ISBN: 978-81-19149-52-0

12. MADA: Malware Application Detection Approach in Android using SVM and ANN Model

Prasun Dutta, Dhritiman Mukherjee

Department of Computer Science and Engineering,
Amity University,
Kolkata, West Bengal, India.

Siddhartha Chatterjee

Department of Computer Science and Engineering,
IMPS College of Engineering and Technology,
Malda, West Bengal, India.

Sutapa Bhattacharya

Department of Computer Science and Engineering,
Siliguri Institute of Technology,
Siliguri, West Bengal, India.

Abstract:

Advancements in growing technology and amidst the ongoing covid-pandemic period, the room for malware applications present in the internet world will only keep increasing. To tilt the scale against cybercriminals various improvements are constantly being made to detect malware applications and one of these techniques is the state of the art in machine learning. Various efforts are being made to improve the predictions both via static and dynamic analysis using appropriate feature sets and classification models. In an effort toward this direction, many researchers are working on the effective prediction of smartphone malware.

In this paper, one of the newer datasets namely CICInvesAndMal2019 has been used to extract the permission data of the apps from the manifest.xml files to perform static analysis on the same. These permission data have been trained with Support Vector Machine (SVM) and Artificial Neural Networks (ANN) (classification models) and further fine-tuned using two popular hyperparametric tuning algorithms namely grid search and genetic algorithm for SVM only to classify the nature of apps.

An accuracy of greater than 95% has been achieved, which is on par with some previous work.

Keywords:

malware, android, ANN, SVM, hyperparameter

12.1 Introduction:

One of the most popular operating systems in the market of smartphones is undeniably the Android operating system ruling over about 70% of the mobile market share. With the technological advancements throughout all these years, the features of the smartphones have only kept improving and have almost reached a point where they are inseparable from our lives, even more so during these years of the covid pandemic where it has acted as a tool and kept the people connected with their friends as well as with their job routines.

However, these technological advancements have also facilitated cybercrimes to a certain extent, and it becomes all the more important for a smartphone user to have the updated knowledge of these crimes and take the required precautionary measures to avoid falling into the traps laid by these cybercriminals. One of the categories of cybercrimes known as PUP (Potentially Unwanted Programs) refers to the set of malicious applications which are developed by cybercriminals to fetch hidden personal information from the users for their own monetary gains.

In this paper, the CICInvesAndMal2019 (Laya Taheri et al., 2019) dataset has been used to extract the permission data of the apps from the manifest.xml files to perform the static analysis. The analysis is carried out by training these permission data using classification models (SVM and ANN) and finally fine-tuning the results using two popular hyperparametric tuning algorithms namely grid search and genetic algorithm in order to classify the apps as malign or benign. The remaining paper is structured in the following manner: First the paper will highlight the related studies with respect to this field. Next, the paper will focus on the methodology used for performing this analysis. This is followed by, analysis of the results. Finally, the paper is concluded with a small note on scope of future work.

12.2 Related Study:

In one of the papers titled “Android Malware Detection Using Deep Learning”, the authors make use of the deep learning technique namely Gated Recurrent Unit (GRU) on the CICAndMal2017 dataset for malware detection in android applications. They made use of two static features namely Permissions and Application Programming Interface (API) from the applications and were successfully able to detect malware with an accuracy of 98.2%. (Omar N. Elayan & Ahmad M. Mustafa, 2021).

In another paper titled “Android Malware Detection Using Machine Learning”, the authors came up with a system wherein the user can see the percentage of malicious content of an android application. The system has a user-friendly Graphic User Interface (GUI) and performs permission-based and semantic analysis on applications to produce the desired output to user. (Rishab Agrawal et al., 2020). In the paper titled “Android Malware Classification Using Machine Learning and Bio-Inspired Optimisation Algorithms”, the authors made use of the permissions and class file of android applications to detect malware in the applications. They made use of various machine learning algorithms like SVM, SGD in combination with various bio-optimized algorithms like Particle Swarm Optimization, Genetic Algorithm, etc. for enhanced results.

