XGBOOST-BASED DETECTION OF MALWARE FAMILIES IN THE DREBIN DATASET

Michela Ricciardi Celsi (1580884)

1. INTRODUCTION

This work is aimed at training and validating a supervised learning algorithm for *classifying malware families*, that is, for detecting which malware family a given malware sample belongs to, based on the *DREBIN dataset* and relying on *Extreme Gradient Boosting*.

The DREBIN dataset [1, 2], focused on Android software only, consists of 123,453 benign applications and 5,560 malware samples. In particular, the *features* (S1, …, S8) characterizing the 5560 samples of malicious code are extracted from the related *manifest.xml* file (S1 requested hardware components, S2 requested permissions, S3 app components, S4 filtered intents) and from the disassembled code (S5 restricted API calls, S6 used permissions, S7 suspicious API calls, S8 network addresses). Such features, once extracted, have been stored into a text file whose name is recorded as the SHA-1 hash of the corresponding Android package. The dataset is also accompanied by a dictionary file in .csv format associating each malware sample in the dataset with the family it belongs to.

I chose to use XGBoost – Extreme Gradient Boosting [3, 4], a scalable, portable and accurate library implementing boosted trees – to train a multi-class classifier in Python that classifies the malware samples in the DREBIN dataset among the different malware families.

2. DATA PREPARATION

For the specific task of malware family detection (i.e., a multi-class classification problem), only the dataset samples accounting for malicious code are needed. So, first of all, I extracted such samples from the dictionary file. In this respect, since the different labels identifying each single malware family are provided in the form of categorical variables, one-hot encoding is performed, defining a binary variable for each distinct malware family label.

Then, I considered the following three different datasets for training the XGBoost classification algorithm: 15S, containing the malware samples belonging to families with more than 15 samples; 30S, containing the malware samples belonging to families with more than 30 samples; 40S, containing the malware samples belonging to families with more than 40 samples – this is the same as the one used by the authors of Drebin.

3. GRADIENT BOOSTING

Since a single tree is usually not strong enough to be used in practice, Gradient Boosting has been designed to rely on a tree ensemble model, which follows the CART (Classification And Regression Trees) approach: in brief, it adds one classifier at a time, so that the next classifier is trained to improve the previously trained ensemble. Eventually, the prediction scores of each individual tree are summed

up to get the final score. Classifiers are added sequentially until no further improvement can be made. Boosting reduces error mainly by reducing bias.

More precisely, by resorting to a least-square regression setting for the sake of clarity, the goal of Gradient Boosting is to teach a model to predict values of the form $\hat{y} = F(x)$ by minimizing the mean squared error $\frac{1}{n}\sum_i(\hat{y}_i - y_i)^2$, where i indexes over some training set of size of actual values of the output variable. At each stage of GB, the current model F_m may be assumed to still be imperfect. Hence, Gradient Boosting improves on F_m by constructing a new model that adds an estimator h in order to provide a better model, $F_{m+1}(x) = F_m(x) + h(x)$. To find , the gradient boosting solution starts with the observation that a perfect h would imply $F_{m+1}(x) = F_m(x) + h(x) = y$. Therefore, gradient boosting will fit h to the residual $y - F_m(x)$. As in other boosting variants, each F_{m+1} attempts to correct the errors of its predecessor F_m. For further detail, refer to [5].

In brief, this is the rationale behind Gradient Boosting. Extreme Gradient Boosting, in turn, extends this approach by performing optimized distributed Gradient Boosting to perform classification and regression tasks on large datasets in a faster and more accurate way.

4. ALGORITHM TUNING AND PERFORMANCE

The objective function chosen for the algorithm is the *softmax* one, which is very common in multi-class classification. In this respect, I tested two distinct evaluation metrics (see Figs. 1-6):

1. Classification error rate: $\mathcal{E} \coloneqq \frac{FP+FN}{n}$, where FP and FN are the false positives and false negatives, respectively.
2. Multi-class cross-entropy loss: $-\frac{1}{n}\log P(data|model) = -\frac{1}{n}\sum_i k_i \, log \, y_i$, where $n = k_1 + \cdots + k_r$ and $y_1, \ldots, y_r$ are the hypothetical occurrence probabilities of the r classes (or malware families).

