**Malware detection using Machine Learning**

Murtuza Shareef (G01024452)

Deepak Kanuri (G01070295)

**Description:**

**Introduction:**

Lately, the malware business has turned into an efficient market including a lot of cash. Very much supported, multi-player syndicates put intensely in advances and abilities worked to sidestep conventional security, requiring against malware sellers to grow counter systems for finding and deactivating them. Meanwhile, they exact genuine monetary and enthusiastic agony to clients of PC frameworks.

One of the significant difficulties that enemy of malware faces today is the tremendous measures of information and records which should be assessed for potential vindictive expectation. For instance, Microsoft's ongoing identification against malware items are available on over 160M PCs worldwide and examine over 700M PCs month to month. This produces a huge number of every day information focuses to be broke down as potential malware.

One of the fundamental explanations behind these high volumes of various documents is the way that, **to avoid identification**, **malware creators** **acquaint polymorphism with the malevolent segments**. This implies pernicious records having a place with the equivalent malware "family", with similar types of malevolent conduct, are continually changed as well as muddled utilizing different strategies, to such an extent that they look like a wide range of documents.

So, **the idea is** to get Machine Learning involved – to get the predictions depending on the contents of the file.

The project is on Kaggle. It is a Microsoft malware classification challenge of 2015.

The **training and test data**, of each malware file has an **ID**, a 20-char hash value uniquely identifying the file, and a Class, an integer representing one of 9 family names to which the malware may belong. Our task is to classify the test data into 9 malware classes which are:

1. Ramnit 2) Lolipop 3) Kelihos\_ver3 4) Vundo 5) Simda

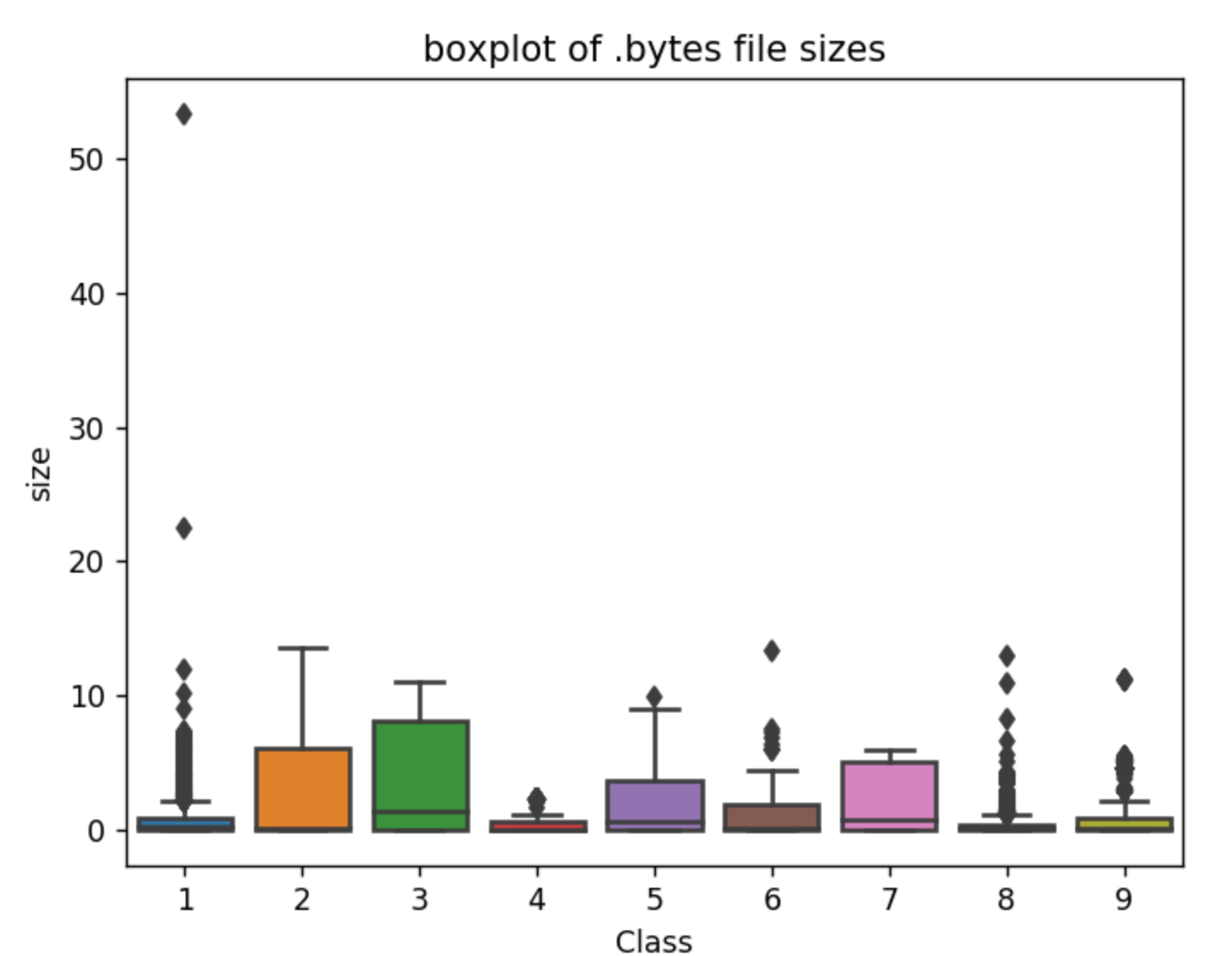
6) Tracur 7) kelihos\_ver1 8) Obfuscator.ACY 9) GATAK