These were tested on three datasets namely CICInvesAndMal2019, KuafuDet and Andro-Dump. For each of the mentioned datasets the best accuracy was achieved on Andro-Dump dataset using SGD with an accuracy of 99.6%. (Jack Pye et al., 2020).

12.3 Methodology:

12.3.1 Dataset:

The dataset used for the android malware detection analysis is CICInvesAndMal2019. The dataset consists of permissions and intents of 426 malware and 5,065 benign applications as static features. The samples of Malware that the dataset consists of are Adware, Ransomware, Premium SMS, SMS and Scareware. A tool named Androguard has been used in this paper for extracting permission-based features from different apps. On the other hand, the Jupyter notebook has been used for data preparation and model training.

12.3.2 Data Preprocessing:

In the data pre-processing step, the values which are missing and possible outliers are taken care of. However, the dataset used in this paper had no values missing nor were there any outliers as a result none of the attributes were omitted. For the analysis purpose though, only the permission data was used and therefore all the additional data were neglected.

12.3.3 Classification Models:

In this paper two classification models have been used, whose working and implementation in accordance to our extracted permission dataset are explained below:

12.3.4 Support Vector Machine (SVM):

The SVM systematically classifies the data points by first starting with data that are relatively in the lower dimensions which are the permission feature vectors of the smartphone applications, then it elevates them into a higher dimension followed by systematic segregation of the higher dimensional data into two categories (benign and malignant for this case) with the help of kernel function. One of the major kernel functions which has been used for SVM is the Radial Basis Function (RBF) which helps with computing the relationships between observations (permission related data points) in higher dimensions. The Radial Basis Function has been considered for training the model due to the fact that it is one of the main functions as it has fewer hyperparameters than the polynomial kernels, it non-linearly maps samples into higher dimensional space and faces fewer numerical difficulties. (Durgesh K Srivastava et al., 2010).

12.3.5 Artificial Neural Network (ANN):

The Artificial Neural Network used in training the model in this paper consists of the main units which are referred to as perceptron’s (permission feature vector in this analysis) and each of these are further connected to others in different layers.

These interconnections among so many perceptrons are referred to as multiple perceptron interconnections or in other terms ANN. The ANN passes on the data taken by input perceptrons to others in successive layers before it reaches the final set of perceptrons in this case the two nodes classifying the app into malignant or benign in the output layer (refer to Figure 12.1). The activation function in ANN determines if a node (malignant or benign) has to be fired or not.

For this analysis, the Rectified Linear Unit (ReLU) activation function was used as it has faster computation compared to Tanh and Sigmoid also it is one of the most used activation functions in the world right now. (Chigozie Nwankpa et al., 2020).

12.3.6 Hyperparameter Tuning:

For this analysis, in order to obtain the best possible results for the classification models (SVM and ANN), hyperparametric tuning has been used. The two algorithms implemented here are as follows:

12.3.7 Grid Search:

The algorithm goes through all the different combinations of hyperparameters. Followed by evaluating the performance value for each of these combinations and selects the best possible combination as hyperparameter (refer to Figure 12.2 for flow chart of grid search with respect to malware detection).

The analysis makes use of the GridSearchCV () method available in the scikit-learn class model_selection. It consists of 4 main arguments which are:

- **estimator:** it refers to the model instance (SVM and ANN in this case) for which the hyperparameters are check against.
- **param_grid:** it is a dictionary with parameter names as keys and a list of parameter values. Here the param_grid is made accordingly for the SVM and ANN models.
- **scoring:** it refers to performance measure metric. For this analysis, scoring has been equated to none.
- **cv:** an integer representing the number of K fold cross-validation. For the SVM model 5-fold cross-validation was used before outputting the best set of hyperparameters.

12.3.8 Genetic Algorithm:

This algorithm is a subset of evolutionary algorithms used in computation. This algorithm produces plenty of possible solutions to a given problem.