I have tuned the algorithm parameters by relying on 10-fold stratified cross-validation. Stratification was adopted in order to make sure that each fold stood as a good representative of the whole dataset: this way, the dataset is sufficiently balanced. In particular, I properly tuned the following parameters:

- learning rate for training;
- *max_depth*: maximum depth of the tree, implying greater complexity as it grows;
- *min_child_weight*: minimum sum of instance weight needed in a child; the larger the parameter value, the more conservative the algorithm will be;
- *subsample*: subsample ratio of the training instance; if I set this parameter to 0.5, XGBoost will randomly collect half of the data instances to grow trees and this will prevent the classification algorithm from overfitting the dataset;
- *colsample_bytree*: subsample ratio of columns when constructing each tree;
- *gamma*: minimum loss reduction required to make a further partition of a leaf node of the tree; the larger the parameter value, the more conservative the algorithm will be.

This procedure for algorithm tuning has been repeated for each of the three datasets. In particular, in the considered case, suitable values of *max_depth* and *min_child_weight* are, respectively, 8 and 1 in order to prevent the algorithm from overfitting the dataset.

The simulations have been carried out in Python 3 on a 2,5 GHz Intel Core i5 equipped with 4 GBs of RAM.

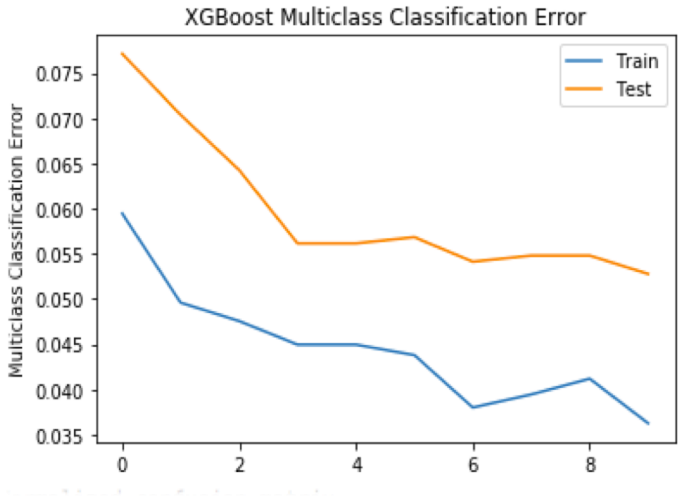

Figure 1: XGBoost multi-class classification error rate evaluated with respect to the 15S dataset.

Figure 2: XGBoost multi-class cross-entropy loss evaluated with respect to the 15S dataset.

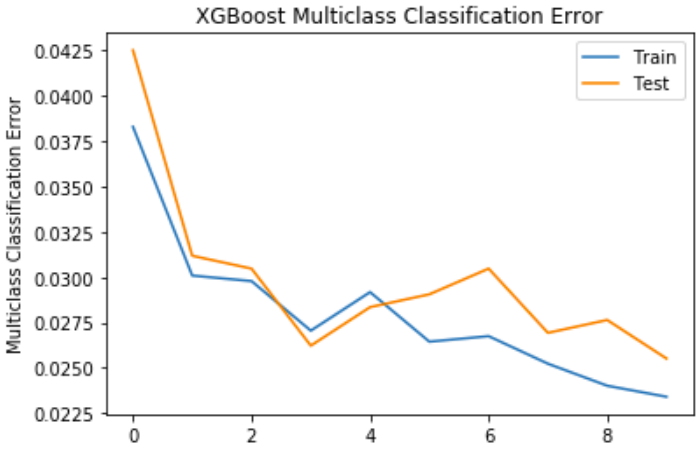

Figure 1: XGBoost multi-class classification error rate evaluated with respect to the 30S dataset.

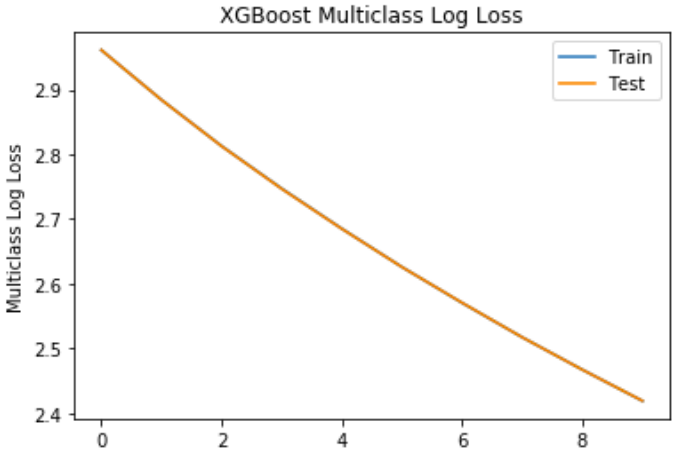

Figure 2: XGBoost multi-class cross-entropy loss evaluated with respect to the 30S dataset.

