On the Automatic Categorisation of Android Applications

Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero and Pablo Garcia Bringas
S^3Lab
DeustoTech - Computing
Deusto Institute of Technology
Avenida de las Universidades 24, 48007
Bilbao, Spain
Email: {borja.sanz,isantos,claorden,xabier.ugarte,pablo.garcia.bringas}@deusto.com

***Abstract*—The presence of mobile devices has increased in our lives since they offer almost the same functionality as a personal computer. Android devices have appeared lately and, since then, the number of applications available for this operating system have exponentially increased. Google already has its Android Market where applications are offered and, as happens with every popular media, it is prone to misuse. A malware writer may insert a malicious application into this market without being noticed. Indeed, there are already several cases of Android malware within the Android Market. Therefore, an approach that can automatically characterise the different types of applications can be helpful for both organising the Android Market and detecting fraudulent or malicious applications. In this paper, we propose a new method for categorising Android applications through machine-learning techniques. To represent each application, our method extracts different feature sets: (i) the frequency of occurrence of the printable strings, (ii) the different permissions from the application itself and (iii) the permissions of the application extracted from the Android Market. We have evaluated this approach of automatically categorization of Android applications and show that our method achieves a high performance.**

I. Introduction

Smartphones are becoming increasingly popular. They have evolved into small and portable personal computers that allow users to browse the Internet, send e-mails or connect to a remote desktop. These small computers are able to perform complex computing tasks and communicate through different methods (e.g., GPRS, WiFi or Bluetooth).

In the last decade, users of these devices had problems installing mobile applications. They had to download an application from a website and then, install it on the device. In order to protect the device and avoid piracy, several operating systems (e.g., Symbian) employ an authentication system based on certificates that causes several inconveniences for the users (e.g., they cannot install applications despite having bought them). Nowadays, thanks to the deployment of Internet connections in mobile devices, there are new methods to distribute applications. Users can install any application without even connecting the mobile device to the computer. They only need an account of an application store in order to buy and install new applications. Apple's AppStore was the first store to implement this new model and it turned to be successful. Other manufacturers such as Google, RIM and Microsoft have followed the same business model developing application stores accessible from the device.

These facts have increased the number of developers for mobile platforms and the number of mobile applications. According to Apple[1], the number of available applications on the App Store is over 350,000, while Android Market[2] has over 200,000 applications.

Regarding the application stores, while for Apple devices the AppStore is the single official way to obtain applications, Android allows users to install applications that have been downloaded from alternative markets or directly from Internet.

According to their response to the US Federal Communication Commission's July 2009[3], Apple applies a rigorous review process made by at least two reviewers. In contrast, Android relies on its security permission system and on the user's sound judgement. Unfortunately, users usually have no security consciousness and they do not read required permissions before installing an application.

Although both AppStore and Android Market include clauses in the terms of services that urge developers not to submit malicious software, both have hosted malware in their stores. To solve this problem, they have developed tools for removing remotely these malicious applications. Therefore both models are insufficient to ensure user's safety and new models should been included in order to improve the security of the devices.

Machine learning techniques have been applied for classifying applications mainly focused on malware detection [1], [2], [3]. Their goal is to classify applications on 2 main categories: malware or goodware. There are other works [4], [5] that try to classify applications specifying the malware class, e.g., trojan, worms, virus.

With regards to Android applications, there is a lack of malware samples. Anyway, the number of samples is

[1]http://www.apple.com/iphone/features/app-store.html
[2]http://googleblog.blogspot.com/2011/05/android-momentum-mobile-and-more-at.html
[3]http://online.wsj.com/public/resources/documents/wsj-2009-0731-FCCApple.pdf

TABLE I
DESCRIPTION OF THE CATEGORIES.

