MICROSOFT MALWARE PREDICTION

A PROJECT REPORT

Submitted by

ATHARV SHAH [Reg No:RA1811028010099]

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that the Industrial training report titled "**Microsoft Malware Prediction** " is the bonafide work of "**RA1811028010099** submitted for the course 18CSP103L Seminar – I. This report is a record of successful completion of the specified course evaluated based on literature reviews and the supervisor. No part of the Seminar Report has been submitted for any degree, diploma, title, or recognition before.

SIGNATURE
Faculty Name
Assistant Professor
Dept. of Computer Science & Engineering

SIGNATURE
Name
Academic Advisor/Hod
Dept. of Computer Science & Engineering

Index

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my guide , Srividhya S and M Anand for their valuable guidance, consistent encouragement, personal caring, timely help and providing me with an excellent atmosphere for doing research. All through the work, in spite of their busy schedule, they cordially supported me for completing this research work.

Atharv Shah

ABSTRACT

Malware is a malicious software written with intend of doing harm to data, device or to the people. Predicting the whether the software is malicious or not is very common issue's in all most every Tech Companies.

In this project we will discover the ways to Predict whether the software is malicious or not using traditional Machine Learning Algorithms and Ensemble models on Bytes and ASM Files as the data. Data is taken from a Hiring Kaggle Competition by Microsoft. It consists of 9 classes of Malware which are Rammit, Lollipop, Kelihos_ver3, Vundo, Simda Tracur, Kelihos_ver1 Obfuscator.ACY, Gatak. For every Malware we have two file .asm file and. Bytes file. Total size of data set is 200GB data out of which 50GB is. byte file and 150GB is .asm files.

Performance metric being used here is Multi-class log loss and Confusion Matrix.

In Particular, it will describe effective way to work with .bytes , .asm files and feature engineer them effectively.

KEYWORDS

Machine learning, Feature Engineering, Bigram Bag of words, TSNE, Logistic Regression, KNN, Random Forest, XGBoost.

LITERATURE SURVEY

1) Review research paper

It's a research paper of Winner of Microsoft Malware Classification Challenge. Here he discusses about the all the major process involved in the attaining 1st position in the completion. Dataset is almost a half terabyte. It consists of 9 types of Malware Ramnit, Lollipop, Kehilos_ver3, Vundo, Simbda, Tracur, Kehilos_ver1, Obfuscater.ACY, Gatak. Raw data contains the hexadecimal representation of the file's binary content, without the header. The dataset also includes a metadata manifest, which is a log containing various metadata information extracted from the binary, such as function calls, strings, etc. This was generated using the IDA disassembler tool. The original question posed to participants was to classify malware to one of the 9 classes. The dataset can be downloaded from the competition website.

In this paper, they have provided a short description of the characteristics of the Microsoft Malware Classification Challenge dataset. This dataset is becoming a standard dataset with more than 50 papers citing it. We enumerated these references as much as possible and compared their main contributions with respect to the dataset. The comparison helps the understanding of what the

existing contributions are, and what the potential research directions can be. The authors aim to keep the reference table updated.

2) XGBoost

This paper discusses very famous and important Ensemble Machine learning algorithm which is XGBoost. In this paper, we describe a scalable end-to-end tree boosting system called XGBoost, which is used widely by data scientists to achieve state-of-the-art results on many machine learning challenges. We propose a novel sparsity-aware algorithm for sparse data and weighted quantile sketch for approximate tree learning. More importantly, we provide insights on cache access patterns, data compression and sharing to build a scalable tree boosting system. By combining these insights, XGBoost scales beyond billions of examples using far fewer resources than existing system. In this paper they have described a scalable tree boosting system which is widely used by data scientists and provides state-of-the-art results on many problems. We proposed a novel sparsity aware algorithm for handling sparse data and a theoretically justified weighted quantile sketch for approximate learning. Our experience shows that cache access patterns, data com-press ion and sharing are essential elements for building scalable end-to-end system for tree boosting. These lessons can be applied to other machine learning systems as well. By combining these insights, XGBoost is able to solve real-world scale problems using a minimal amount of resources.

3) Malicious Software

It’s a task of Malware Classification, so for that first we need understand the problem statement very well. So this blogs in detail discusses everything about the malware like history of malware, Morris worm etc.

Malware includes Computer virus, Trojan Malware, Spyware, Ransomware, wiper Malware, Computer worm, adware, botnet, cryptocurrency miner malware, lifeless malware, Internet of things Malware.

Some of the most basic cybersecurity practices can go a long way to protecting systems -- and their users -- from falling victim to malware. Simply ensuring software is patched and up to date, and all operating system updates are applied as quickly as possible after they're released, will help protect users from falling victim to attacks using known exploits.

Time and again, delays in patching have led to organizations falling victims to cyberattacks, which could've been prevented if patches had been applied as soon as they were released.

4) K-Nearest-Neighbor

This article in details discusses about the classic machine learning algorithm which is K-nearest Neighbor (KNN)
K-nearest Neighbor is a Supervised machine learning algorithms.
It can be used for both Classification and regression based tasks,

However, it is more widely used in classification problems in the industry. To evaluate any technique, we generally look at 3 important aspects Ease to interpret output, Calculation time, Predictive Power.

KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

We can find K by Hyper-parameter tuning, As the values of K increases model start to Over-fit and vice versa

In the case of classification and regression, we saw that choosing the right K for our data is done by trying several Ks and picking the one that works best.