These solutions further down the line undergo recombination and mutation, producing new off-springs and this process is repeated over several generations unless it reaches a fitter generation of individuals, and the entire process ends when a stopping criterion is met. (Refer to Figure 12.3 for a detailed flow chart of genetic algorithm with respect to malware detection).

12.4 Result Analysis:

The analysis was performed on Windows operating system having 16GB RAM running on an i3 processor. (Table 13.1 gives an overview of the evaluated results and results of some previous work).

12.4.1 Support Vector Machine (SVM):

For this classifier model, the permission data was split into – 80% for training and 20% for testing the model, along with 5-fold cross-validation which implies one part was used for testing and the remaining 4 parts were used as training set. This model was then further fine-tuned using grid search and genetic algorithm. Finally, cross-checking was done with the fine-tuned model by feeding it some random malware and benign apps to check if it correctly classifies them or not. (Refer to Figure 12.4).

12.4.2 Grid Search:

For grid search the Param grid parameter was defined as follows:

{'C': [0.1, 1, 10, 100, 1000],], 'kernel': ['rbf'’], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001]}

Here:

- ‘C’ is the penalty parameter to control the error.
- ‘gamma’ defines the influence of each training session on the data points.
- ‘kernel’ refers to the kernel function used for the SVM.

On running the grid search algorithm using GridSearchCV () method, following result was obtained as the best hyperparameter values: {'C’: 10, 'kernel': 'rbf', ‘gamma’: 0.1} which in turn gave an F1 score of 94.

12.4.3 Genetic Algorithm:

For the genetic algorithm, the parameters under it are defined as follows: total generations (halting criteria – 7th generation) = 7, population size (number of chromosomes in population) = 200, number of children (created during crossover) = 5, mutation rate (probability of chromosome mutation) = 5%. On running the algorithm with this defined set of parameters, it resulted in an F1 score of 95.

12.4.5 Artificial Neural Network:

For this classifier model too the permission data was split into – 80% for training and 20% for testing the model. This model was then further fine-tuned using grid search and as a final step cross-checking was done using the fine-tuned model by feeding it random malware and benign apps so as to check if it correctly classifies them or not. (Refer to Figure 12.4).

12.4.5 Grid Search:

For grid search algorithm we defined the param_grid as follows:

{'learning_rate': [‘constant’, ‘adaptive’, ‘invscaling’], 'activation': [‘identity’, ‘tanh’, ‘logistic’, ‘relu’], 'solver': ['adam', ‘lbfgs’, ‘sgd’], ‘hidden_layer_sizes’: [3, 10, 20]}

Here:

- learning rate: it determines how quickly or slowly an ANN learns a problem.
- activation: refers to the set of activation function which has to be used while training the ANN.
- solver: refers to stochastic gradient descent (SGD) based optimizer for optimizing the parameter.
- hidden_layer_sizes: represents the number of neurons to be used in a given hidden layer. For this model only one layer of hidden layer was used.

On running the grid search algorithm using GridSearchCV () method, following result was obtained as the best hyperparameter values: {'learning_rate': ‘constant’, 'activation': ‘relu’, ‘solver’: 'lbfgs', ‘hidden_layer_sizes’: 20} which in turn gave a F1 score of 94.3.

12.5 Conclusion:

This research made use of the CICInvesAndMal2019 dataset to extract the permission data out of the manifest.xml files of the android applications. Further, these extracted permission data were trained using SVM and ANN classification models and fine-tuned using grid search and genetic algorithm. Even with the limited computational resources of our laptop system both the hyperparametric algorithms managed to bring an improvement in the model’s classification accuracy for both SVM and ANN when grid search and genetic algorithm were used. In future, this research could be extended to other new updated datasets of the upcoming years and an in-depth analysis can be conducted i.e., by exploring the other classification models, feature selection techniques and improving further on the hyperparametric tuning factor in a more computationally powerful system so as to get the best classification predictions from these models.

