Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2019) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

The 18th International Conference on Mobile Systems and Pervasive Computing (MobiSPC)
August 9-12, 2021, Leuven, Belgium

An Efficient Android Malware Prediction Using Ensemble machine learning algorithms

Neamat Al Sarah[a], Fahmida Yasmin Rifat[a], Md. Shohrab Hossain[b,*], Husnu S. Narman[c,*]

[a]*Department of Computer Science and Engineering, Military Institute of Science and Technology, Mirpur, Dhaka, Bangladesh*
[b]*Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh*
[c]*Department of Computer Sciences and Electrical Engineering, Marshall University, Huntington, WV, USA*

Abstract

Malwares are designed to disrupt, disable or take control of a computer system. Android malware specially targets Android OS through leakage of confidential information and crashing the system. Several attempts have been made to detect Android malware. However, those works are unable to detect malware automatically and most of them are signature based which cannot detect new variants of malware. In our work, we have explored different algorithms to obtain the best algorithm for malware prediction and to obtain the best set of features that will help us in predicting malware efficiently. From our analysis, we have seen that ensemble methods are better than traditional machine leaning algorithms for predicting malware. We have reduced the number of features from 215 to 100 achieving an accuracy of 99.5% using Light GBM. In addition, we have obtained an accuracy of 99.1% using Random Forest having only 55 features.

Keywords: Android; Malware; Machine Learning; Feature Selection; Ensemble learning; Drebin Dataset

1. Introduction

Android, a Linux-based mobile operating system, is the popular mobile OS worldwide, with a market share of 71.93% in Jan 2021 [2]. Unlike other OS which are protected by various laws and copyrights, Android is free that allows developers from all over the world to contribute to this platform. Due to its huge popularity, Android is mostly targeted for malware attacks.

Malware is programmed to interrupt the operation or exploit the data stored in any computer system. The most common way through which malware enters the android OS is through app downloads. The app which are down-

* Corresponding author. Tel.: +0-000-000-0000 ; fax: +0-000-000-0000.
E-mail address: neamatsarah@gmail.com

loaded from recognized sources often do not have malware but apps from illegitimate and lesser known sources have malwares. Often devices have vulnerabilities that the hackers exploit and attack with malware.

There are many existing solutions to predict Android malwares. But these solutions are based on signatures which is actually digital footprints underneath the code. The signatures are extracted from the APK of the app and then compared with the malicious signatures stored in the corresponding database. However, this type of solution is not able to detect unknown malwares which are not present in the database [3]. The number of attacks on mobile devices increased by 50% in 2019, from 40,386 in 2018 to 67,500 in 2019 [5]. Due to the growing malware, it is necessary to propose a solution which will be able to detect all types of malware accurately with a minimum amount of time and resources. The aim of this paper is to provide a solution which will be able to detect malwares from several families efficiently and accurately with minimum time and resources.

There have been many attempts for android malware detection. Traditional approach which are mainly signature based detects malware by comparing the APK files signature with malicious app's signature in the malware database thus failing to detect malware not present in the database.

The main *contributions* of this paper are (i) automating the process of malware detection and identify the most significant and common features in malware, (ii) achieving higher accuracy with minimum number of false positives and (iii) identifying the algorithm that suits the best in predicting malware under various criteria.

In our analysis, we have worked with Drebin dataset which has 215 features. The *novelty* of our work is that we have analysed the dataset with 8 different machine learning algorithms along with feature selection using Recursive Feature Elimination (RFE) method. Here we have applied traditional machine learning algorithms and ensemble machine learning methods for classification and have observed the accuracy and other evaluation matrices of them.

From our results, we have seen that LightGBM, which is an ensemble machine learning method, have given us a better accuracy of 99.5% than any other algorithms. Furthermore, we have analysed our results by selecting a range of features in predicting malware for android applications.

The rest of the paper is organized as follows. In Section 2, we describe all the related works regarding Android malware. In Section 3, a brief discussion of our proposed methodology is presented for the prediction of Android malware. Section 4 discusses the results and analysis from our experimentation. Section 5 discusses the limitations, contributions and future works of our system.