5) t-SNE

It’s an article about the very famous dimensionality reduction technique t-SNE. After all the data processing, feature engineering we need to visualize it on the plane, But technically it’s not possible to visualize more the 3 dimensional, So we reduce all dimensions/total features into a 2D space, using t-SNE.

-Distributed Stochastic Neighbor Embedding (t-SNE) is an non-linear technique primarily used for data exploration and visualizing high-dimensional data. In simpler terms, t-SNE gives you a feel or intuition of how the data is arranged in a high-dimensional space. It was developed by Laurens van der Maatens and Geoffrey Hinton in 2008.

The t-SNE algorithm calculates a similarity measure between pairs of instances in the high dimensional space and in the low dimensional space. It then tries to optimize these two similarity measures using a cost function.

t-SNE is used in different domain such as climate research, computer security, bioinformatics, cancer research, etc. t-SNE could be used on high-dimensional data and then the output of those dimensions then become inputs to some other classification model. t-SNE could be used to investigate, learn, or evaluate segmentation.

6) Text Processing

This blog discusses about one of the Text-Processing Technique.

We are given with the bytes file which is a Hex Code file, which can be considered as a text and for processing we are using here Bag of Words.

Bag-of-words model, or Bow for short, is a way of extracting features from text for use in modeling, such as with machine learning algorithms.

The approach is very simple and flexible, and can be used in a myriad of ways for extracting features from documents.

A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things vocabulary of known words, to measure of the presence of known words.

To represent a word we can represent it 1,2, 3,. . .. N sequence of words.

So we are going to use the Bi-Gram BOW i.e. Considering two sequence of words as one entity and processing on the same method on all entity.

7) Random Forest

This Blog discusses about the another Ensemble Machine learning algorithm which is Random Forest.

The random forest is one of many supervised machine learning algorithms that can be used to predict an outcome, whether that outcome is a number (a prediction process we call *regression*) or represents membership in a group (known as *classification*). It's a non-parametric model, and thus does not make any strong assumptions about the data distribution, or specify any parameters (like the slope of a linear regression line).

random forest also uses the *ensemble* learning method, meaning it combines many underlying models into one, using all the individual models' predictions together. The models it combines together are called *decision trees* (hint: where the 'forest' name comes from). Consequently, to truly understand a random forest, we must first understand the component models that form its foundation.

Introduction

All the modules are implemented in order to improve the Performance metrics, Here the aim of the project is to predict whether the file contains malicious software or not.

This helps various companies to implement in this their check system, because some time malicious software is created in such a way it can bypass antivirus, so organization implement this system as security check over with other requirements.

The entire project has been implemented in python on Jupiter notebook for easy understanding of the code and clean implementation into the system.

Modules

It consists of 3 modules:

- Data Preprocessing and data formation
- Feature selection and feature Visualization
- Hyper Parameter tuning and modelling.

CODING AND TESTING

In this chapter we will be disusing about the coding and implementation part of the project.

In this project Python programming language is being used because it offers concise and readable code. While complex algorithms and versatile workflows stand behind machine learning and AI, Python's simplicity allows developers to write reliable systems. Python code is understandable by humans, which makes it easier to build models for machine learning.

This step includes the processing and formation of the data :

1) Data consisted of Bytes and ASM Files together in one folder, so for first asm files were separated from byte's files in another folder.

```
## Seperating the asm files and byte files

source = 'train/'
destination = 'byteFiles'

# we will check if the folder 'byteFiles' exists if it not there we will create a folder with the sam
if not os.path.isdir(destination):
    os.makedirs(destination)

# if we have folder called 'train' (train folder contains both .asm files and .bytes files) we will r
# for every file that we have in our 'asmFiles' directory we check if it is ending with .bytes, if ye
# 'byteFiles' folder

# so by the end of this snippet we will separate all the .byte files and .asm files
if os.path.isdir(source):
    data_files = os.listdir(source)
    for file in data_files:
        if (file.endswith("bytes")):
            shutil.move(source+file,destination)
```

2) First processing was done on byte's file, they were basically the Hex Code files, so text processing was performed on top of it using CountVectorizer library in sklearn library, Bi-gram Bag of words was performed on all bytes files and 66000 features were extracted as Bigram Bag of words

```
vectorizer = CountVectorizer(lowercase=False,vocabulary=bigram_vocab,ngram_range=(2,2))
folder1_matrix = csr_matrix((2717,66049))

for i,file in enumerate(os.listdir('f1/')):
    f = open('f1/'+file)
    folder1_matrix[i,:] += csr_matrix(vectorizer.fit_transform([f.read().replace('\n',' ').lower()]))
    f.close()
folder1_matrix = folder1_matrix.todense()
folder1_data = pd.DataFrame(folder1_matrix,columns=bigram_vocab)
folder1_data['id'] = os.listdir('f1/')
folder1_data.to_pickle('f1.pkl')
```

