# The Effect of Sampling in the Machine Learning-Based Malware Analysis

**K. Sakshi Thimmaiah, Lakshmi S. Raj, Prasanthi Bolimera, and M. Anand Kumar**

## 1 Introduction

Android Operating System has become very popular over the years, and it is a Linux-based operating system. It has been designed primarily for touchscreen mobile devices and tablets. They are increasingly used to access services, such as messaging, video/music sharing and e-commerce transactions that have been previously available on PCs only. Subsequently, it has attracted several Malware developers who target these mobile users [3, 10].

We must detect this malicious software that tampers with the device performance and steals personal data, such as accessing contacts, media and personal messages, without the user's knowledge. Machine Learning techniques can be used to classify software into two categories: malware and safeware. This classification can be done by using the XML file called "Android Manifest" to present in each Android application. It provides essential information to the operating system, like the first class to use when starting the app or the type of permissions used in the application [3].

Only permissions provided in the file will be used in the application, and this is done only after asking the user to grant these permissions. If the application tries to use some other permissions which were not allowed in the Android Manifest file, the execution fails. Unfortunately, many users tend to grant permissions to unknown applications, which is why malicious software infects the device [3].

K. S. Thimmaiah (✉) · L. S. Raj · P. Bolimera · M. A. Kumar
Department of Information Technology, National Institute of Technology Karnataka, Surathkal, India
e-mail: ksakshithimmaiah.191it124@nitk.edu.in

L. S. Raj
e-mail: lakshmisraj.191it225@nitk.edu.in

P. Bolimera
e-mail: prasanthibolimera.191it240@nitk.edu.in

M. A. Kumar
e-mail: m_anandkumar@nitk.edu.in

B. Agarwal et al. (eds.), *Proceedings of International Conference on Intelligent Cyber-Physical Systems*, Algorithms for Intelligent Systems,
https://doi.org/10.1007/978-981-16-7136-4_12

Thus, users must be made aware of the type of software they are installing so that they do not fall prey to malicious software and lose essential data from their mobile devices. For this particular project, we are making use of the DREBIN dataset. It contains 5,560 applications from 179 different malware families. The samples have been collected from August 2010 to October 2012 [5].

## 2 Literature Survey

There are two approaches to detecting malware in Android operating systems. The first one is a signature-based approach which generates a signature for every kind of malware and compares it with the application [1]. Typical antivirus software (e.g., Norton and McAfee) use signature-based methods to identify malware. However, this can be easily evaded by attackers. Example methods involve changing signatures using code obfuscation or repackaging [11]. The second is behavioural detection. The behaviour of an application is compared at runtime to identify malicious intent [1, 3].

In recent years, there has been an increasing trend using machine learning to overcome the challenges mentioned above to develop automatic and intelligent malware detection methods. These techniques are capable of discovering certain patterns to detect previously unseen malware samples and identifying the malware families of malicious samples. These systems can be classified into two categories: Dynamic analysis and Static analysis [3, 11].

Dynamic analysis [12–14] involves accumulating information regarding API calls, environmental variables and data transmission during the execution of an application. Dynamic analysis gives precise predictions and has lower false positive rates [3, 11].

Static analysis involves two parts—feature extraction and classification. The features are first extracted from the source file and a model is created to identify the malware families. A number of known datasets are used for feature extraction. DroidMat is used for static analysis using the manifest file and source code to extract features and k-means clustering and k-NN classification [7]. DREBIN uses the manifest file to extract features from 5,560 applications and SVM as a classifier [3, 5, 11].

Some effort has been to integrate static and dynamic analyses for better performance. Dynamic analysis could be used to reduce false positives obtained after static analysis but doing so could in turn increase the false positive if a particular path is not executed during the dynamic analysis [3, 8, 9, 11].

Permissions accessed by Android Applications have been significantly studied to understand malicious intent. The Android operating system provides a coarse-grained mandatory access control (MAC). A permission-based classifier can identify more than 81% of malicious software. It can be used for preliminary malware check before a complete second analysis [3, 3].