2. Literature Review

There have several works on Android malware detection in the literature. Khariwal et al. [4] used information gain to obtain the best set of permissions and intents to detect Android malware with higher accuracy. An accuracy of 94.73% using Random Forest was achieved. However, their solution would fail to detect malware having very few permissions and intents. Wang et al. [7] used multiple deep neural networks and combined their predictions with ensemble for final prediction achieving 98.3% precision rate and 98.1% malware recall rate in Android malware detection. Mahindru et al. [6] a web based solution where 21 algorithms have been used for malware detection and 6 feature ranking approaches and 4 feature subset selection approaches to select the best features. An accuracy of 98.8% is achieved. However, the proposed solution does not state the number of possible permissions and API calls are required to detect malware.

Sun et al. [9] uses 3-level pruning to identify the significant permissions required for effectively detecting android malware. It can detect 93.62% of malware in the data set, and 91.4% unknown malware.The main limitation of SIGPID is that if a type of malware signatures is not available in dataset, the system would not be able to detect that particular type. Wang [8] uses both string features and structural features to characterize the static behaviors of Android apps. An accuracy of 95.8% is achieved if only string features are used and 90.68% is achieved if only structural features are used and 98.4% is achieved if an ensemble of both types of features is used. This approach lacks the capability of run time analysis.

2.1. *Overall Gap Analysis*

- Most of the works used traditional machine learning ad deep learning algorithms.

- Most of the research works have aimed to find the best set of static features using filter methods, such as information gain and chi squared tests.
- In every feature selection method the number of features to be selected has to be explicitly defined which is not realistic.

3. Proposed Approach

In our proposed approach, we have used ensemble learning algorithms where several machine learning techniques are combined in order to produce one predictive model. We have used wrapper methods, such as Recursive Feature Elimination (RFE) for feature selection. We have also worked with Recursive Feature Elimination with Cross-Validation (RFECV) for feature selection where the algorithm automatically selects the number of features needed to classify the android malware thus achieving higher accuracy. After data collection our main stage of operation begins.

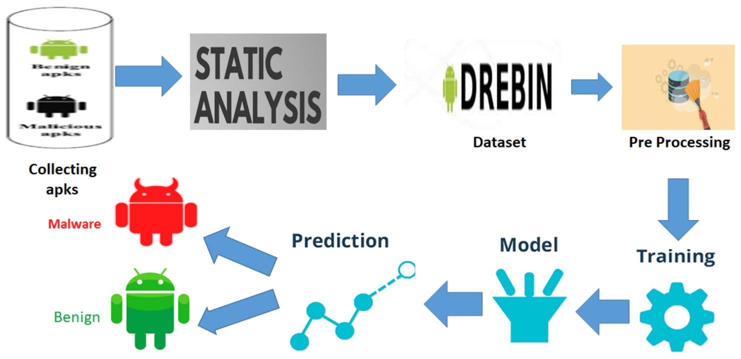

Fig. 1. Flowchart describing the overall activities of android malware prediction using machine learning

Our proposed approach consists of four phases as shown in Fig. 1.

3.1. Data Preprocessing

Data Preprocessing This is the phase where we have processed the raw dataset so that it becomes suitable for machine learning model. As our dataset was already in a good condition we did not have to do much preprocessing here. Here our data inputs were in the form of 0 or 1 where 1 means the presence of a particular attribute for an entity and 0 means its absence.

- **Identifying and handling missing data:** Here the rows and columns which had values other than 0 or 1 were identified as missing data. These missing data had to be handled so that runtime error or wrong prediction does not occur.

- **Balancing imbalanced classes:**An imbalanced class is an arrangement of data where most of the examples belong to one class thus causing very few examples to belong to other classes. This causes minority class to have a poor prediction rate whereas in most cases correct prediction of minority classes is more important like in our case. That is why class balancing is done to ensure both class have equal number of examples [**?**]. In our case majority class was the benign android samples and minority class was the malware samples. In order that both classes are equally identified by our model both classes needed to have the same ratio. To balance imbalance classes we upsampled minority classes. Upsampling is the process of randomly duplicating observations from minority classes in order to reinforce its signal.

We can see from 1 balancing the dataset causes our model to be able to classify both malware and benign classes efficiently.