3) Out of this 66000 features many features were of no use so it was necessary to perform feature selection, so after this I extracted top 500 features using random forest's in build feature selection object.

```
## We need to do feature extraction in order to remove curse of dimensionality

## making a random model to extract features

def select_feature(data):
    y = list(data['Class'].values)
    x = data.drop(['id','Class','size'],axis=1)
    select_model = RandomForestClassifier(n_jobs=-1)
    select_model.fit(x,y)
    all_columns = list(data.columns)
    feature_select = np.argsort(select_model.feature_importances_)[::-1]
    all_imp_features = np.take(all_columns,feature_select)

    return all_imp_features



    ## after making model
```

```
## Get important features

selected_features = select_feature(byte_bigramdata_all)
```

```
## Making data frame accordingly
final_byte_bigramdata_all = list(selected_features[:500])
final_byte_bigramdata_all.extend(['id','Class','size'])

final_bigram_data = byte_bigramdata_all[final_byte_bigramdata_all]

final_bigram_data.head()
```

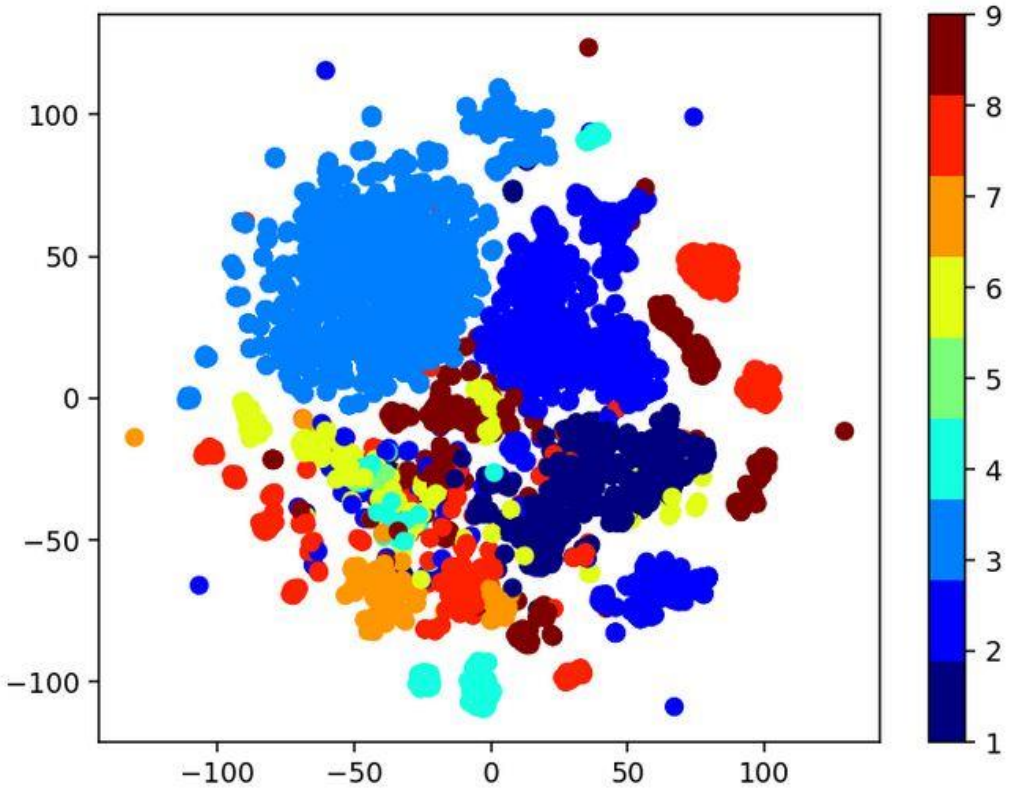
	ee 58	e5 87	71 ed	ee 52	f7 8c	f7 ad	2c 8c	fa b3	8a 06	45 08	...	00 3f	00 fe	f9 00	b9 10	8f 9a
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.011834	0.000019	0.003030	0.004016	0.005495	0.000432	0.049180	0.000036	0.003215	...	0.002601	0.002166	0.000951	0.000096	0.000000
2	0.000000	0.000000	0.000000	0.006061	0.000000	0.000000	0.000000	0.000000	0.000515	0.103952	...	0.000355	0.004831	0.000903	0.000000	0.000000
3	0.025641	0.035503	0.000173	0.018182	0.044177	0.054945	0.003459	0.147541	0.000089	0.001206	...	0.003310	0.000833	0.000475	0.000096	0.065693
4	0.000000	0.017751	0.000000	0.003030	0.000000	0.000000	0.000000	0.016393	0.000036	0.001340	...	0.002601	0.000250	0.000143	0.000038	0.007299

5 rows × 503 columns

4) After this visualization of all the features are very essential for checking whether the features are useful are not.