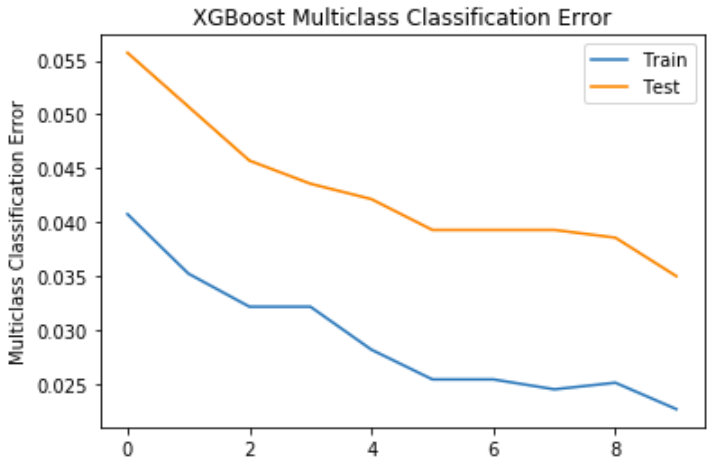

Figure 5: XGBoost multi-class classification error rate evaluated with respect to the 40S dataset.

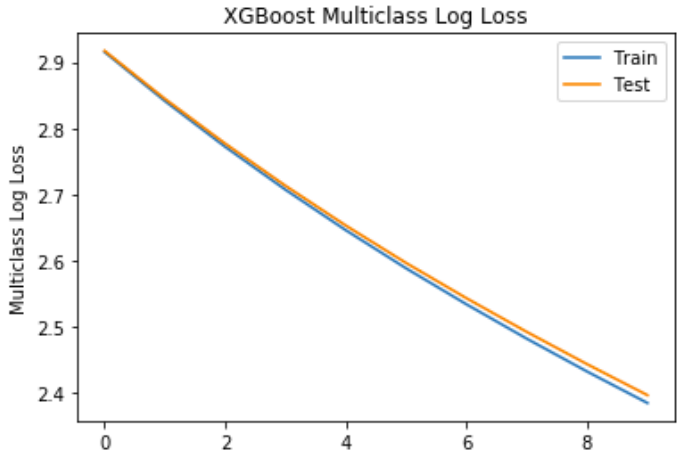

Figure 6: XGBoost multi-class cross-entropy loss evaluated with respect to the 40S dataset.

With the tuned parameters, it turns out that, if I consider a dataset referring to families with more samples, a reduction in standard deviation and an increase in the average model accuracy are obtained.

Dataset	**Size**	**Families**	**Average accuracy**	**Accuracy std. dev.**
15S	4926×12039	32	98.66%	0.36%
30S	4701×10961	21	99.26%	0.33%
40S	4664×10675	20	99.17%	0.32%

Moreover, the confusion matrices are reported below (see Figs. 7-9). Such matrices were obtained by shuffling and splitting the dataset into 70% (training set) and 30% (test set) of each of the three considered datasets.