**Methods & Results:**

1. **Feature extraction**
   1. **File size:**

To classify the files into appropriate class, - the **file size** was identified as one of the useful features.

The box-plot of the distribution of the file size over the classes is:



We can see that the size of the files – was able to achieve **clean separation among at least some of the classes** – and hence can contribute to the classification.

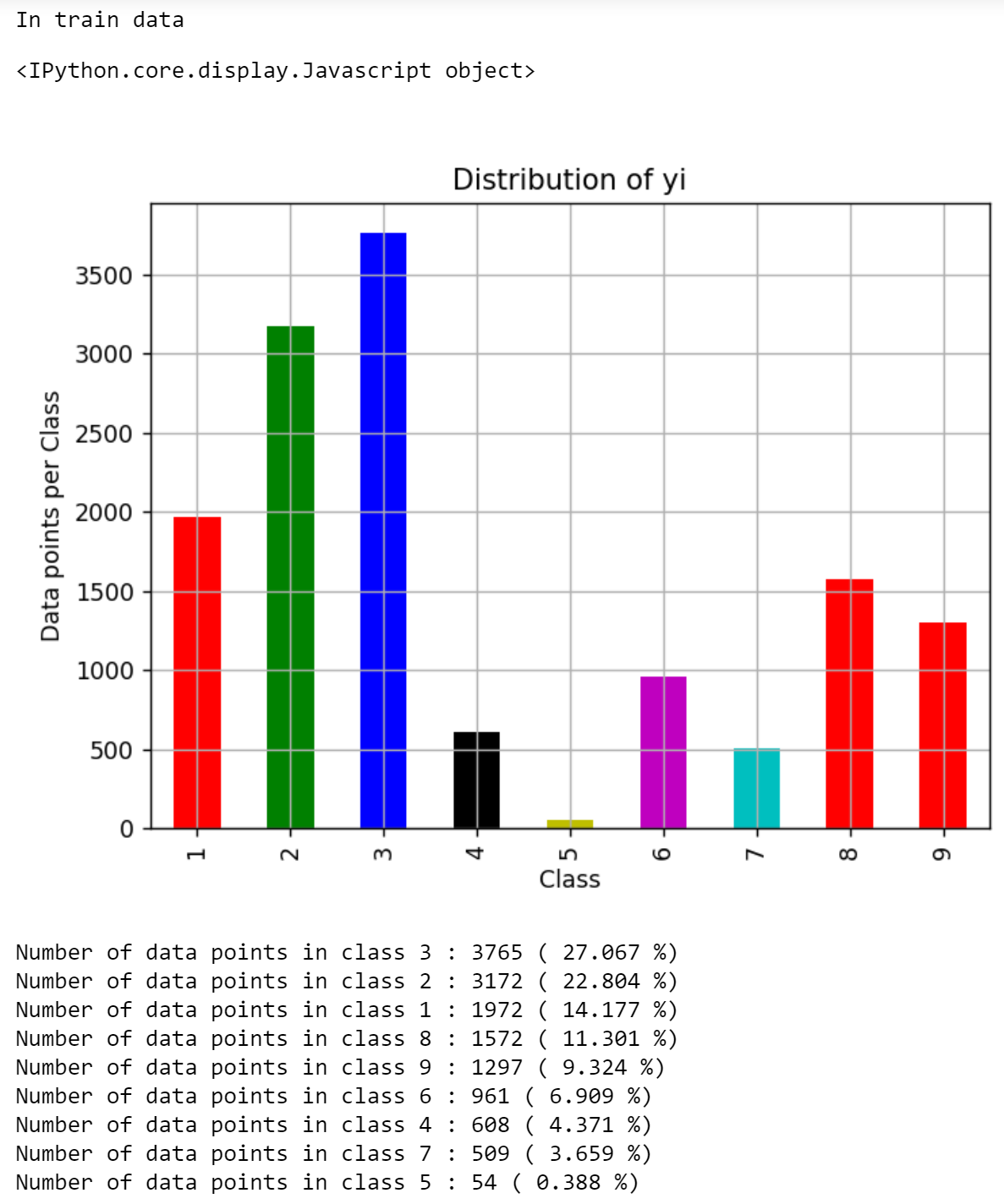
**1.2 Count vectorizer:**

We have used count vectorizer to get a running count of the number of all the possible hexa-decimal values in each file.

