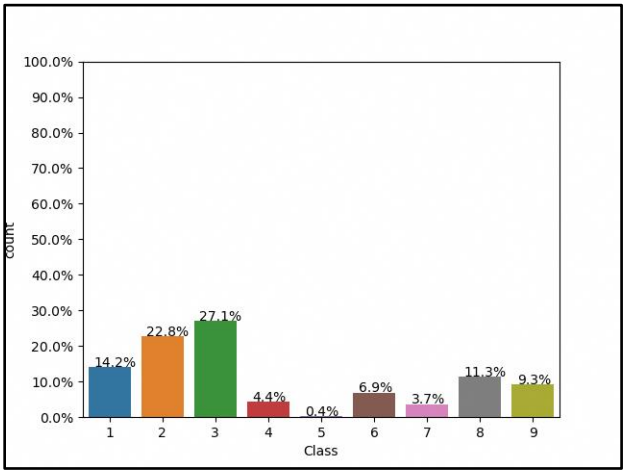
1. Ramnit
2. Lollipop
3. Kelihos_ver3
4. Vundo
5. Simda
6. Tracur
7. Kelihos_ver1
8. Obfuscator.ACY
9. Gatak

Feature Extraction:

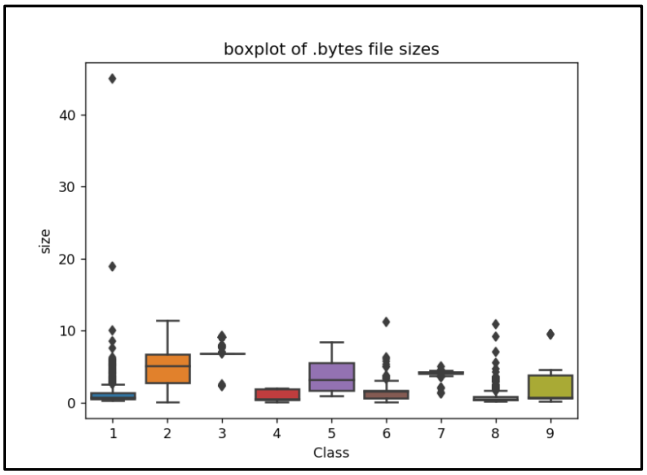
1. N-gram features of .bytes file: We tried using uni-gram (About 256 features).
2. Extract size of both .byte and .asm files.
3. For .bytes file: Extract 9 features converted to values between 0 and 1 where the sum of the values should be equal to 1.

## 4. EXPLORATORY ANALYSIS

The dataset is unbalanced which we can see from the plot below:




The boxplot of .bytes file size distributions:

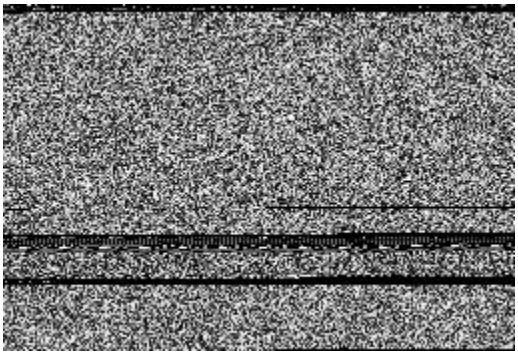


## 5. LEVEL - 1: ANALYSIS ON A 2-CLASS DATA.

### PROCESS FLOW

Data Cleaning and Pre-processing:

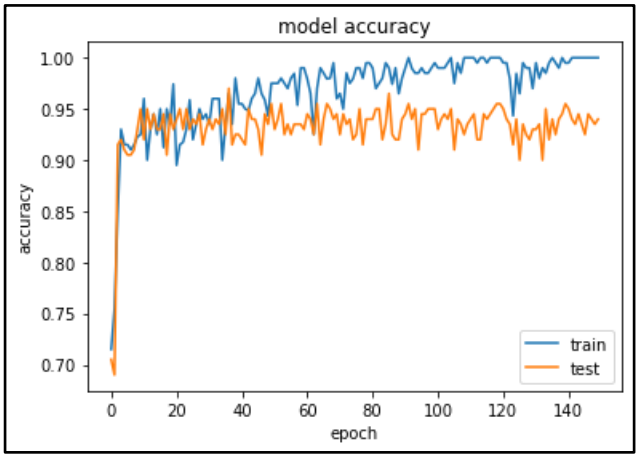
1. Convert the image files to grayscale image with a dimension of 256x256.
2. In total, we have about 1416 executables that were converted to grayscale images as shown below.