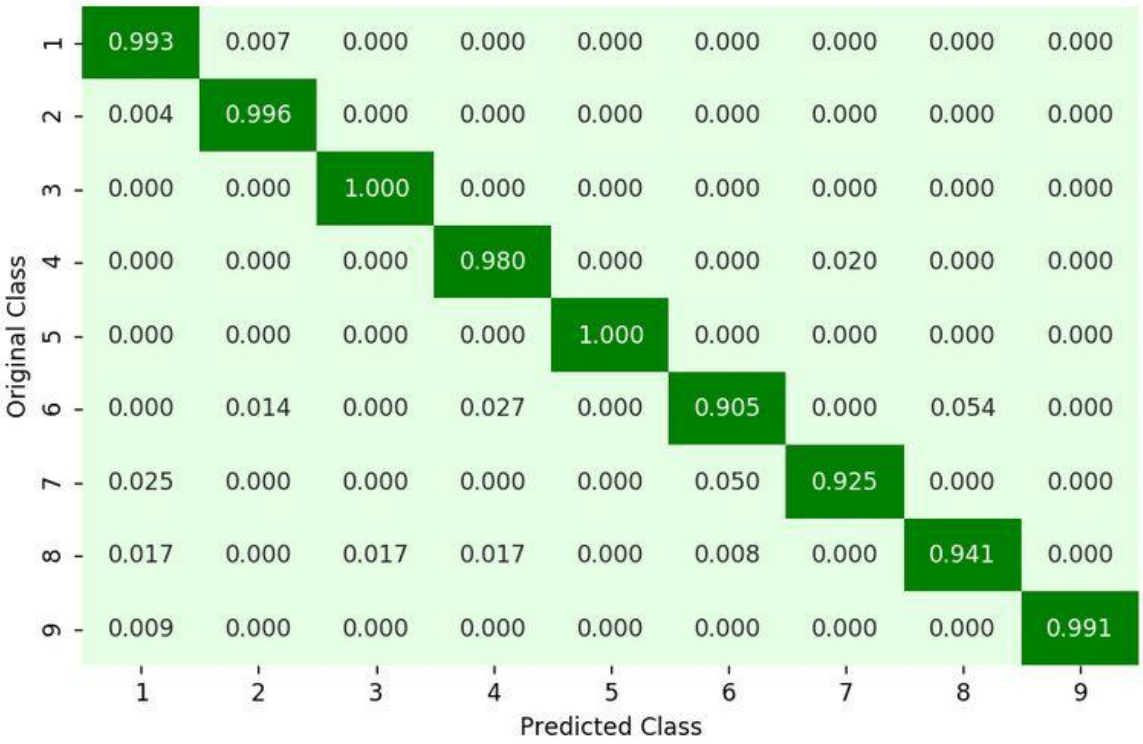
```
9]: from sklearn.manifold import TSNE

visualize_data = final_bigram_data.copy()
m_vis = TSNE(perplexity=10)
transform_vis = m_vis.fit_transform(visualize_data.drop(['id','Class','size'],axis=1))
x_axis = transform_vis[:,0]
y_axis = transform_vis[:,1]
plt.scatter(x_axis,y_axis,c=final_bigram_data.Class,cmap=plt.cm.get_cmap("jet",9))
plt.colorbar(ticks=range(10))
plt.show()
```

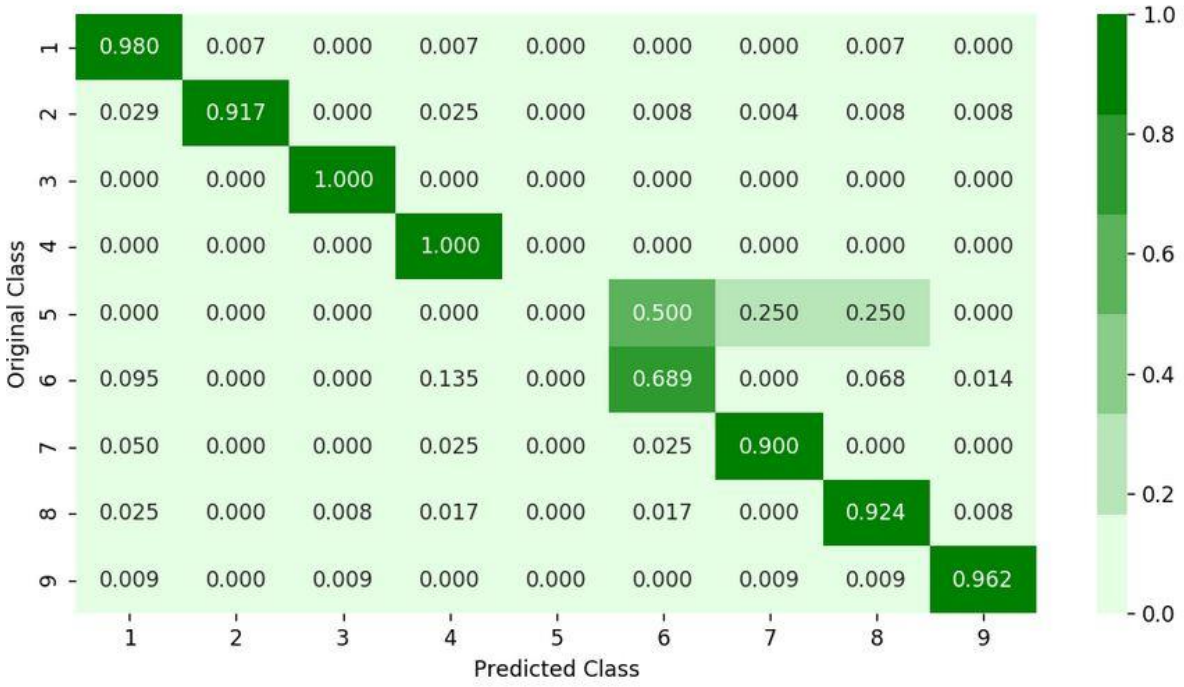
5) It was found out that features were working well as clusters were forming well for all the classes, but some of them were interacting with each other, which indicated that more featurization is needed