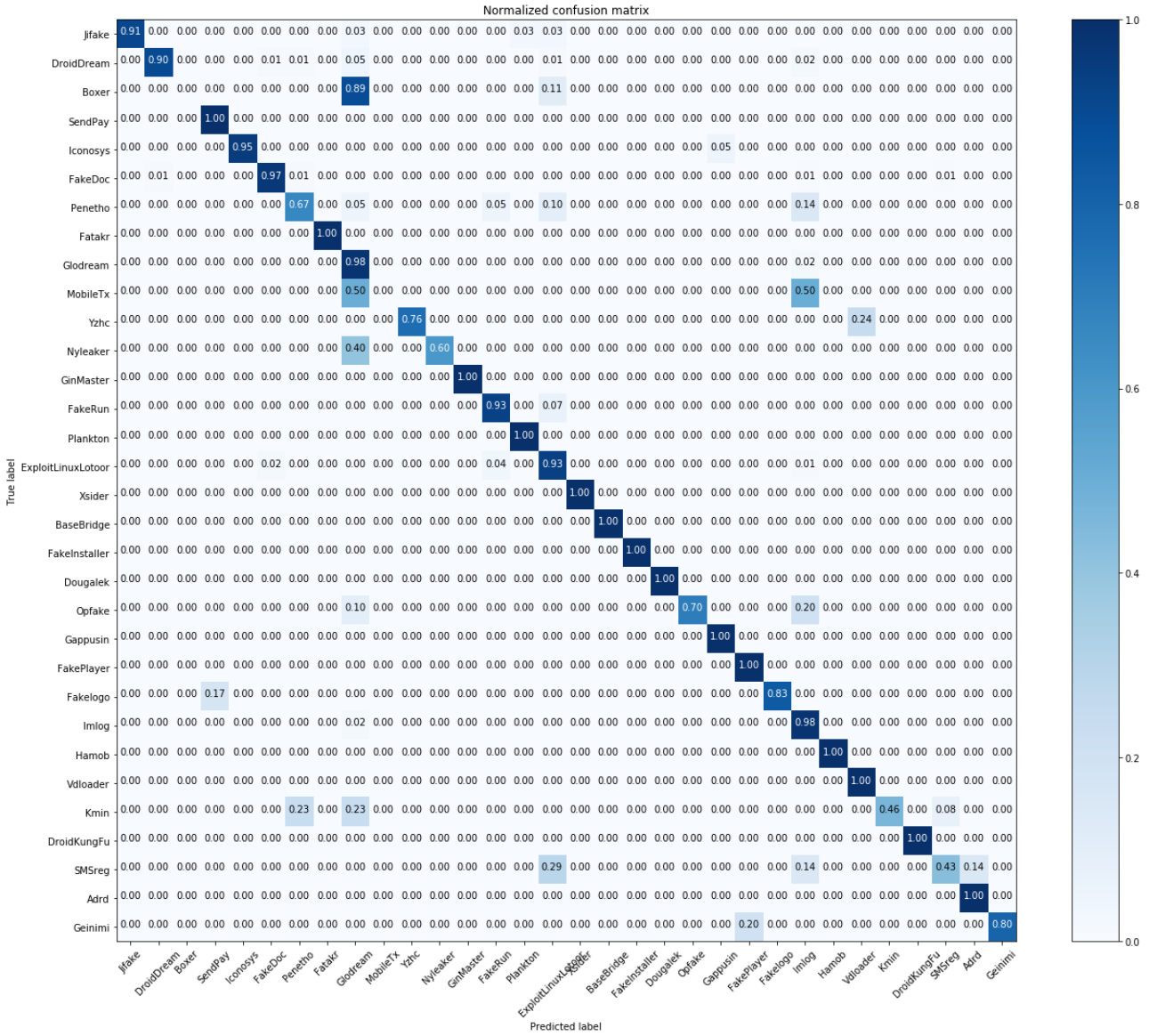

Figure 7: XGBoost confusion matrix evaluated with respect to the 15S dataset.