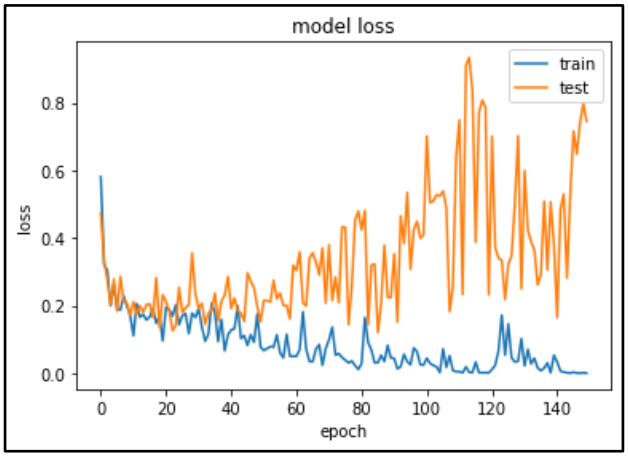


3. Train test split is 80-20% and the dataset have 50-50% split of malign ware and benign ware files.

### MODELS

1. Used CNN architecture with 6 layers, where activation function for first layers is ReLU and the last layer is Sigmoid. Optimizer is Adam and loss function is binary cross-entropy.
2. Accuracy and loss function plots as shown below:




### CONCLUSION

1. With 150 epochs, the model accuracy is 94% on test set.
2. Based on our analysis, we can summarize the following:

**Model Parameters:**

1. Epochs: 150.
2. Loss Function: Binary-Cross Entropy.
3. CNN Layers: 6.
4. Activation Function: ReLU.
5. Optimizer: Adam.

**Results:**

1. Accuracy(Test): 94%.
2. Highest Accuracy Achieved at Epoch 40.
3. Train and Test Show Similar Trends for Accruacy.
4. Loss Function on Test Shows Higher Deviation Which Could be Stablized by Using More Data.

## 6. LEVEL – 2,3: ANALYSIS ON A 9-CLASS DATA.

### PROCESS FLOW

Data Cleaning and Pre-processing:

1. Segregate .asm and .bytes files into separate folders. Then convert the .bytes files into text files to extract features.
2. In total, we have about 1780 files 8 gigs of .bytes files and 22 gigs of .asm files. For this example, we are working with a subset of the data and the entire analysis was run locally.

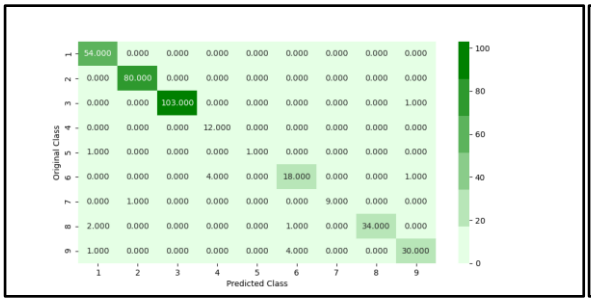
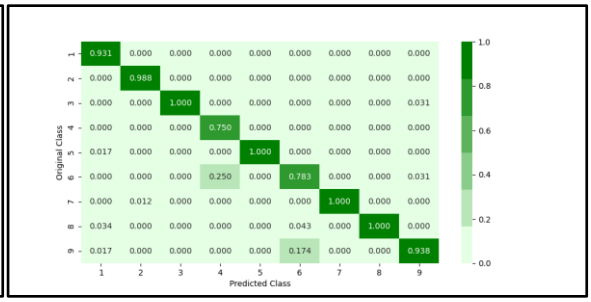
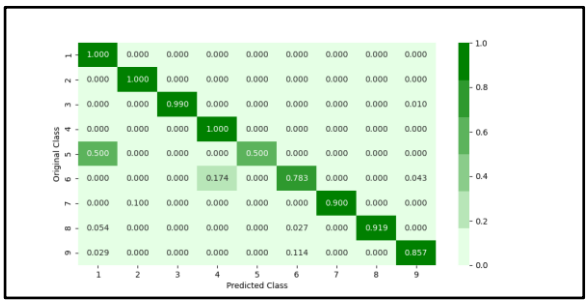
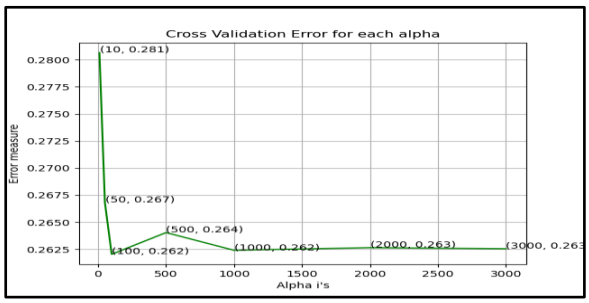
### MODELS