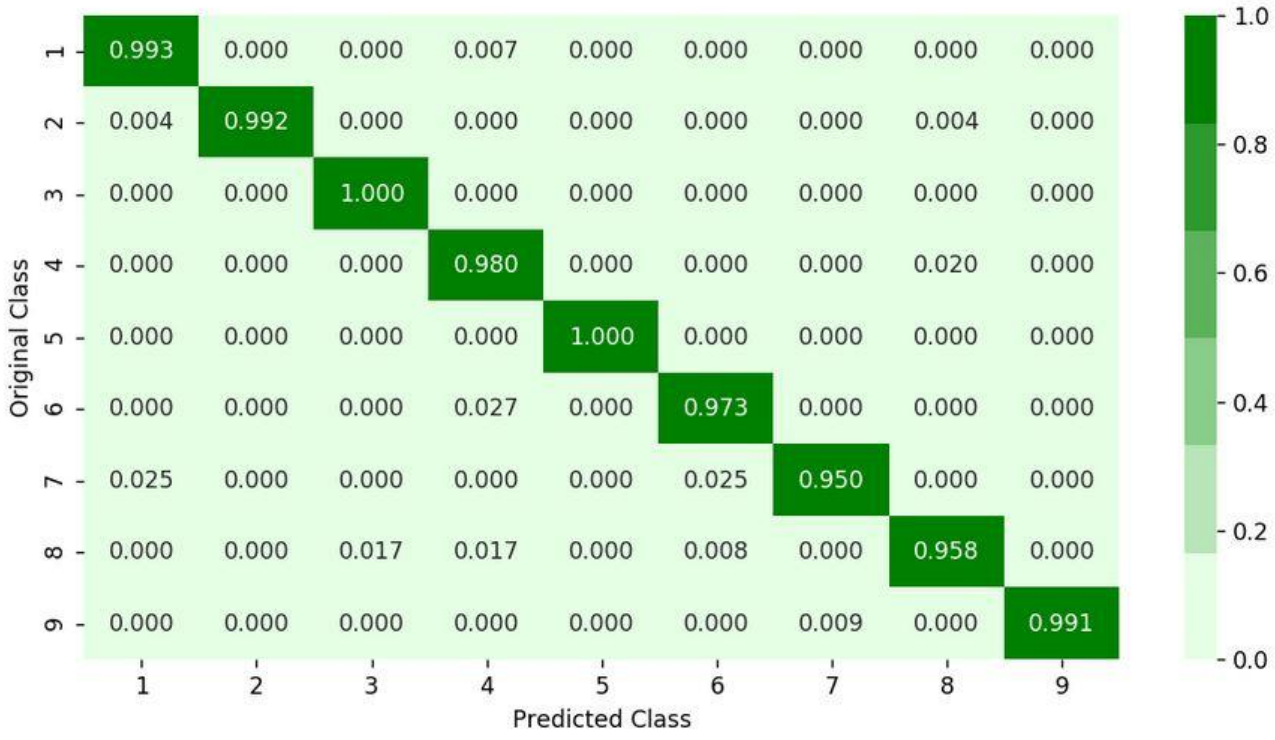


6) Logistic Regression, Random Forest and XGBoost were performed on the features extracted from all the byte's file

7) Confusion matrix formed by the Logistic regression, Random Forest, XGBoost are down respectively.


Original Class
Predicted Class
1 0.993 0.007 0.000 0.000 0.000 0.000 0.000 0.000 0.000
2 0.004 0.996 0.000 0.000 0.000 0.000 0.000 0.000 0.000
3 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000
4 0.000 0.000 0.000 0.980 0.000 0.000 0.020 0.000 0.000
5 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000
6 0.000 0.014 0.000 0.027 0.000 0.905 0.000 0.054 0.000
7 0.025 0.000 0.000 0.000 0.000 0.050 0.925 0.000 0.000
8 0.017 0.000 0.017 0.017 0.000 0.008 0.000 0.941 0.000
9 0.009 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.991
1 2 3 4 5 6 7 8 9


Original Class
Predicted Class
1 0.980 0.007 0.000 0.007 0.000 0.000 0.000 0.007 0.000
2 0.029 0.917 0.000 0.025 0.000 0.008 0.004 0.008 0.008
3 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000
4 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000
5 0.000 0.000 0.000 0.000 0.000 0.500 0.250 0.250 0.000
6 0.095 0.000 0.000 0.135 0.000 0.689 0.000 0.068 0.014
7 0.050 0.000 0.000 0.025 0.000 0.025 0.900 0.000 0.000
8 0.025 0.000 0.008 0.017 0.000 0.017 0.000 0.924 0.008
9 0.009 0.000 0.009 0.000 0.000 0.000 0.009 0.009 0.962
1 2 3 4 5 6 7 8 9
1.0
0.8
0.6
0.4
0.2
0.0



As we can see Best result was given by XGBoost on performance metrics Confusion Matrix.

As of now featurization was only done on bytes, so here I have started featurization on ASM files.