But, the count vectorizer from sklearn had a limitation – that it would need all the data to fit in RAM to be able to run which, **given the scale of data** **200 GB** was not possible – so, we had to **implement the count vectorizer** and to count of all the hexa-decimal characters in each file and **normalized it**.

# **Train Test split**

We split the data into training and testing data and the crossvalidation data, we split the data by **stratifying the class-Y**: (to ensure equal distribution of y classes in all splits)



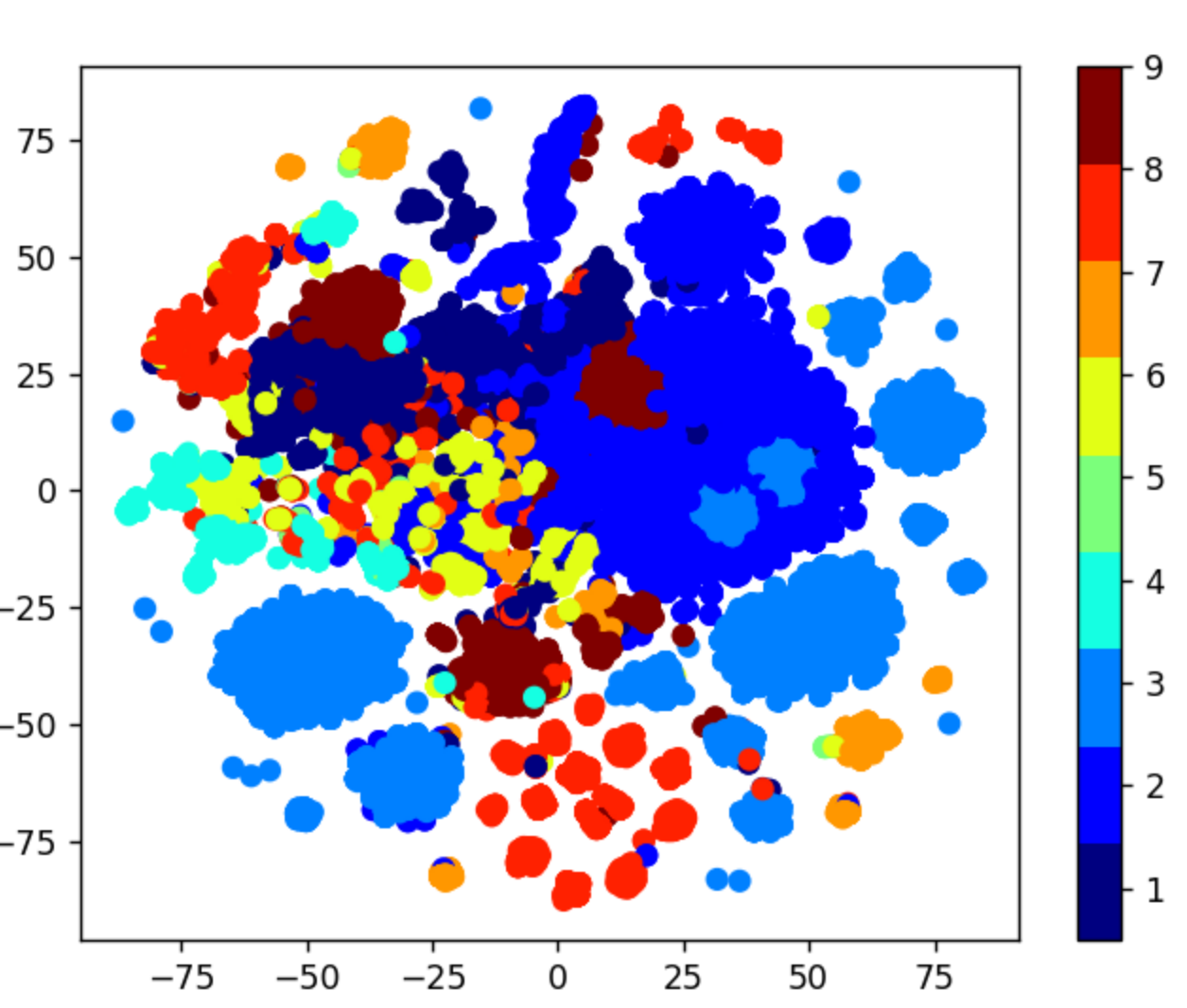
The distribution for cross validation and test data is similar.