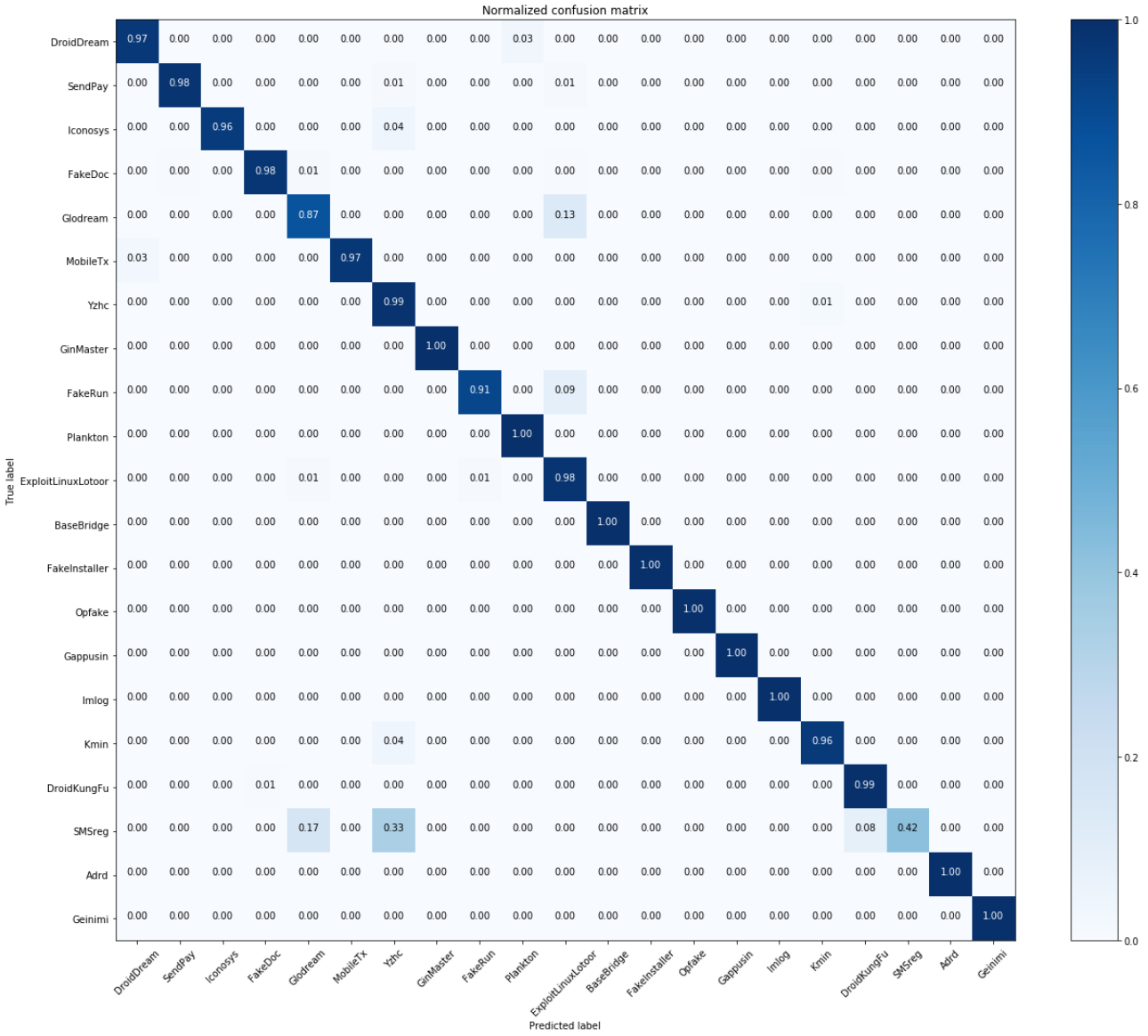

Figure 8: XGBoost confusion matrix evaluated with respect to the 30S dataset.