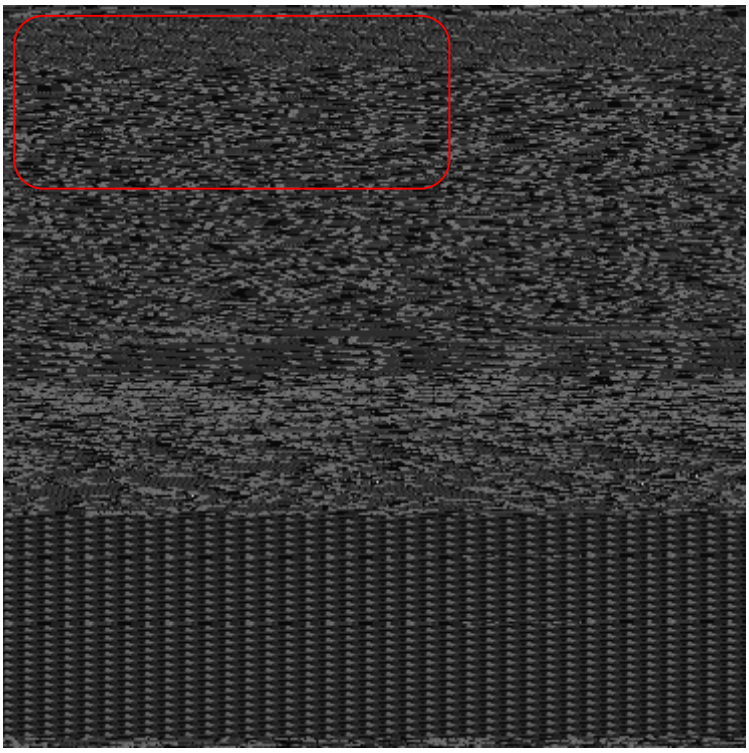
8) In order to use asm files for featurization, I have converted each asm file to image, and extracted first 800 pixels of that image.

```
## https://towardsdatascience.com/malware-classification-using-machine-learning-7c648fb1da79
import time,array
import imageio,cv2
from tqdm import tqdm_notebook as tqdm

def asm_image():
    for asmfiles in os.listdir('asmfiles/'):
        filename = asmfiles.split('.')[0]
        file = codecs.open('asmfiles/'+asmfiles,'rb')
        filen = os.path.getsize('asmfiles/'+asmfiles)

        width = int(filen**0.5)
        rem = (filen/width)
        ar = array.array('B')

        ar.frombytes(file.read())
        file.close()

        reshaped = np.reshape(ar[:width * width], (width, width))  # creating the shape of image
        reshaped = np.uint8(reshaped)
        imageio.imwrite('asm_images/' + filename + '.png',reshaped)  # i've created a new folder and adding to that
```

```
start = datetime.now()
asm_image()
end_time = datetime.now() - start
print(end_time)
```

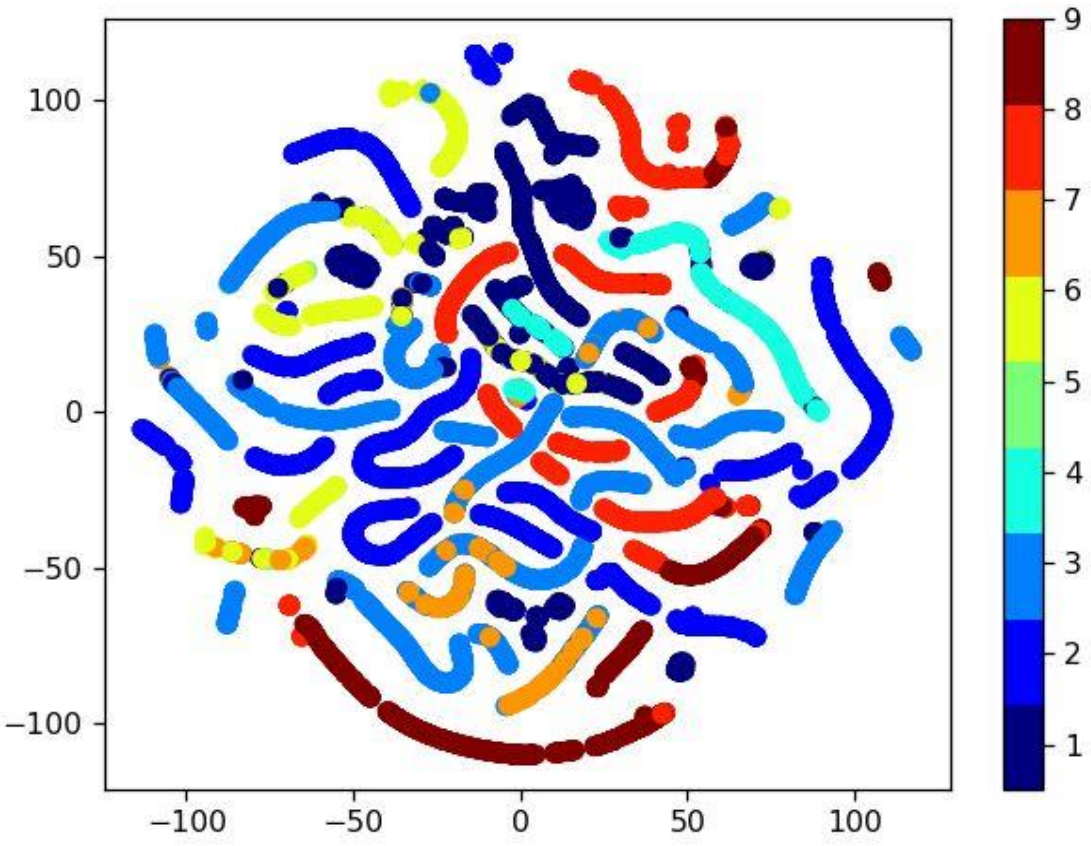
```
2:34:00.389140
```

Example of one of the image formed by ASM Files (Red box indicates the top 800 pixels)



9) And performed all the data processing on top of it.

10) Performed t-SNE as a dimensionality reduction process in order to visualize 800 dimension obtained in 2D.