12.6 References:

1. Taheri, Laya & Abdul kadir, Andi Fitriah & Habibi Lashkari, Arash. (2019). Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls. 1-8.
   doi: 10.1109/CCST.2019.8888430.
2. Omar N. Elayan, Ahmad M. Mustafa. (2021). Android Malware Detection Using Deep Learning.
3. Procedia Computer Science, 184, 847-852. doi: 10.1016/j.procs.2021.03.106.
4. Agrawal, Rishab & Shah, Vishal & Chavan, Sonam & Gourshete, Ganesh & Shaikh, Nahid.

5. (2020). Android Malware Detection Using Machine Learning. 1-4. doi: 10.1109/ic-ETITE47903.2020.491.
6. J. Pye, B. Issac, N. Aslam and H. Rafiq. (2020). Android Malware Classification Using Machine
7. Learning and Bio-Inspired Optimisation Algorithms. IEEE 19th International
8. Conference on Trust, Security and Privacy in Computing and Communications
9. (TrustCom), 1777- 1782. doi: 10.1109/TrustCom50675.2020.00244.
10. Srivastava, Durgesh & Bhambhu, Lekha. (2010). Data classification using support vector machine. Journal of Theoretical and Applied Information Technology. 12. 1-7.
11. Nwankpa, Chigozie & Ijomah, W. & Gachagan, Anthony & Marshall, Stephen. (2020).
12. Activation Functions: Comparison of trends in Practice and Research for Deep Learning.
13. Chen, Sen, Xue, Minhui, Fan, Lingling et al. (2018). Automated poisoning attacks and defenses
14. in malware detection systems: An adversarial machine learning approach. Computers and Security, 73, 326-344.
15. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., & Rieck, K. (2014). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. *NDSS.*
16. Fatima, Anam & Maurya, Ritesh & Dutta, Malay & Burget, Radim & Masek, Jan. (2019).
17. Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning. 220-223.
    doi: 10.1109/TSP.2019.8769039.

Table 12.1: ***Overall Classifier Evaluation Metrics for our work as well as some previous work.***

Paper	**Dataset**	**ML Algorithm**	**Accuracy**	**F1-Score**	**Recall**	**Precision**
Elayan et al	CICAndMal2017	Gated Recurrent Unit	98.2	98	99.2	96.9
Taheri et al	CICInvesAndMal2019	Random Forest	N.A.	N.A.	95.3	95.3
Chen et al.	KuafuDet	Random Forest	96.35	N.A.	N.A.	N.A.
Pye et al	CICInvesAndMal2019	SVM + ABC Optimization	95.7	92	92	92
Pye et al	KuafuDet	Neural Network	94.9	96.7	98.8	94.7
DREBIN	DREBIN	SVM	93	N.A.	N.A.	N.A.
Fatima et al	N.A.	SVM + Genetic Algorithm	95	N.A.	N.A.	N.A.

Paper	Dataset	ML Algorithm	Accuracy	F1-Score	Recall	Precision
Fatima et al	N.A	ANN + Genetic Algorithm	94.1	N.A.	N.A.	N.A.
Our Work	CICInvesAndMal2019	SVM + Grid Search	95	94	93	93