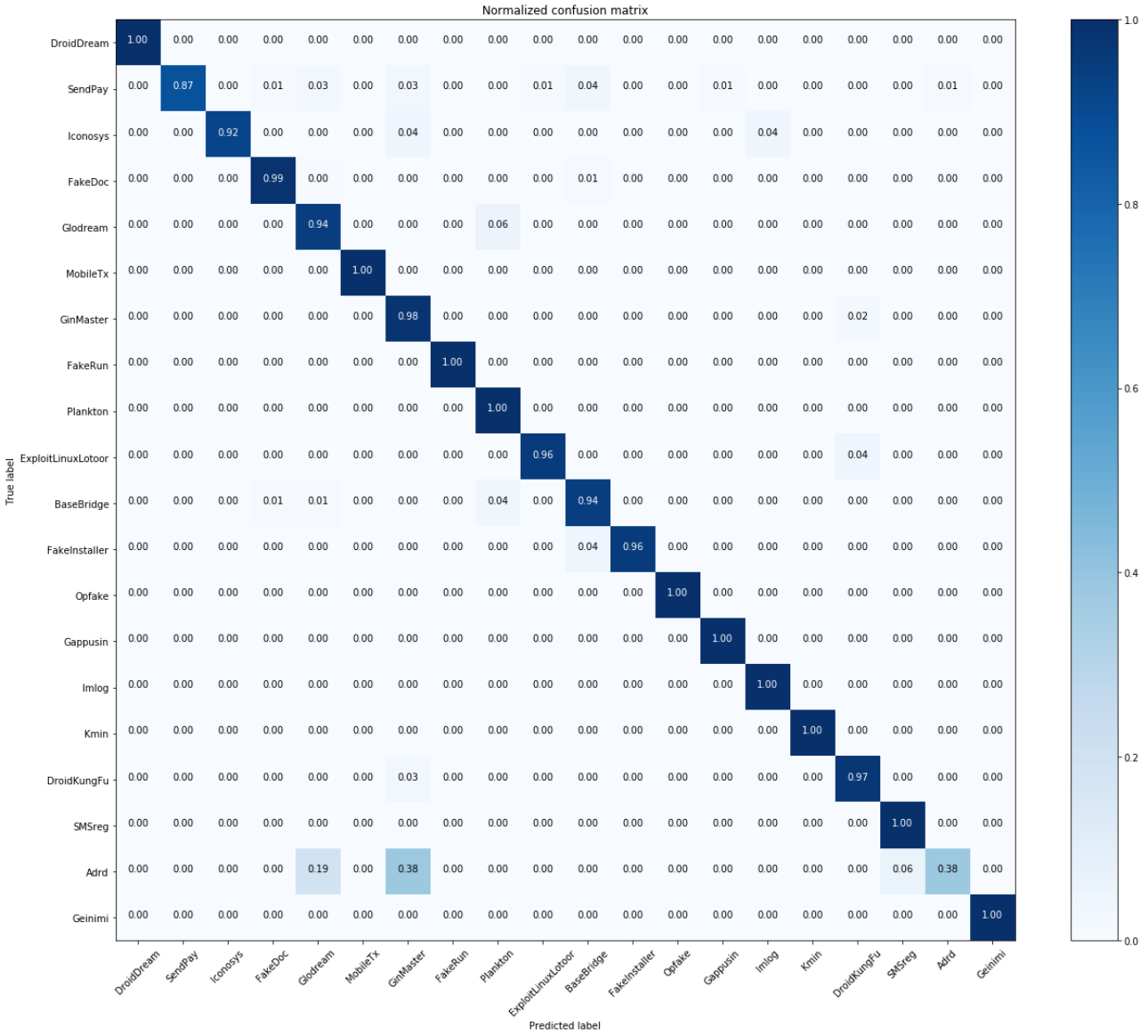

Figure 9: XGBoost confusion matrix evaluated with respect to the 40S dataset.

In particular, the more an attribute is used to make key decisions in decision trees, the higher its relative importance. Importance is calculated for a single decision tree by the amount that each attribute improves the performance measure, weighted by the number of observations the node is responsible for. The values of feature importance are then averaged across all of the decision trees within the model. As it can be seen from Figs. 13-15, the most important features are among the group of permissions and among the group of API calls. Instead, the URLs are the least relevant features due to the fact that they are very easy to hide. Indeed, Figs. 13-15 suggest possible ways to perform feature reduction/selection in order to simplify the malware family classification process: in other words, if we choose to remove the irrelevant features (like URLs), a good classifier should yield very similar performance as the above designed one. Another important aspect is related to the fact that the accuracy is bigger than 98.5%, which means that the classifier is working properly, and the very small values of the standard deviation confirm the goodness of the train model.

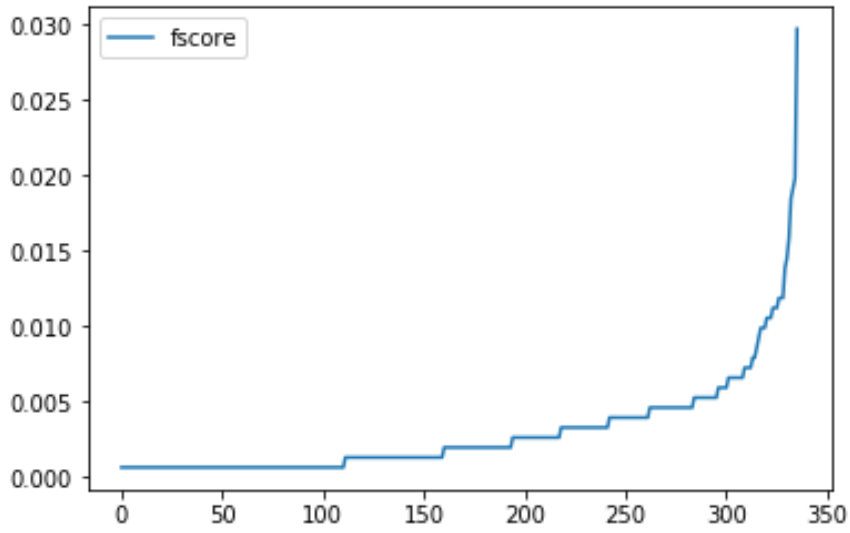

Figure 10: XGBoost F-score evaluated with respect to the 15S dataset.