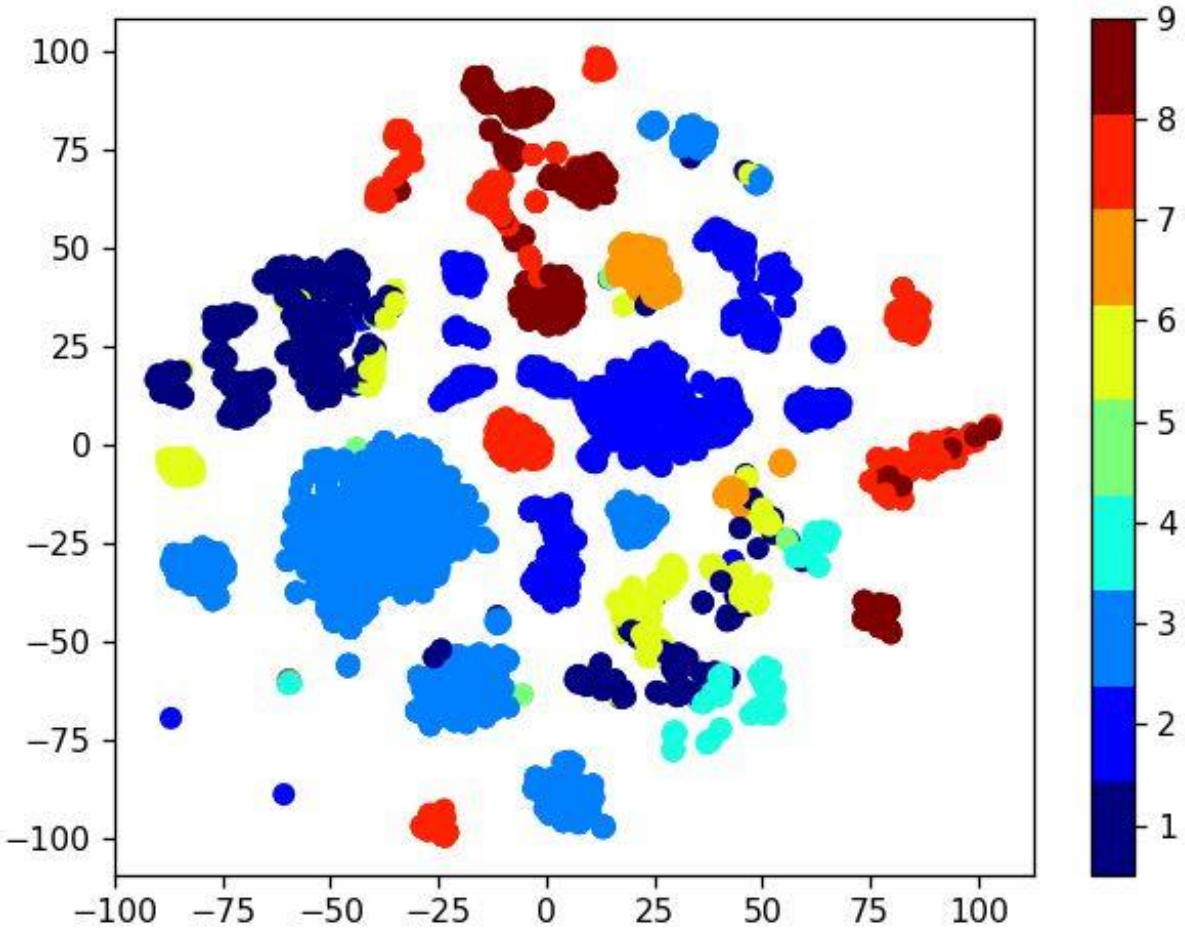


It’s really interesting because using these feature clustering is not maintained properly but it makes sure that no classes intersect each other much.

So after this we conclude that if we combine these features then clusters formation and intersect restriction can be done very well.

So next I combined the features of Bytes and ASM Files and perform modelling on top of it.

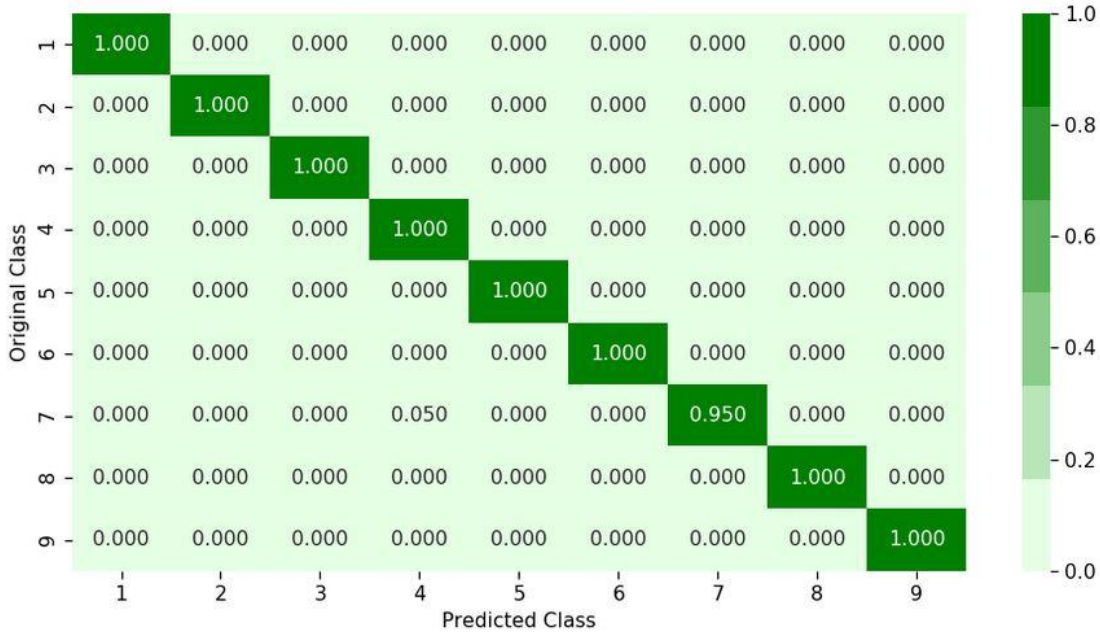
11) Performing the t-SNE and visualizing the features obtained by combination of ASM and Bytes File.