Table 1. Table showing classifiaction rate of benign and malware samples for Random Forest

Sample	**Misclassification Rate for benign samples**	**Misclassification Rate for malware samples**
Before Balancing	0.38	2.56
After Balancing	0.63	1.21

3.2. Feature Selection

Feature Selection refers to techniques that select a subset of the most relevant features (columns) for a data set. Fewer features can allow machine learning algorithms to run more efficiently (less time or space complexity) and be more effective. Some machine learning algorithms can be misled by irrelevant input features resulting in worse predictive performance. For our system we have used two feature selection approaches:

3.2.1. Recursive Feature Elimination

The outcome of malware detection models rely on the features that are taken as input to design a model. In our drebin data set we had 215 features in total. We use RFE or Recursive feature extraction for selecting features that are best for our data set. There are two important configuration when using RFE:

No of features selected:. The number of features that are selected varies between 100 and 15. We took 8 categories of features where the number of features are 100, 75, 65, 55, 45, 35, 25 and 15.

Choice of algorithm used to help choose features:. We ran 8 different algorithms for each category of features So we had to run in total 8*8=64 different models.

3.2.2. Recursive Feature Elimination with Cross Validation

It is also possible to automatically select the number of features chosen by RFE and RFECV does that for us. This can be achieved by performing cross-validation evaluation of different numbers of features as we did in the previous section and automatically selecting the number of features that resulted in the best mean score. We ran RFECV for random forest and Decision tree algorithm. As in RFECV features are ranked by the model's coefficient or feature importance attributes, and by recursively eliminating a small number of features per loop. The algorithm which does not use tree structure, does not use RFECV. That is why we could not use RFECV in Gaussian Naive Bayes or logistic Regression.

3.3. Model Selection and Training

Model selection is a technique for selecting the best model after the individual models are evaluated based on the required criteria. Previously many supervised and unsupervised learning algorithms were applied to detect malware. In our study, we have applied supervised learning algorithms. From our earlier studies and the available resources, we came to the conclusion to use the following eight algorithms that will be best suited for our model. Here we have used a total of 8 algorithms for model training. Among these 8 candidate models choosing one for our prediction of malware is model selection. These 8 algorithms are divided into two classes:

- Traditional machine learning algorithms
- Ensemble machine learning algorithms

3.4. Parameter tuning and Model Testing

When creating our machine learning model we were presented with different choices as to how to define our model architecture.Often we don't immediately know what the optimal architecture should be for a given model and thus

Table 2. Table for showing traditional and ensemble learning algorithms

Traditional Machine learning Algorithms	Ensemble Machine learning Algorithms
Logistic Regression	Random Forest
Gaussian Naive Bayes	Gradient Boosting Decision Tree
Support Vector Machine	Light GBM
Decision Tree	XGBoost

Table 3. Tuning parameters

Name of the Algorithms	**Parameters**
Gradient Boosting Decision Tree	n_ estimators,learning_rate
Light GBM	max_depth, n_estimators, learning_rate, num_leaves, colsample_bytree, objective
XGBoost	objective, n_estimators, learning_rate, colsample_bytree, sub_sample, eval_metric
Decision Tree	max_depth, min_samples_split, min_samples_leaf, max_features
Logistic Regression	max_iter, C, penalty
SVM linear vector	kernel

we needed to explore a range of possibilities which is parameter tuning. Table 3 lists the algorithms along with the hyper-parameters which were tuned.

Model testing refers to the process where the performance of a fully trained model is evaluated on unseen/testing data. We applied two approaches for model testing:

- **Train-Test Split:** Here our dataset is divided into two parts. The first part is the training set and the second part is the test set. We have used the training set to train our model so it is the part of the data that is seen by our model. The test set which is unseen by our model is used to evaluate the performance of our model on unseen examples.
- **Cross Validation:** In cross validation dataset is divided multiple times into training and test set and model training and testing is also done on these multiple sets. Here we have used k-fold cross validation technique. In this the dataset is divided into k subsets(folds). Here K iteration occurs. At each iteration k-1 folds are used for training the model and 1 fold is used for testing the model. Once all the iterations are completed one can calculate the average prediction rate for each model. Here, we have taken the value of k=5 or 10.

4. Results

As we had used two approaches for testing so we will get two accuracy scores for each model. One is for train test split and another for cross validation.