These applications are classified as malicious or benign by the combination of permissions required by them. The DREBIN dataset contains 5,560 applications and the respective permissions from 179 different malware families were making it a sufficient dataset [3, 5].

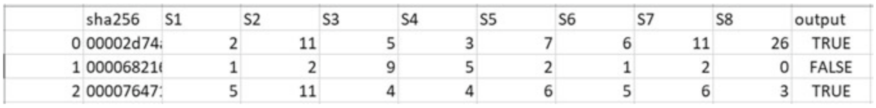
DREBIN database, in comparison with older datasets, gives a better False Positive Ratio parameter overall. Most Machine Learning algorithms provide high accuracy rates, which are more significant than 85% using the dataset. DREBIN performs better than older datasets and 9 out of 10 popular anti-virus scanners [2]. The analysis of the DREBIN dataset is speedy, usually taking lesser than a second on computer systems and lesser than a few seconds on a smartphone [3, 5].

On this dataset, Random Forest Classifier shows better results than Naive Bayes and Logistic Regression [2]. The Random Forest Classifier is most suited for high-dimensional data modelling. It is easy to use as it can handle all data types and manage dataset inconsistencies easily [4]. Support Vector Machine is a good choice for a classifier as it gives high precision and recall values. DREBIN data has embedded feature sets that make it suitable to run the SVM algorithm. Compared to the other two methods, the SVM algorithm takes longer to run but gives accurate results [6].

## 3 Dataset Description

The DREBIN approach made in this paper for malware detection and classification is to gather as many features as possible from the application's manifest and code and embed these features into a joint vector space where each feature is grouped into sets. Some machine learning techniques are used to identify patterns in these features, which were gathered earlier. These features, each collected from each application, have the following properties, which are further grouped into sets: feature as Set S1 (Hardware Components) permission as Set S2 (Requested Permission) activity, service receiver, provider, service as set S3 (App Components) intent as set S4 (Filtered Intents) api call as set S5 (Restricted API calls) real permission as set S6 (Used Permission) call as set S7 (Suspicious API Calls) url as set S8 (Network Addresses) [3].

Due to the large size of features, the actual contents have not been used. Some have different values running into thousands, and not many are the same across other application files. Therefore, building one hot encoder and exponential growth in the feature vectors has been used in the algorithms. The number of feature properties of each feature set has been counted and stored in the dataset. A feature vector of size eight was used where each feature has count values and the output being True (malware) or False (not malware). The input vector looks in the following way (Fig. 1):



| | sha256 | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | output |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00002d74 | 2 | 11 | 5 | 3 | 7 | 6 | 11 | 26 | TRUE |
| 1 | 00006821 | 1 | 2 | 9 | 5 | 2 | 1 | 2 | 0 | FALSE |
| 2 | 00007647 | 5 | 11 | 4 | 4 | 6 | 5 | 6 | 3 | TRUE |

**Fig. 1** A snapshot of the feature_vectors_dataċsv file

## 4 Proposed Work

Using the dataset, the machine learning techniques are used to classify the applications as malware or non-malware. As there is much advancement in the usage of Android apps, it becomes a need for us to detect the malicious behaviour of Android apps for users' security, privacy and safe usage. Our approach is detecting malware systems using machine learning techniques that classify the apps as malicious and benign and suggest a better detection method.

### 4.1 *Sampling Data*

A graph between the count of malware and safeware+malware is plotted (Fig. 2), and it can be seen that the number of malware are 5560 and safeware+malware are 129013. It can be observed that there is a huge difference between the values, and this implies that the data is imbalanced. To balance this data, we use the methods of upsampling and downsampling it. Since the malware is very less in number, it is made as a minority class, and safeware+malware are made as a majority class.

- Upsampling: It is the process of inserting zero-valued samples between original examples in order to increase the sampling rate. In this dataset, the minority class is upsampled using resample method of the scikit-learn library with the number of samples set to 123453 and a random state of 123. The number of malware now is 123453, and the number of malware+safeware is also the same (Fig. 3).
- Downsampling: It is the method of removing samples of a disproportionately low subset of the majority class examples, decreasing the sampling rate. In this dataset, the majority class is downsampled using the resample method with the number of samples set to the length of the minority class (i.e., 5560) and random state to 123. The number of malware now is 5560, and the number malware+safeware is also the same (Fig. 4).