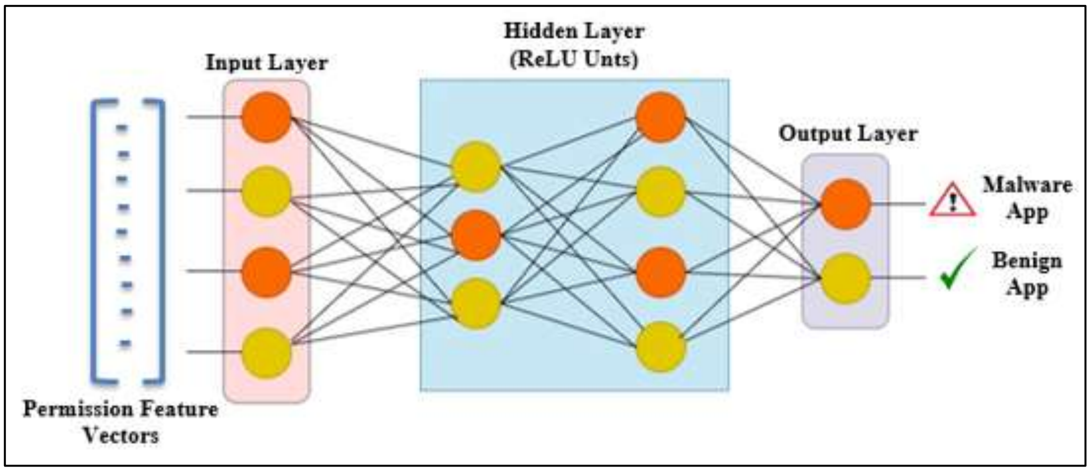


Figure 12.1: ***Architecture of Artificial Neural Network in Proposed Work.***

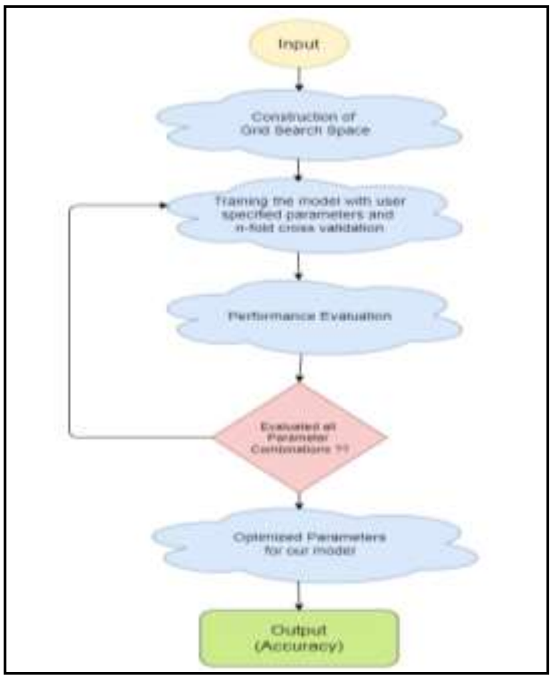

Input
Construction of Grid Search Space
Training the model with user specified parameters and n-fold cross validation
Performance Evaluation
Evaluated all Parameter Combinations ??
Optimized Parameters for our model
Output (Accuracy)