4.1. Train Test Split

For train test split, first we calculate the accuracy of different algorithms before feature selection is done and here we can see from the Fig 2 that the accuracy of Light-GBM is the highest and Gaussian Naive Bayes is the lowest. Also we can see the accuracy of all the models except Naive Bayes are close to each other. From Fig 3, we see that the accuracy of Light GBM is highest (99.29%) but as the number of features decreases accuracy also decreases. In case of Random Forest, as the number of feature decreases accuracy decreases slightly. Also we can see that the accuracy of the graph does not change much after feature number is increased beyond 55. So we can say that 55 is the optimal number of features for our model.

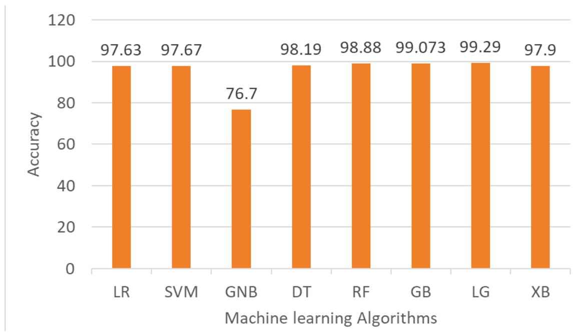

Fig. 2. Accuracy of different algorithms for train-test split before feature selection

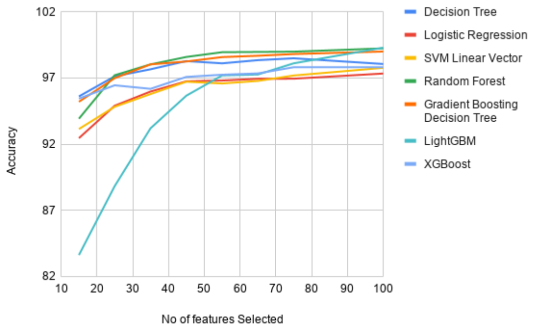

Fig. 3. Accuracy of different algorithms for train-test split after feature selection

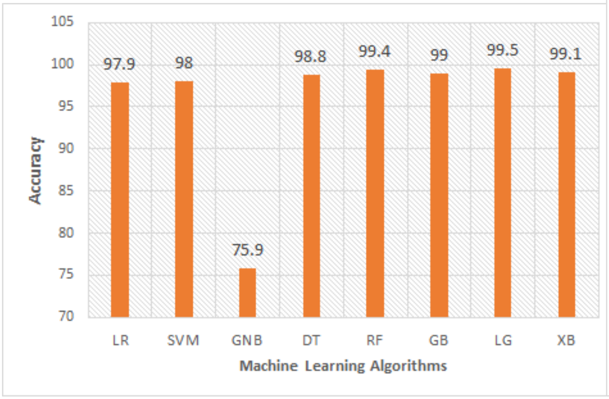

Fig. 4. Accuracy of different algorithms for cross validation before feature selection

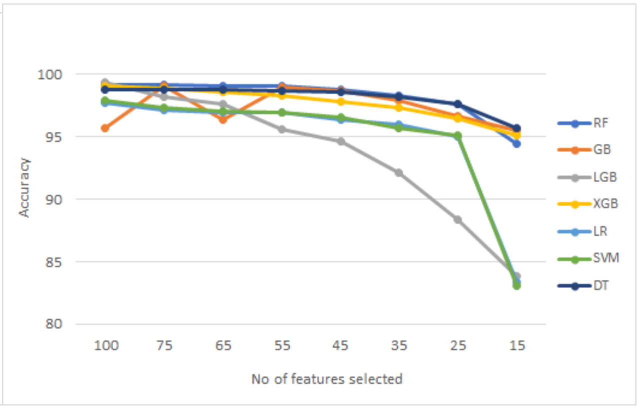

Fig. 5. Accuracy of different algorithms for cross validation after feature selection

4.2. Cross Validation

For cross validation, first we calculate the accuracy of different algorithms before any feature selection is done. In Fig 4, Light-GBM has the highest accuracy which is 99.5% and Gaussian Naive Bayes being the lowest with an accuracy of 75.9%. As in our data set all the data was in the form of 1 or 0 that is why most of the models has an high accuracy. From Fig 5, we can see that here also LightGBM has the accuracy which decreases with decrease of features. Whereas Random Forest has a steadier graph compared to LightGBM. Here we can see from the graph that after 60 features, all the lines becomes close to straight line except for Gradient Boosting. So we can also come to conclusion that optimal number of feature after feature selection can be 55.