For both upsampled and downsampled data, the data is preprocessed, standardised and split into 70% as training set and 30% as testing set.

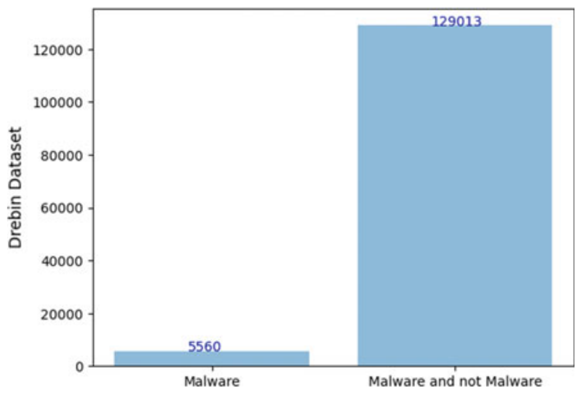


**Fig. 2** A graph between the count of malware and safeware+malware in the Drebin dataset

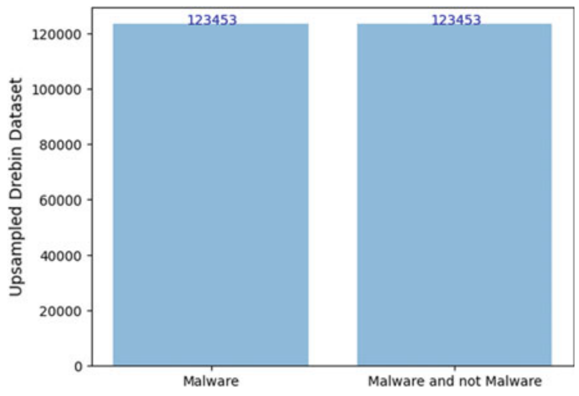


**Fig. 3** A graph between the count of malware and safeware+malware after upsampling the DREBIN dataset

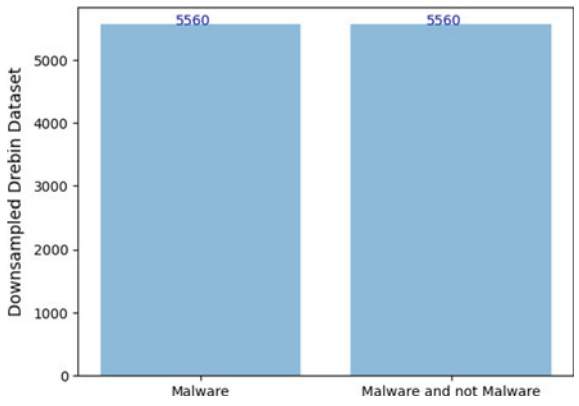


**Fig. 4** A graph between the count of malware and safeware+malware after downsampling the DREBIN dataset

### *4.2 Logistic Regression*

Logistic regression is a classification algorithm that is used to assign observations to a discrete set of classes. In this project, binary classification has been employed as we are classifying the software into two categories, viz. malware and safeware. The model, in which the data was split into training and testing sets, is trained with the logistic regression method from the linear model of scikit-learn. Then the labels of test data are predicted, and metric methods are used on these labels to calculate accuracy, precision, model recall and F1 score of the model. The same procedure was followed for unsampled, upsampled and downsampled data to calculate the results.

### *4.3 Random Forest Classifier*

A random forest classifier is a classifying method that combines many decision trees by recursively selecting subsets of datasets to build different decision trees. It does so by building multiple decision trees and then merging them to get a more accurate and stable prediction. The model is trained with the Random Forest Classifier method from the linear model of the scikit-learn. The test data labels are then predicted, and metric methods are used to calculate accuracy, precision, model recall and F1 score of the model. The same procedure is followed for unsampled, upsampled and downsampled data to calculate the results.