1. For feature extraction on both .bytes and .asm files we used bag of words methods which is also known as the unigram method. It is used to extract features from the .bytes files which have been converted to .txt files.
2. We have custom built a unigram bag of words to extract features. This is followed by train, cross validation, and test splits of the data. We used stratified data split for this analysis.
3. Then we built a number of machine learning models as follows:
    a. Random Model.
    b. K-Nearest Neighbor.
    c. Logistic Regression.
    d. Random Forest.

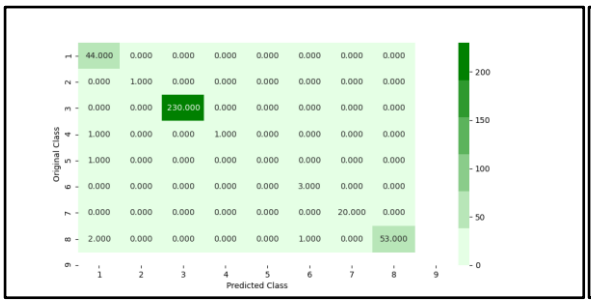
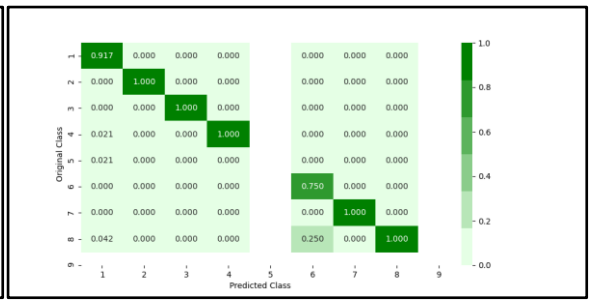
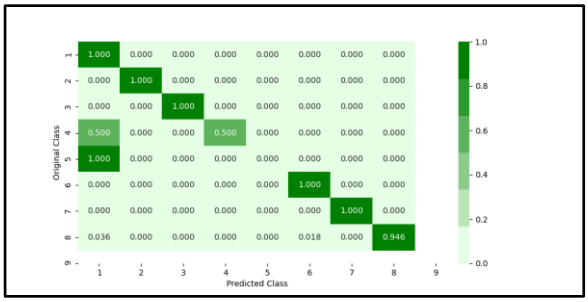
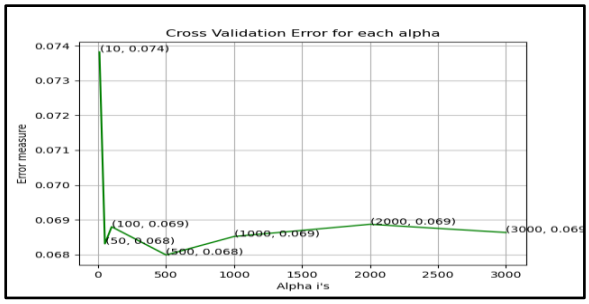
e. XGBoost.

f. XGBoost with Hyperparameters Tuning.

4. We print the confusion matric which is a 9x9 in this case considering the 9-classes and output the log-loss and misclassification rate for each model.
5. Below is a little snip of the Random Forest model confusion matrix that shows the best metrics for .bytes files.