4.3. Performance Metrics

For evaluation of our models, various performance metrics are used which are listed in Table 4.

4.4. Experimental Results

We evaluated these models' performances for both the train set and the test set and compared among them. Each of the results of applying the algorithms on train and test data-set are outlined in Table 5.

Among all ML algorithms, we have decided to choose the LightGBM as the most accurate one. It is based on the train data we labeled for the system. LightGBM depends on decision tree, it parts the tree leaf wise with the best fit though other boosting algorithms part the tree level wise. So when developing on a similar leaf in LightGBM, the leaf-

Table 4. Performance Metrics

Metrics	**Description**
TP - True Positive	Correctly Predicted Positive
FP - False Positive	Incorrectly Predicted Positive
FN - False Negative	Incorrectly Predicted Negative
TN - True Negative	Correctly Predicted Negative Values
Accuracy(α)	The percentage of correct predictions for the test data. $\alpha = \frac{TP+TN}{TP+TN+FP+FN}$
Precision(ρ)	Percentage of positive instances out of the total predicted positive instances. $\rho = \frac{TP}{TP+FP}$
Recall(r)	Percentage of positive instances out of the total actual positive instances. $r = \frac{TP}{TP+FN}$
F1 score (η)	Harmonic mean of precision and recall. $\eta = 2 \times \frac{\rho \times r}{\rho + r}$

Table 5. Accuracy, Precision, Recall and F1_score of ML models

Model	**Train**				**Test**			
	Accuracy	**Precision**	**Recall**	**F1_score**	**Accuracy**	**Precision**	**Recall**	**F1_scored**
LightGBM Classifier	0.9983	1.00	0.9967	0.9984	**0.9937**	**0.99**	**0.99**	**0.99**
XGBoost Classifier	0.9873	0.9928	0.9821	0.9874	0.9821	0.9946	0.976	0.9852
Random Forest Classifier	0.9984	1.00	1.00	1.00	0.9917	0.9917	0.99167	0.99
Logistic Regression	0.9812	0.98	0.98	0.98	0.976	0.976	0.976	0.98
SVM Linear Vector	0.9831	0.98	0.98	0.98	0.978	0.978	0.978	0.98
Decision Tree	0.9984	1.00	1.00	1.00	0.984	0.984	0.984	0.98
Gaussian Naive Bayes	0.7582	0.759	0.826	0.76	0.7576	0.822	0.759	0.76
Gradient Boosting	0.9956	0.996	0.996	1.00	0.9912	0.99	0.99	0.99

wise algorithm can decrease more misfortune than the level-wise and henceforth brings about much better exactness which can once in a while be accomplished by any of the current boosting calculations. Additionally, it is shockingly quick, henceforth the prefix 'Light' in it. Here in Table 5, for ensemble method, accuracy is higher and very close to 100% (In LightGBM, it is 99.37%) but in traditional machine learning model the accuracy varies and Gaussian Naive Bayes performed worst because Gaussian Naive Bayes model is sensitive to outliers. Generally, when an outlier exists in the data, the results may skew. Also the test accuracy is less than the train accuracy, because any model performs well with known data than the unknown ones. So here it we have seen that ensemble methods can be a better solution than traditional machine learning model in predicting malware. Because it combines the prediction from two or more models.

After using the same training dataset for the eight different machine learning algorithms we have seen that they produce different result considering different accuracy and other evaluation criteria. LightGBM has given us the best accuracy of 99.5% (Using Cross Validation method). Here in table 6 a comparison with the existing works is given.