### *4.4 Support Vector Machines*

Support Vector Machine is a popular Supervised Learning algorithm used for classification. This algorithm aims to create the best decision boundary or line (hyperplane), which can segregate n-dimensional space into classes to make it easier for us to put the new data point in the correct category in the future. The algorithm involves choosing extreme points, called support vectors, in creating hyperplanes. The model is trained with the Support Vector Classifier (SVC) method from the SVM of scikit-learn. Prediction on the model and metric calculations on unsampled, upsampled and downsampled are similar to other classification methods.

## 5 Results and Analysis

The output plots, along with the results, are given below. We notice that all the three classifying methods have successfully classified the software in the given DREBIN dataset to Malware and Safeware, respectively.

(Formulae Used For calculations:)

$$TruePositives\% = \frac{TP}{P}$$

$$TrueNegatives\% = \frac{TN}{N}$$

$$Accuracy = \frac{TP + TN}{P + N}$$

$$Precision = \frac{TP}{TP + FP} = TruePositives\%$$

$$Recall = \frac{TP}{P}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

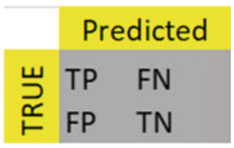
(TP = True Positives, TN = True Negatives, FN = False Negatives, FP = False Positives, P = Positives = TP+FN, N = Negatives = FP+TN)
Here, Positives are Malware and Negatives are Safeware.
Confusion matrix representation (Fig. 5):

Below are the confusion matrices of all three classications performed on unsampled, upsampled and downsampled data:

From the results given in Table 1, we can observe that the classification performed on the initial form of the dataset, which has not undergone resampling, produces inconsistent results. It is also clear that the three classifying methods perform better when the dataset has been upsampled or downsampled [3] (Fig. 6, 7, 8, 9, 10).

We can see that the Random Forest Classifier gives us the best results with an accuracy of 0.993 with the upsampled dataset, whereas with the downsampled dataset, it gives slightly lesser accuracy. It produced 99.7% true positives and 78.8% true negatives. The Logistic Regression method, on the other hand, gives a good accuracy of around 0.82. And on the other hand, the Support Vector Machine gives an accuracy of 0.866 when upsampled and 0.863 when downsampled, but these values are much lesser than the Random Forest Classifier method. Hence, we can conclude that the Random Forest Classifier method is a better method for malware classification among the three (Fig. 11, 12, 13, 14).

**Fig. 5** Representation of a confusion matrix



| | Predicted | |
|---|---|---|
| TRUE | TP | FN |
| | FP | TN |

**Table 1** Results of classifiers on unsampled, upsampled and downsampled data

| Classifier | Accuracy | Precision | Recall | F1 Score | TruePositives % | TrueNegatives % |
|---|---|---|---|---|---|---|
| Logistic regression (without sampling) | 0.961011 | 0.671532 | 0.216853 | 0.327839 | 99.51 | 21.68 |
| Logistic regression (upsampled) | 0.819756 | 0.852773 | 0.773277 | 0.811082 | 85.3 | 77.3 |
| Logistic regression (downsampled) | 0.821342 | 0.854838 | 0.78125 | 0.816389 | 86.28 | 78.12 |
| Random forest (without sampling) | 0.989561 | 0.935353 | 0.818503 | 0.873035 | 99.73 | 81.73 |
| Random forest (upsampled) | 0.993627 | 0.989014 | 0.998354 | 0.993662 | 99.01 | 99.73 |
| Random forest (downsampled) | 0.940647 | 0.938524 | 0.9451650 | 0.941833 | 93.6 | 94.45 |
| Support vector machine (without sampling) | 0.956076 | 0.454545 | 0.008839 | 0.017341 | 99.95 | 0.0088 |
| Support vector machine (upsampled) | 0.866656 | 0.989275 | 0.850039 | 0.864488 | 88.32 | 85.00 |
| Support vector machine (downsampled) | 0.863609 | 0.879047 | 0.848466 | 0.863486 | 87.92 | 84.84 |