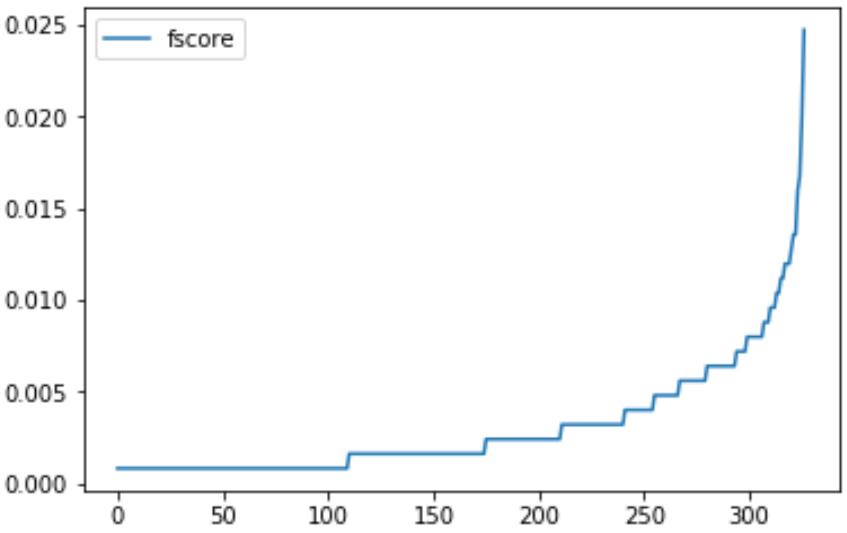

Figure 11: XGBoost F-score evaluated with respect to the 30S dataset.

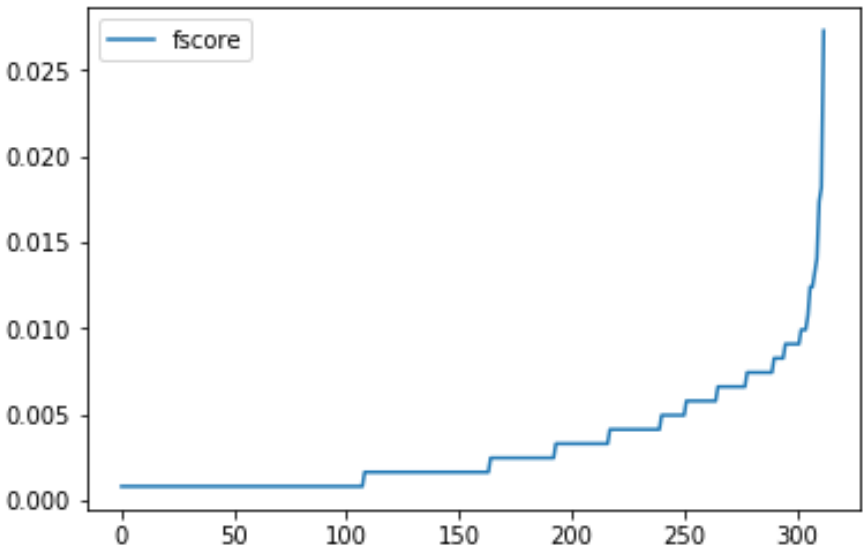

Figure 12: XGBoost F-score evaluated with respect to the 40S dataset.

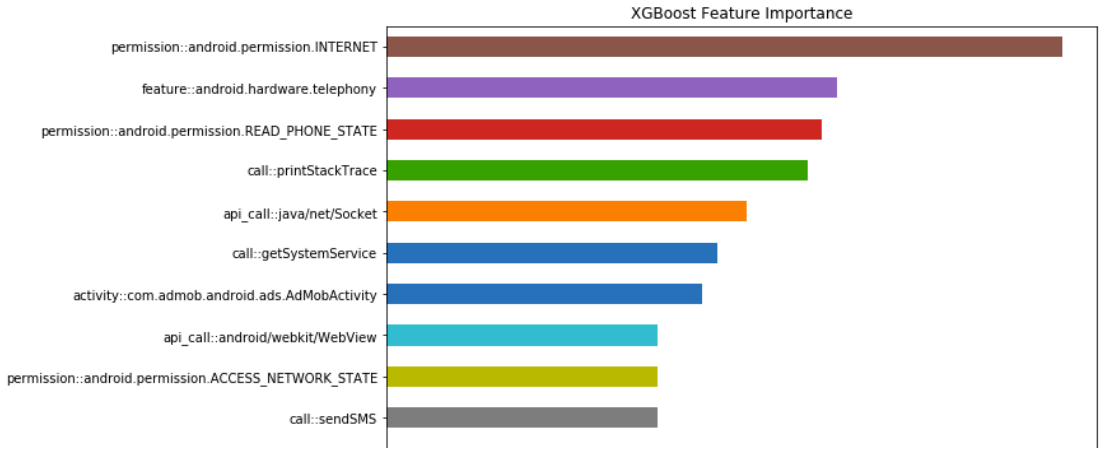

Figure 13: Head of the feature importance vector obtained through XGBoost when considering the 15S dataset.