**The final split on the train ; cv and test data is 64 – 16 – 20 (in percentages)**

### **EDA - Multivariate Analysis**

We now see, if the features selected – can meaningfully separate the data classes.

The TSNE plot of the file features (file size + output of count vectorizer) using perplexity=50 is: (different values of perplexity were also tried with similar results)



We can see that the data is **reasonably well-separated**, and we could go ahead with building the ML models.

1. **Metrics:**

To evaluate our classification we used two metrics:

- **Multi-class log loss**

**- Confusion Matrix**

1. **Machine learning:**

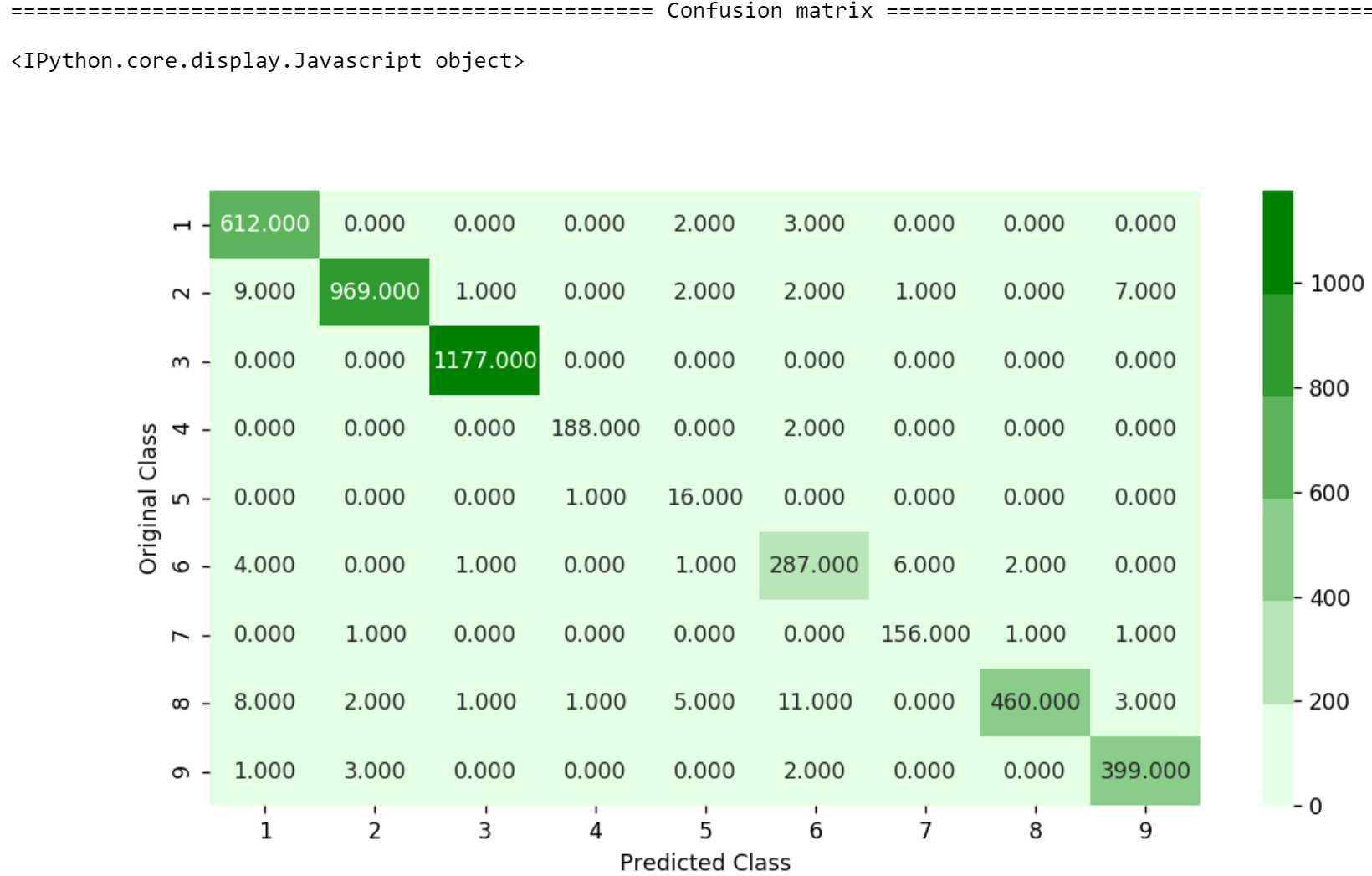
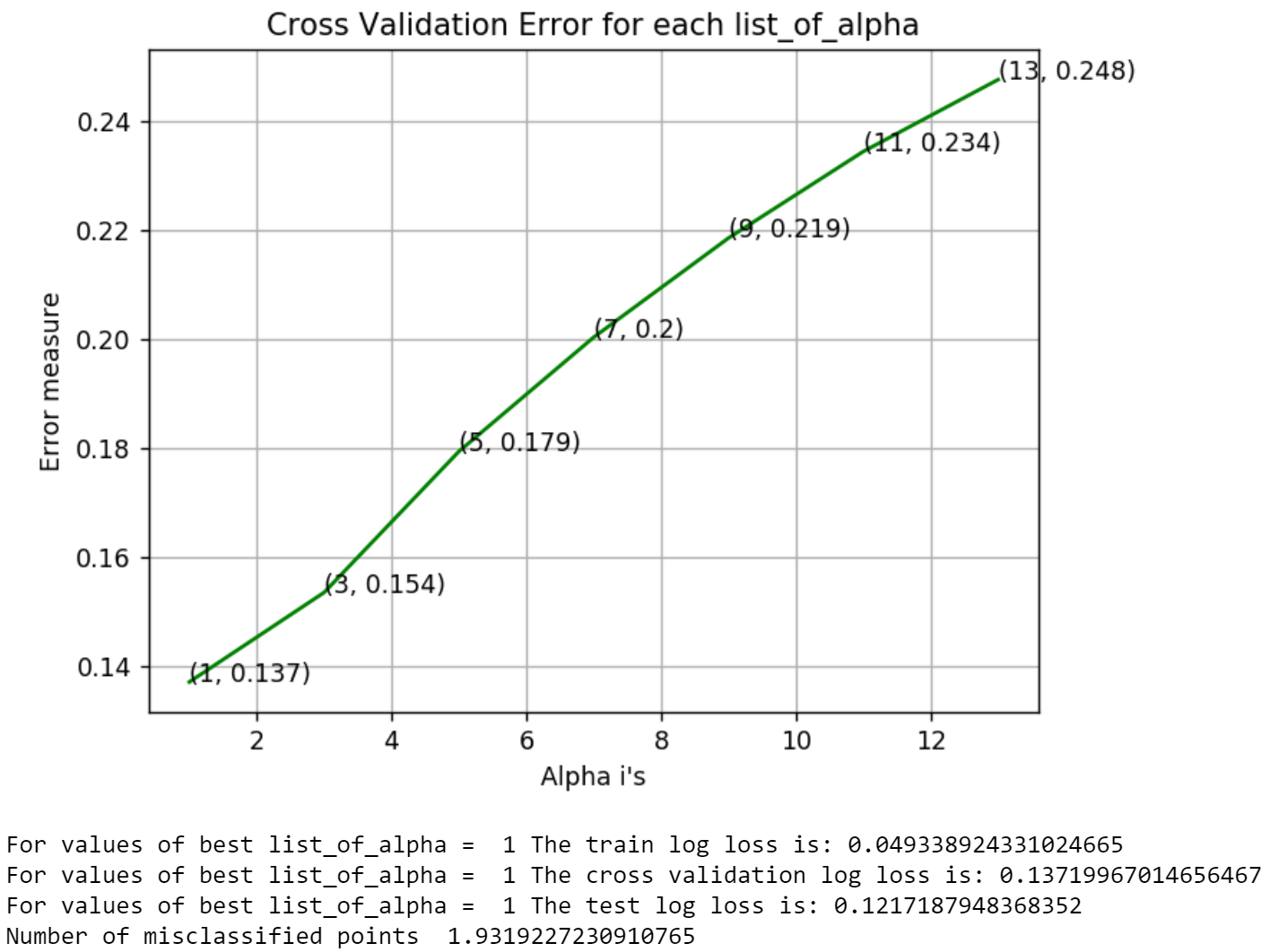
Different classification algorithms have been implemented –