### Random Forest Classifier

*1. Without sampling*

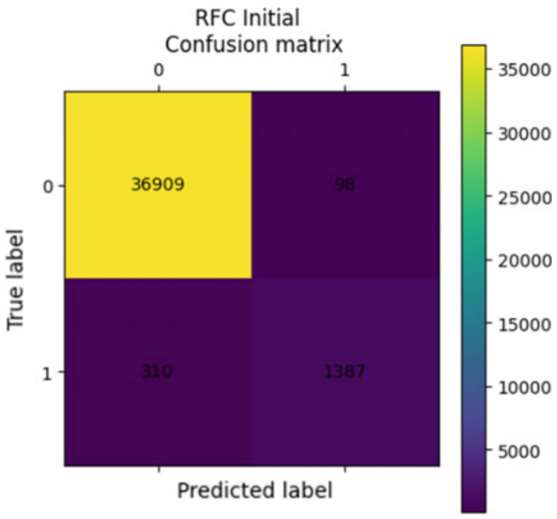


**Fig. 6** Confusion matrix obtained for Random Forest Classifier method performed on the non-sampled data

*2. For Upsampled data*

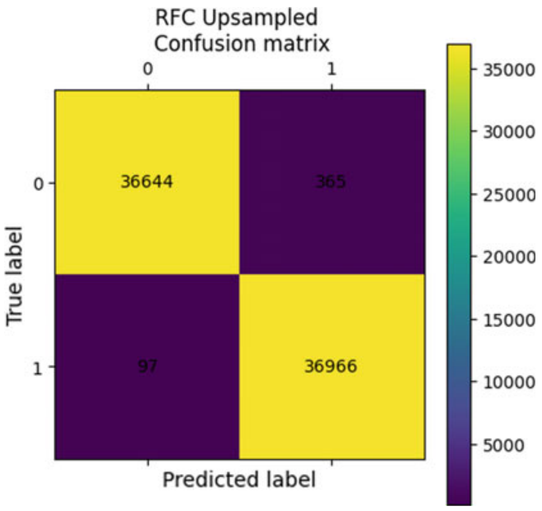


**Fig. 7** Confusion matrix obtained for Random Forest Classifier method performed on the upsampled data

*3. For Downsampled data*

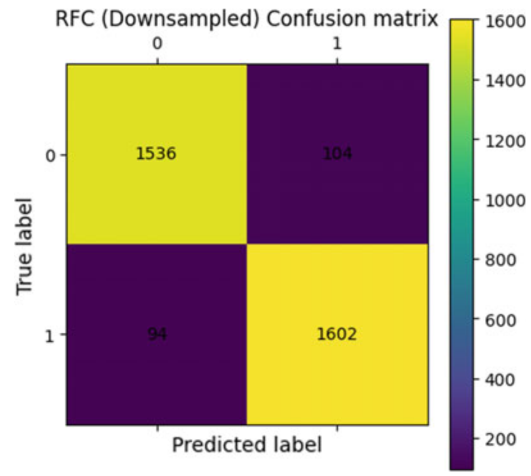


**Fig. 8** Confusion matrix obtained for Random Forest Classifier method performed on the down-sampled data

### Logistic Regression

*1. Without Sampling*

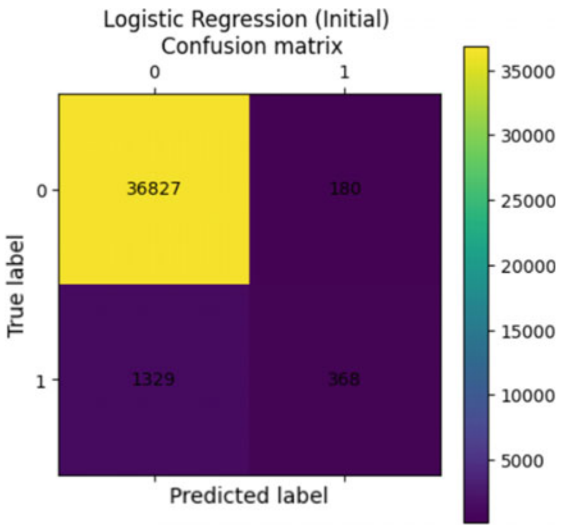


**Fig. 9** Confusion matrix obtained for Logistic Regression method performed on the non-sampled data

*2.For Upsampled data*

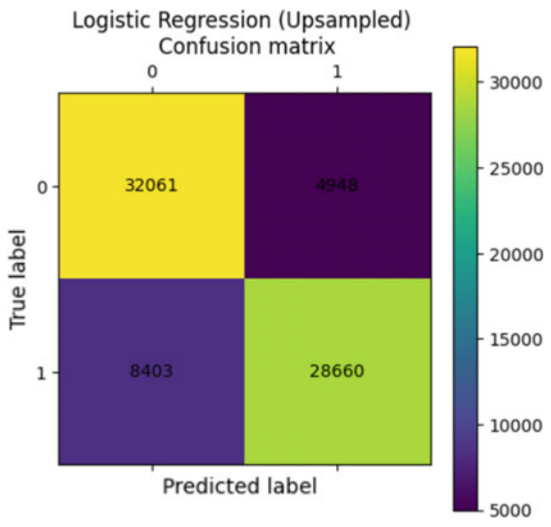


**Fig. 10** Confusion matrix obtained for Logistic Regression method performed on the upsampled data

*3.For Downsampled data*

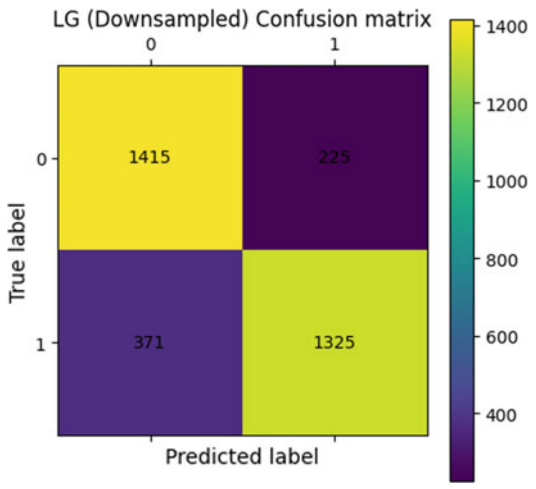


**Fig. 11** Confusion matrix obtained for Logistic Regression method performed on the downsampled data

**Support Vector Machine**

*1. Without sampling*

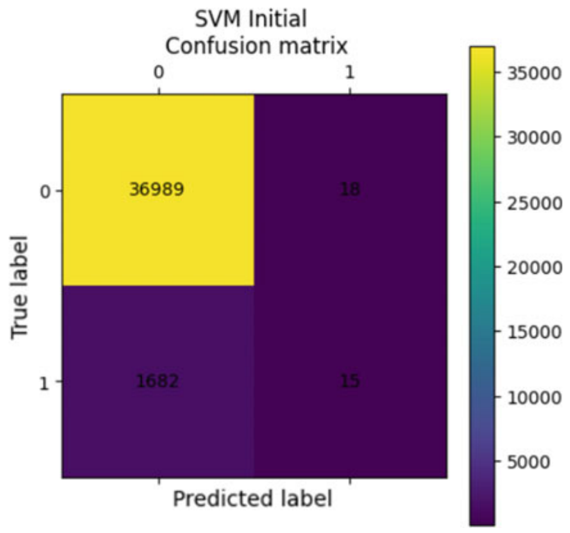


**Fig. 12** Confusion matrix obtained for SVM method performed on the non-sampled data

*2. For Upsampled data*