Table 6. Comparison with other works

Different Works	**Name of the Algorithm**	**Accuracy(%)**
Mahindru, A., Sangal, A.L[6]	Multi Class Classifier	99%
Chang, Wei-Ling, Hung-Min Sun and Wei Wu[11]	Random Forest	97%
Alswaina, Fahad,and Khaled Elleithy[10]	Random Forest	95.68%
Wang, Zhaoguo,et al[8]	classical WarShall	81.8%
Yuan, Zhenlong, Yongqiang Lu and Yibo Xue [12]	Deep Learning	96.76%
Kumaran, Monica and Wenjia Li[13]	Support Vector Machine	91.7%
Kothari, Sonali, Pravin Karde and Vilas Thakare[14]	Artificial Neural Network	99%
Our Work	**LightGBM**	**99.5%**

5. Conclusion

Ensemble methods perform better than traditional machine learning algorithms. LightGBM is one of the most recent algorithms which is lightweight. In this paper, we have explored several machine learning algorithms to find out the best prediction model for detecting Android malware and also used feature selection methods to reduce the number of features thus optimizing time and resources. Our results show that lightGBM achieves the highest accuracy (99.5%) among other ensemble methods. Optimal features for LightGBM is 100 with an accuracy of 99.4%, but in Random Forest, optimal number of feature is 55 with an accuracy of 99.1%. Now if higher accuracy is concerned LightGBM can be better a solution but if efficiency or less feature is concerned then Random Forest gives consistent and better accuracy.

In future, we will focus on including dynamic analysis approach for predicting malware, testing our analysis on real live applications and taking automatic action accordingly whenever a malware is detected.

References

[1] EliteDataScience. 2021. How to Handle Imbalanced Classes in Machine Learning. [online] Available at: ¡https://elitedatascience.com/imbalanced-classes¿ [Accessed 20 April 2021]

[2] Statista. 2021. Mobile OS market share 2021 — Statista. [online] Available at: ¡https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/¿ [Accessed 20 April 2021].

[3] Wen, Long and Yu, Haiyang (2017) "An Android malware detection system based on machine learning" *AIP Conference Proceedings* **1864** (1): 020136.

[4] Khariwal, Kartik and Singh, Jatin and Arora, Anshul (2020) "IPDroid: Android Malware Detection using Intents and Permissions" *IEEE Access* **8** (1): 194729–194740

[5] Bai, Hongpeng and Xie, Nannan and Di, Xiaoqiang and Ye, Qing(2020) "AFAMD: A Fast Multifeature Android Malware Detection Framework, Design, and Implementation" *IEEE Access* **(8)**: 194729–194740

[6] Mahindru, Arvind and Sangal, AL (2020) "MLDroid—framework for Android malware detection using machine learning techniques" *Neural Computing and Applications*: 1–58

[7] Wang, Ji and Jing, Qi and Gao, Jianbo and Qiu, Xuanwei (2020) "SEdroid: A robust Android malware detector using selective ensemble learning" *2020 IEEE Wireless Communications and Networking Conference (WCNC)*:1–5

[8] Wang, Wei and Gao, Zhenzhen and Zhao, Meichen and Li, Yidong and Liu, Jiqiang and Zhang, Xiangliang (2018) "DroidEnsemble: Detecting Android malicious applications with ensemble of string and structural static features" *IEEE Access* (6): 31798–31807

[9] Sun, Lichao and Li, Zhiqiang and Yan, Qiben and Srisa-an, Witawas and Pan, Yu (2016) "SigPID: significant permission identification for android malware detection" *2016 11th international conference on malicious and unwanted software (MALWARE)* : 1–8

[10] Alswaina, Fahad and Elleithy, Khaled (2018), "Android malware permission-based multi-class classification using extremely randomized trees" *IEEE Access*, (6): 76217–76227

[11] Chang, Wei-Ling and Sun, Hung-Min and Wu, Wei (2016), "An Android behavior-based malware detection method using machine learning" *2016 IEEE International conference on signal processing, communications and computing (ICSPCC)*: 1–4.

[12] Yuan, Zhenlong and Lu, Yongqiang and Wang, Zhaoguo and Xue, Yibo (2014), "Droid-sec: deep learning in android malware detection" *Proceedings of the 2014 ACM conference on SIGCOMM*:371–372

[13] Kumaran, Monica and Li, Wenjia (2016), "Lightweight malware detection based on machine learning algorithms and the android manifest file" *2016 IEEE MIT Undergraduate Research Technology Conference (URTC)*: 1–3.

[14] Kothari, Sonali and Karde, Pravin and Thakare, Vilas (2018) "Static Analysis of Android Permissions and SMS using Machine Learning Algorithms" *International Journal of Computer Applications* 975: 8887