**Random model – for coming up with the base line**

**Logistic Regression (dropped due to its inability to classify – Malware class 5)**

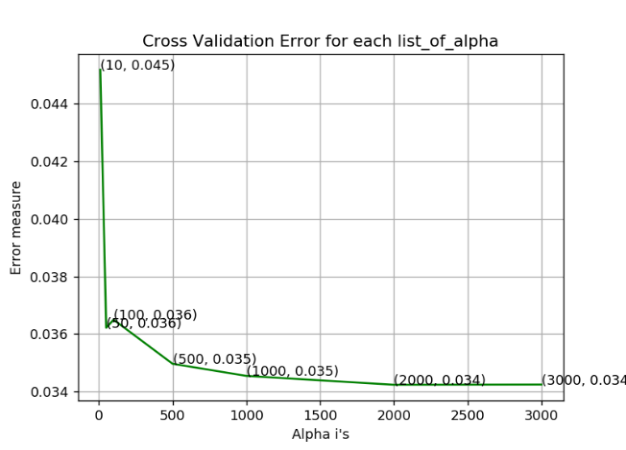
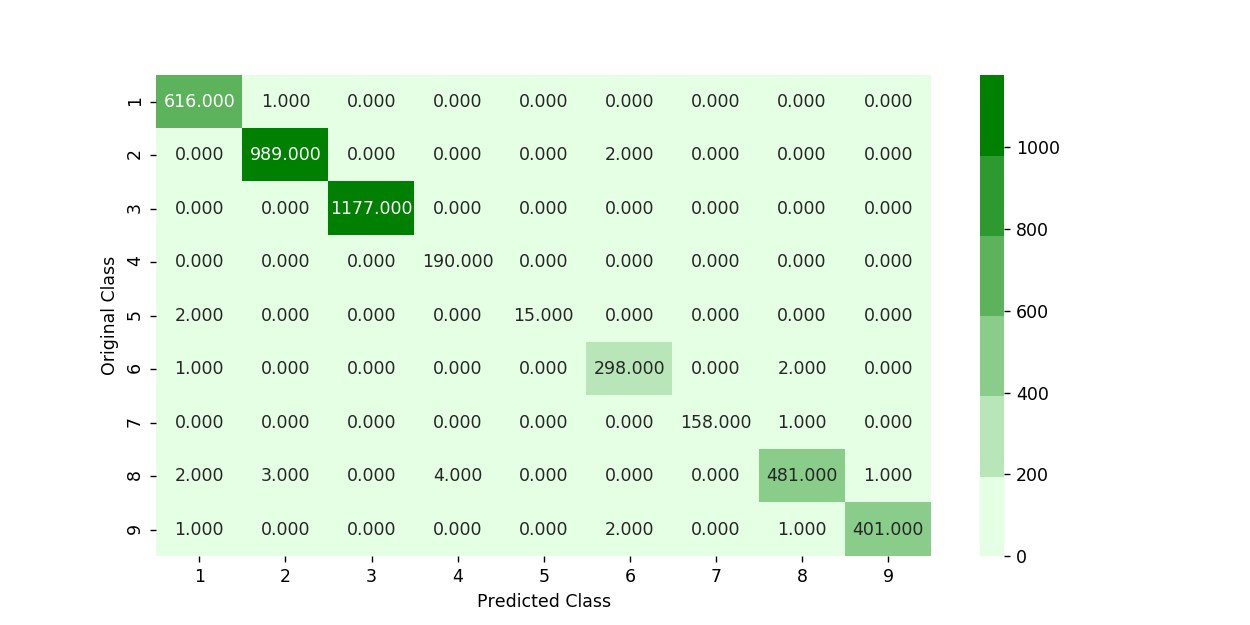
### **K Nearest Neighbor Classification**