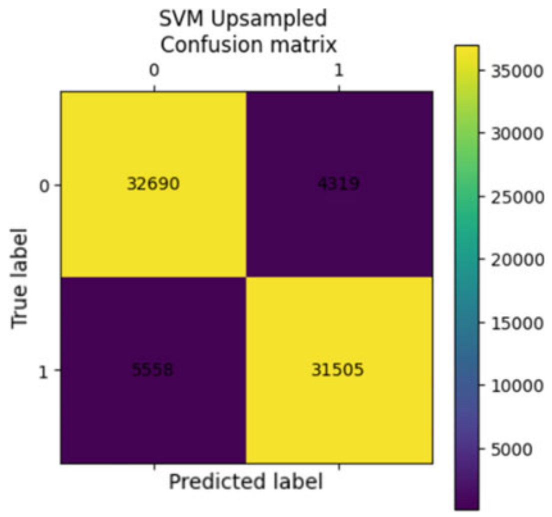


**Fig. 13** Confusion matrix obtained for SVM method performed on the upsampled data

*3. For Downsampled data*

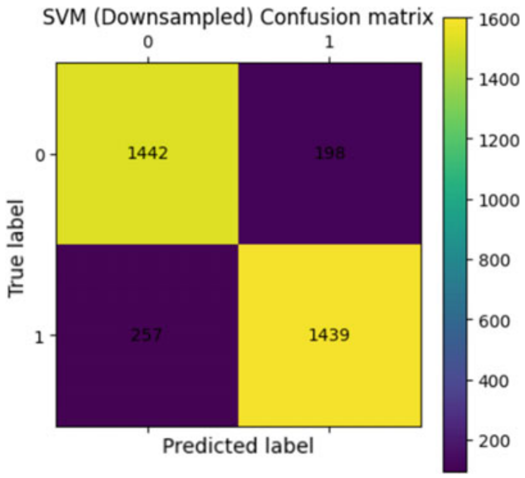


**Fig. 14** Confusion matrix obtained for SVM method performed on the downsampled data

## 6 Conclusion and Future Work

In this paper, safeware static analysis has been carried out using the AndroidManifest.xml file to extract the features such as "permissions" and "API calls" after which the results of the classification carried out by the three methods have been analysed and compared. We have concluded from the results that the Random Forest Classifier method is more effective in malware classification among the three. For future work, the software can be classified using dynamic analysis by extracting system calls.

## References

1. Patel, Z.D.: Malware Detection in Android Operating System, Department of Computer Engineering, Sarvajanik College of Engineering and Technology, Surat, India
2. de la Puerta, J.G., Sanz, B., Grueiro, I.S., Bringas, P.G.: The Evolution of Permission as Feature for Android Malware Detection
3. Huang, C-Y., Tsai, Y-T., Hsu C-H.: Performance Evaluation on Permission-Based Detection for Android Malware
4. Jehad Ali, J., Khan, R-U., Ahmad, N., Maqsood, I.: Random Forests and Decision Trees