Figure 12.2: ***Grid Search, Working Flow Diagram***

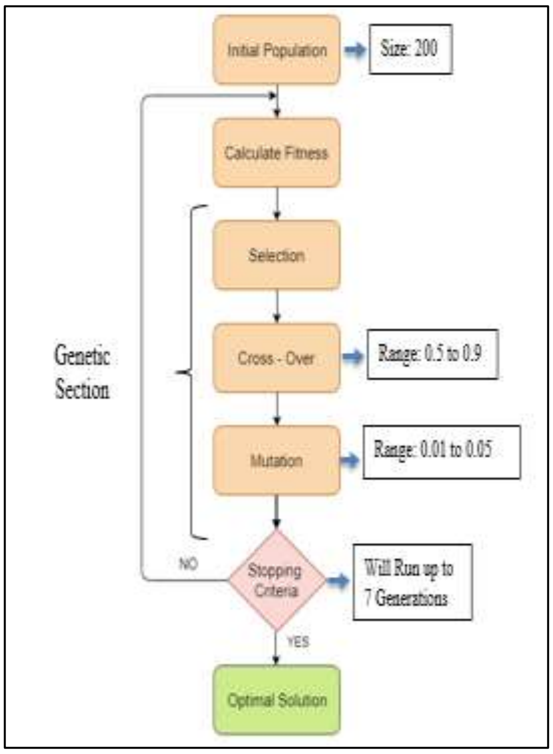
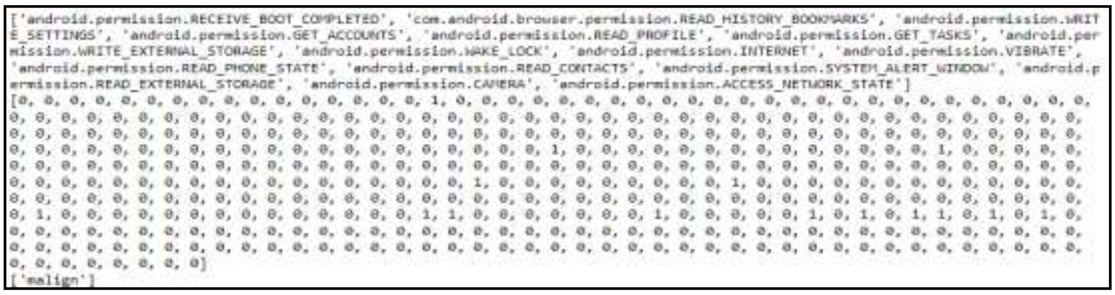
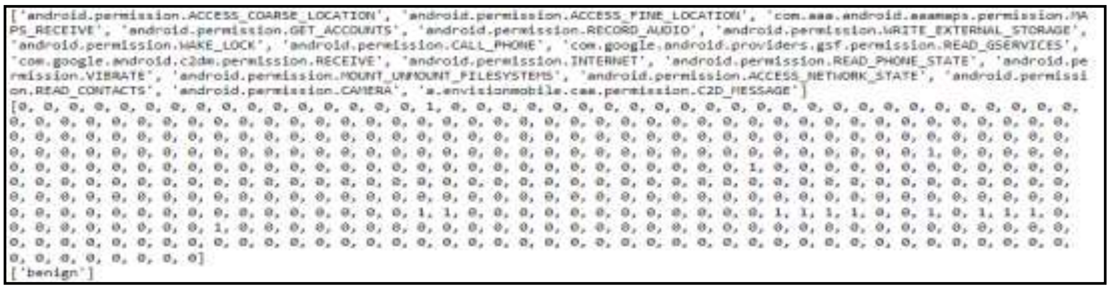


Figure 12.3: ***Genetic Algorithm, working flow diagram***



```
['android.permission.RECEIVE_BOOT_COMPLETED', 'com.android.browser.permission.READ_HISTORY_BOOKMARKS', 'android.permission.WRIT
E_SETTINGS', 'android.permission.GET_ACCOUNTS', 'android.permission.READ_PROFILE', 'android.permission.GET_TASKS', 'android.per
mission.WRITE_EXTERNAL_STORAGE', 'android.permission.WAKE_LOCK', 'android.permission.INTERNET', 'android.permission.VIBRATE',
'android.permission.READ_PHONE_STATE', 'android.permission.READ_CONTACTS', 'android.permission.SYSTEM_ALERT_WINDOW', 'android.p
ermission.READ_EXTERNAL_STORAGE', 'android.permission.CAMERA', 'android.permission.ACCESS_NETWORK_STATE']
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]
['malign']
```



```
['android.permission.ACCESS_COARSE_LOCATION', 'android.permission.ACCESS_FINE_LOCATION', 'com.aaa.android.aaamaps.permission.MA
PS_RECEIVE', 'android.permission.GET_ACCOUNTS', 'android.permission.RECORD_AUDIO', 'android.permission.WRITE_EXTERNAL_STORAGE',
'android.permission.WAKE_LOCK', 'android.permission.CALL_PHONE', 'com.google.android.providers.gsf.permission.READ_GSERVICES',
'com.google.android.c2dm.permission.RECEIVE', 'android.permission.INTERNET', 'android.permission.READ_PHONE_STATE', 'android.pe
rmission.VIBRATE', 'android.permission.MOUNT_UNMOUNT_FILESYSTEMS', 'android.permission.ACCESS_NETWORK_STATE', 'android.permissi
on.READ_CONTACTS', 'android.permission.CAMERA', 'a.envisionmobile.caa.permission.C2D_MESSAGE']
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]
['benign']
```

Figure 12.4: SVM and ANN Predict the App Classes Malware or Not Based on Their Permission Access Values.