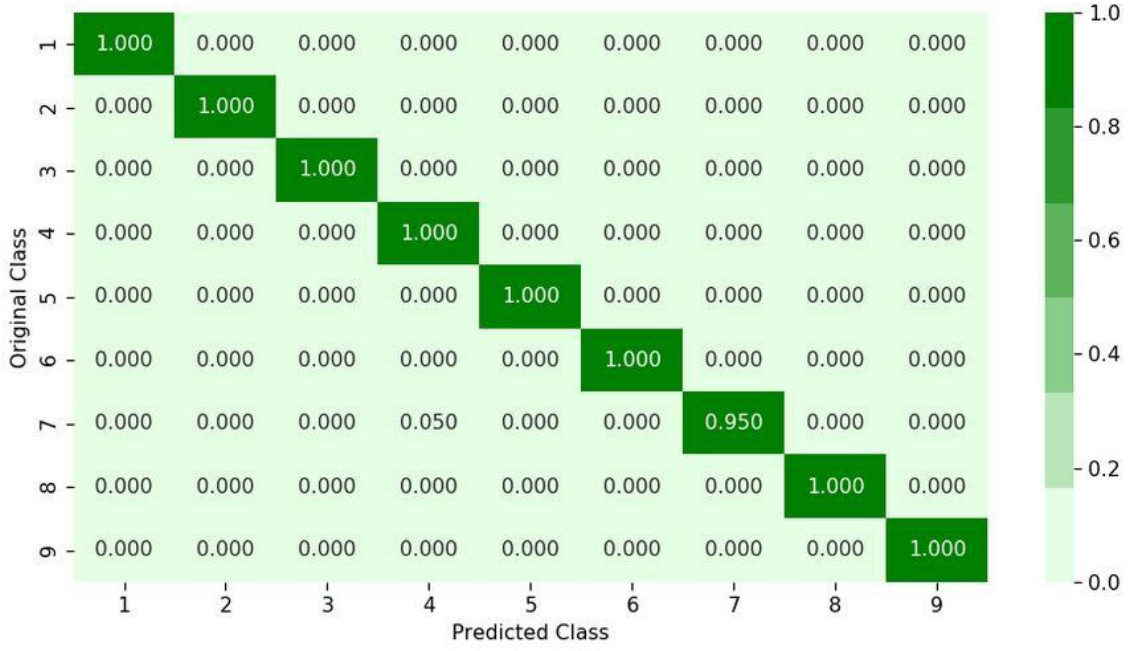


Features are really impressive as we can see each and every class are clearly separated.

After this we perform modelling on all final dataset.

12) Confusion matrix of Logistic regression, Random Forest and XGBoost are shown respectively





Here as we can see XGBoost perform impressively with following dataset.

And log loss achieved here was equal to 0.021 which is best among all the models

Observation and Results

1) I got validation 0.022 and also I tried to get 0.01 but I was not able I think because I used only 50% points because of limitated computation, I already informed about it.

2) I got 0.08 validation loss only with bigram bytes' file

3) TSNE with bigram bytes' file was very good, it showed that features are very effective

4) Then I used the .asm file and then converted them to images and extracted top 800 pixels from it and made data frame from it

5) TSNE with only .asm was also very good and each clusters were visible separately

6) Merged the both .asm data frame and. byte files data frame and also used the given unigram features

7) TSNE plot with the merged file was very distinguishable.

8) Log was least with merged dataset with XGBoost, it was 0.0223 with 50% dataset, could have been achieved more, if used more data points

FUTURE ENHANCEMENT

1) As I have used 50% data due to lack of computation power , if trained with 100% data, Performance metrics can be improved and prediction can be made more accurate

2) In text processing, bigram bag of words in being implemented, some non-trivial techniques like tfidf weighted word-to-vec , average tfidf could be implemented to reduce the loss further

Reference

1) Review Research paper - https://arxiv.org/pdf/1802.10135.pdf
2) XGBoost - https://arxiv.org/pdf/1603.02754.pdf
3) Malicious Software - https://www.zdnet.com/article/what-is-malware-everything-you-need-to-know-about-viruses-trojans-and-malicious-software/
4) K-Nearest-Neighbor - https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbors-algorithm-clustering/
5) t-SNE - https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1
6) Text-Processing - https://machinelearningmastery.com/gentle-introduction-bag-words-model/
7) Random Forest - https://medium.com/opex-analytics/opex-101-random-forest-6a8e4bd4626c