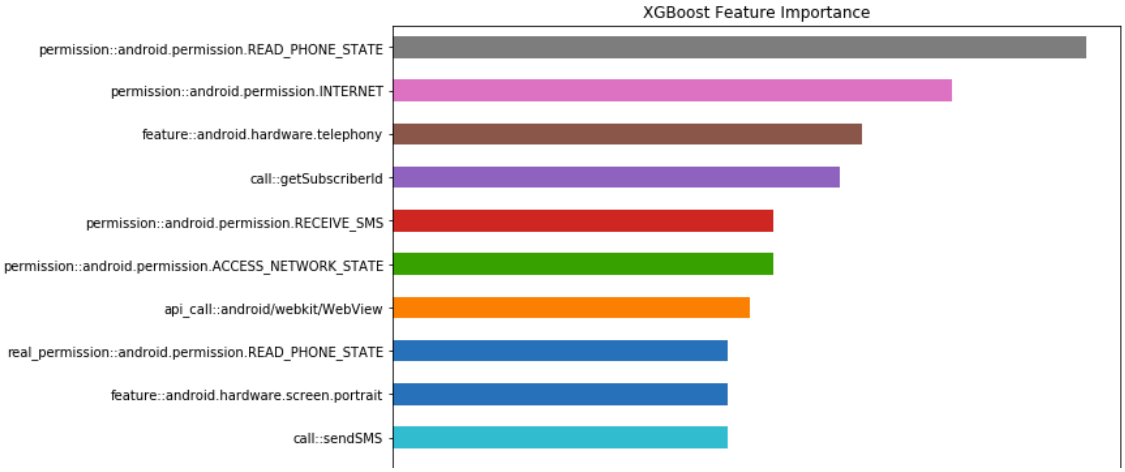

Figure 14: Head of the feature importance vector obtained through XGBoost when considering the 30S dataset.

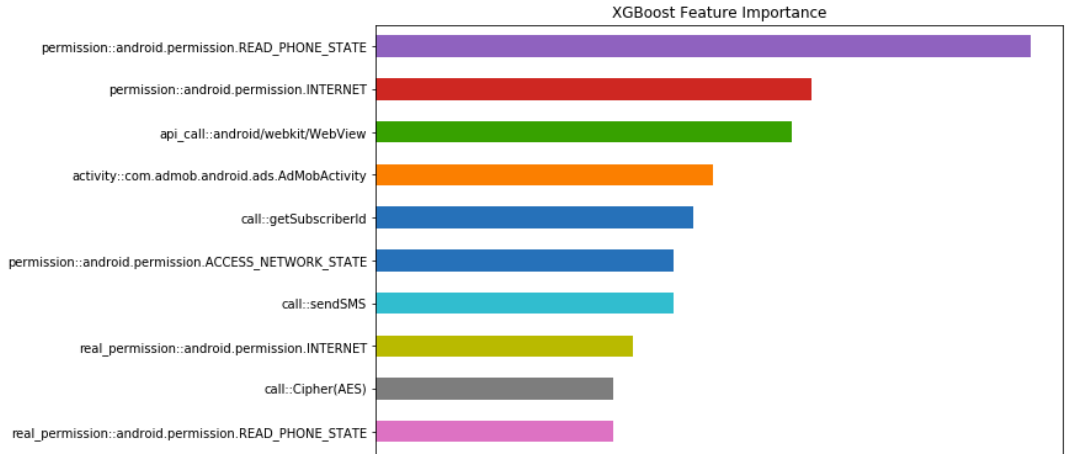

Figure 15: Head of the feature importance vector obtained through XGBoost when considering the 40S dataset.

5. REFERENCES

[1] DREBIN dataset, https://drive.google.com/file/d/0Bxxqx_AAp2u2enI0UzBqSEZQRHc/edit
[2] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket, *Symposium on Network and Distributed System Security (NDSS)*, 2014, 10.14722/ndss.2014.23247.
[3] XGBoost documentation, https://xgboost.readthedocs.io/en/latest/
[4] XGBoost library, https://github.com/dmlc/xgboost
[5] J. Friedman, Greedy Function Approximation: A Gradient Boosting Machine, *The Annals of Statistics, 29*(5), 1189-1232, 2001. Retrieved from http://www.jstor.org/stable/2699986