5. Arp, D., Spreitzen-Barth, M., Hubner, M., Gascon, H., Rieck, K.: Drebin: Effective and Explainable Detection of Android Malware in Your Pocket
6. Rana, M.S., Sung, A.H.: Malware Analysis on Android Using Supervised Machine Learning Techniques, University of Mississippi
7. Wu, D.J., Mao, C-H., Wei, T-E., Lee, H-M., Wu, K-P.: Droidmat: Android Malware Detection Through Manifest and Api Callstracing

8. Ge, X, Taneja, K., Xie, T., Tillmann, N.: Dyta: Dynamic Symbolic Execution Guided with Static Verification Results
9. Jiang, Y.Z.X., Xuxian, Z.: Detecting Passive Content Leaks and Pollution in Android Applications
10. Bose, A., Hu, X., Shin, K.G., Park, T.: Behavioral Detection of Malware on Mobile Handsets
11. Li, C., Zhu, R., Niu, D., Mills, K., Zhang, H., Kinawi, H.: Android Malware Detection Based on Factorization Machine
12. Enck, W., Gilbert, P., Han, S., Tendulkar, V., GonChun, B., Cox, L.P., Jung, J., McDaniel, P., Sheth. A.N.: TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones
13. Tam, K., Khan, S.J., Fattori, A., Cavallaro, L.: CopperDroid: Automatic Reconstruction of Android Malware Behaviors
14. Wu, W.-C., Hung, S.-H.: DroidDolphin: A Dynamic Android Malware Detection Framework Using Big Data and Machine Learning