6. Below is a little snip of the Random Forest model confusion matrix that shows the best metrics for .asm files.

## CONCLUSION

1. Model summary as shown below:

### 6. Analysis Summary

#### 6.1. ML Model Results

**Results for ML Analysis on Byte Files [Level - 2]**

Random Model:

1. Log-Loss on Cross Validation Split: 2.5
2. Log-Loss on Test Split: 2.4
3. Misclassified Points: 92.4

K-Nearest Neighbours Model:

1. Log-Loss on Train Split: 0.21
2. Log-Loss on Cross Validation Split: 0.26
3. Log-Loss on Test Split: 0.26
4. Misclassified Points: 4.2

Logistic Regression Model:

1. Log-Loss on Train Split: 0.91
2. Log-Loss on Cross Validation Split: 0.88
3. Log-Loss on Test Split: 0.87
4. Misclassified Points: 28.2

Random Forest Model:

1. Log-Loss on Train Split: 0.08
2. Log-Loss on Cross Validation Split: 0.16
3. Log-Loss on Test Split: 0.14
4. Misclassified Points: 2.8

XGBoost Model:

1. Log-Loss on Train Split: 0.08
2. Log-Loss on Cross Validation Split: 0.15
3. Log-Loss on Test Split: 0.12
4. Misclassified Points: 1.4

XGBoost Hyperparameter Tuned Model:

1. Log-Loss on Train Split: 0.08
2. Log-Loss on Cross Validation Split: 0.15
3. Log-Loss on Test Split: 0.11

**Results for ML Analysis on ASM Files [Level - 3]**

Logistic Regression Model:

1. Log-Loss on Train Split: 0.13
2. Log-Loss on Cross Validation Split: 0.14
3. Log-Loss on Test Split: 0.16
4. Misclassified Points: 1.96

Random Forest Model:

1. Log-Loss on Train Split: 0.05
2. Log-Loss on Cross Validation Split: 0.06
3. Log-Loss on Test Split: 0.09
4. Misclassified Points: 1.4

2. Based on our analysis, we can summarize the following:

#### 6.2. Best Model & Future Scope

**Results:**

1. For .bytes files, the best model is the Hyperparameter tuned XGBoost Model.
2. For .asm files, the best model is the Random Forest Model.
3. For both .bytes and .asm files, the best model is the Random Forest Model if we want to consider a single ML model for both file types.

**Future Scope:**

1. We can try combining the features for both .bytes and .asm files and build ML models on a single combined featureset.
2. For this analysis we used unigram feature extraction. We can try bi-gram or tri-gram techniques to extract more features and build more accurate models.
3. We used a small part (30GB) of the original dataset (200GB) for this analysis. We can repeat the same analysis on cloud using the entire 200GB dataset.

## 7. CONCLUSION

From our analysis, for .bytes files the best model is Hyperparameter tuned XGBoost model with log-loss of 0.15 on cross-validation split and 0.11 on test split. Misclassified points of about 1.4. For .asm files the best model is Random Forest model with log-loss of 0.06 on cross-validation and 0.09 on test set. Misclassified points of about 1.4. The performance metric log-loss needs to be minimized, so the closer it to zero the better. If we need to find the best model, then Random Forest is the best for both .bytes and .asm files.

For future scope, we can try building a combined feature set for .bytes and .asm files and build ML models and see which model performs the best in terms of log-loss and misclassification points. We can also try other methods like bi-gram and tri-gram to extract more features to minimize multi loss even more. Our analysis was on limited dataset of 30GB, we can do the same analysis on the entire dataset which is around 200GB on cloud and compare model performance in terms of metrics and time taken.

**Note:** There is an issue with showing plots on the jupyter notebook, so please run the file to see the updated plots.

## 8. REFERENCES


- https://www.cisco.com/c/en/us/products/security/advanced-malware-protection/what-is-malware.html
- https://www.mcafee.com/en-us/antivirus/malware.html
- https://stackoverflow.com/a/29651514
- https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
- https://stackoverflow.com/a/18662466/4084039
- http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
- http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
- http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
- https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/