After hyper parameter tuning the CV error, log-loss and the confusion matrix -



### **Random Forest Classifier**

After hyper parameter tuning the CV error, log-loss and the confusion matrix -

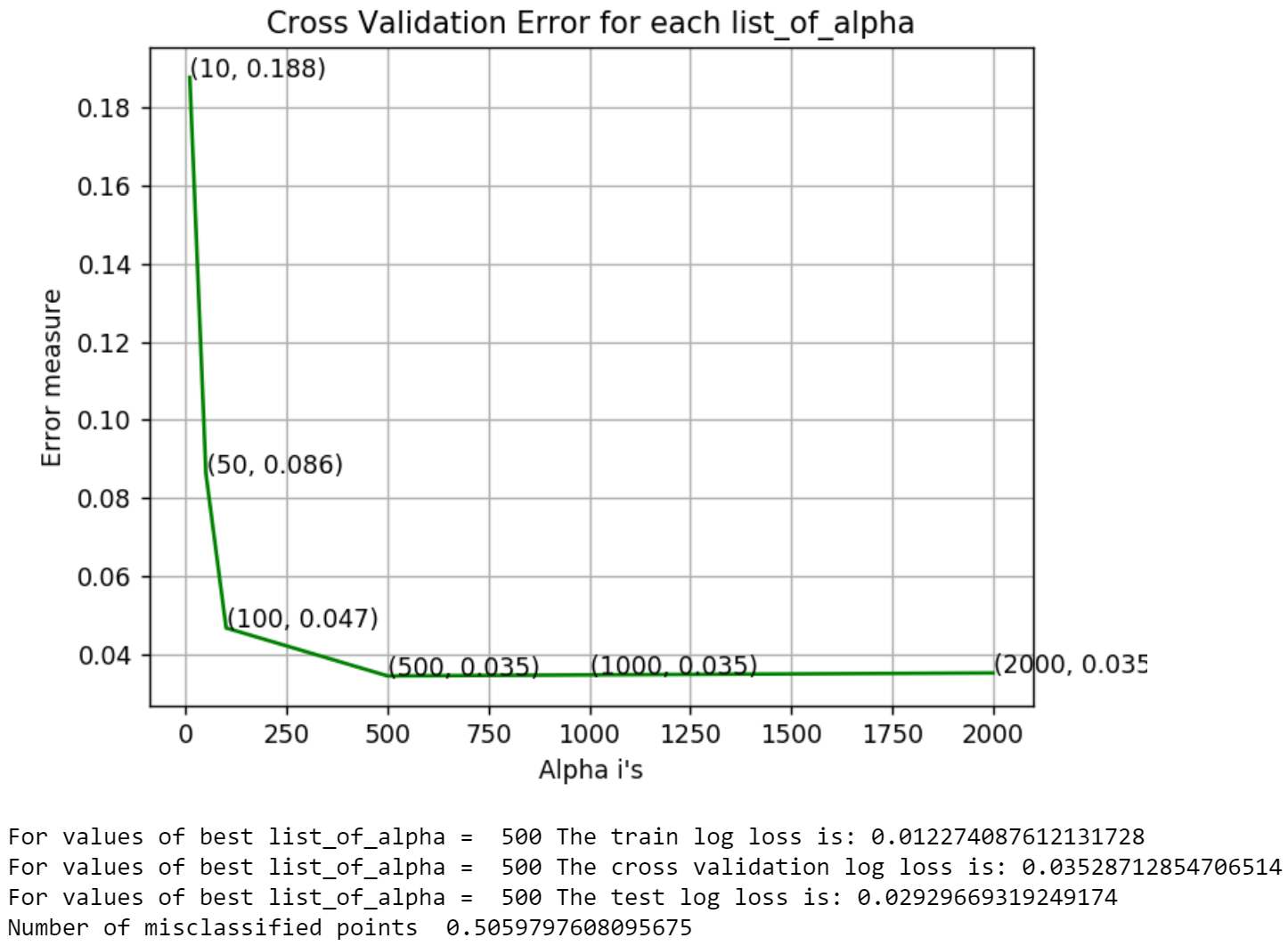
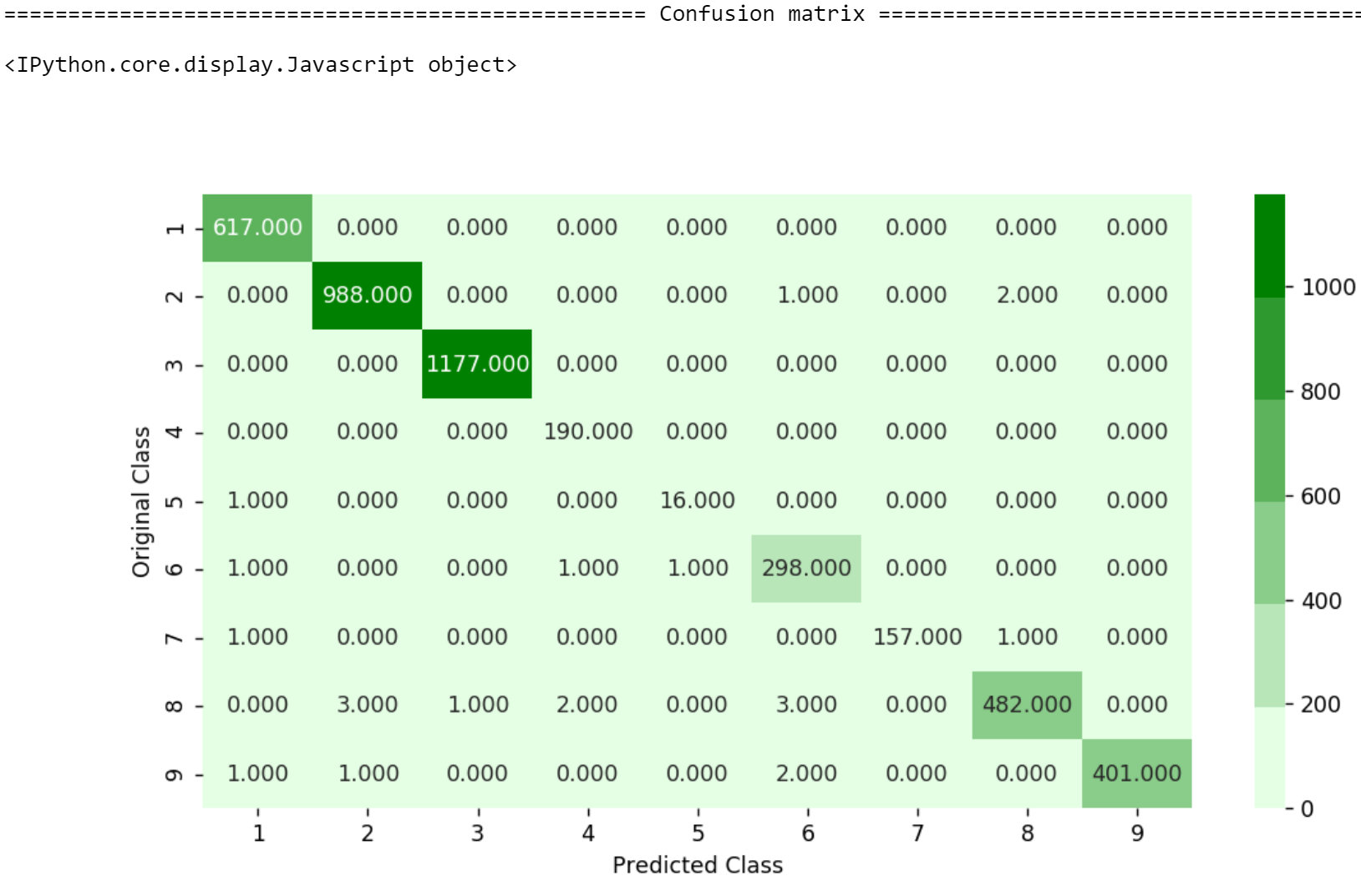
For values of best list\_of\_alpha = 2000 The train log loss is: 0.01350791518204678

For values of best list\_of\_alpha = 2000 The cross validation log loss is: 0.0342358931760652

For values of best list\_of\_alpha = 2000 The test log loss is: 0.030827729253342366

### **XGBoost Classification**

Again, hyper parameter tuning and running the model with tuned params (using Random search CV – the cross validation error, log-loss and the confusion matrix –

1. **Conclusion:**

**It covered:**

After applying K-Nearest Neighbour, Logistic Regression, Random Forest and XGBoost Classifiers against the baseline Random model, we have concluded that

* XGBoost produced the **lowest log-loss** among all the used classifiers.
* **Confusion matrix, precision matrix and recall matrix** for XGBoost further strengthened its case when compared to other classifiers.

Therefore, XGBoost can be used to reliably classify a file into one of the malware classes.

**Future improvements:**

* The featurization can be improved on – more sophisticated features could be used to improve the performance of the model.
* More complex models – like ensembles can be tried out
* Probably deep learning can be applied – if we are treating it as a text classification problem as we already have huge amount of data available.