Name	*Description of the category*
Communication	Applications which use the communication capabilities the device offers (e.g., Browsers, VoIP Programs)
Entertainment	Users install these small applications to personalise the device or enjoy with features of the device. Some examples are live wallpapers or ringtone makers.
Tools	These applications provide tools for a wide range of problems. FTP Clients, system monitors or application managers are some applications that are classified in this category.
Multimedia and Video	Users install these applications to use multimedia capacities of the devices. Video and music players or remote controllers are some examples.
Productivity	Applications that enhance productivity are classified here. Some examples of this category are task killers, application launchers or calendars.
Puzzles and brain games	These simple games seek to test user's capabilities through simple tests.
Society	Applications that connect the user with social networks (e.g., Twitter or Facebook).

increasing exponentially and several approaches have been proposed to detect Android Malware. Shabtai et al. [6] trained machine learning models using other features, e.g., parsing apk contained xml and counting xml elements, attributes or namespaces. To evaluate their models, they selected features using three selection methods: Information Gain, Fisher Score and Chi-Square. They obtained 89% of accuracy classifying applications into 2 categories: tools or games.

In light of this background, we propose here a new method for classifying Android applications into several categories (e.g., entertainment or productivity) using features extracted both from the Android Market and the application itself.

Summarising, our main findings in this paper are:

- We describe the process of extracting features from the Android .apk files.
- We propose a new representation for Android applications in order to develop an automatic categorisation approach.
- We perform an empirical validation of our approach and show that it can achive high accuracy rates.

The reminder of this paper is organised as follows. Section II details the representation method and shows how to extract features from the applications. Section III shows a brief introduction of the machine-learning methods we used. Section IV describes the empirical evaluation of our method. Finally, section V discusses the results and shows the avenues of further work.

II. APPS FEATURE EXTRACTION

To train the models, we extract several features both from the applications and from the Android Market:

- Strings contained in the application.
- Permissions of the applications.
- Other features extracted from the Android Market: rating, number of ratings and size of application.

We extract strings from the application because they have been previously used to identify desktop malware [1]. We also extract the permissions required by the application in order to be executed. The Android platform facilitates the identification of these permissions because they are stored in an XML file inside each application, named "AndroidManifest.xml". This file declares the requirements for the execution of the application, among other features and other requirements (e.g., version of the operating system that requires and libraries used).

We use Android Market features to add another kind of information to the model. The Android Market contains information about the behaviour of an application and, also, information users provide, such as the application rating or the number of ratings, which according to the results of the application of the Information Gain method [7], are important features.

We obtain the permissions of the application using both the Android Market and the configuration file "AndroidManifest.xml". Although they may seem redundant information, in some cases, there are several changes between permissions from the application and the Market (e.g., the difference between app version and hosted version). In these cases, Android Market provides the permissions of the last version of the application.

TABLE II
NUMBER OF SAMPLES OF EACH CATEGORY.

Category	*Number of samples*
Entertaiment	83
Puzzle and brain games	84
Comunication	111
Multimedia and video	123
Society	137
Productivity	151
Tools	160
Total	820

We developed a model to classify each Android Package File (*.apk*) in several categories. Despite they are very well-known applications, arcade games have special features and, therefore, we have omitted them in this categorisation. They usually are very large files with multimedia material (e.g., music, photos and 3D models.). The games included in this categorisation are puzzles and brain games (e.g, quiz-games or educative games). You can see the list of categories in Table I. To build the model, we have collected 820 applications, that have been classified in 7 categories. Each category has a different number of samples. Albeit machine-learning classifiers work better with a balanced dataset, we decided to increase the number of categories and samples. The number of the samples within each category is shown in Table II.

Features extraction method

In this section, we describe the process we followed to obtain data from both the Android Market and the file. The general steps we have followed for each application are:

1) We extract the permissions and resources from the application.
2) We disassemble the sample.
3) We extract the strings from the disassembled sample.
4) We obtain data from the Android Market.
5) We build an ARFF [8] file with the extracted data.

First, we decompress the *.apk* file to extract the content. During the first three steps we retrieve the information from this source. We process the Android Manifest file to extract these data (a snippet of the structure of the AndroidManifest file is shown in Listing 1).

We use the *Android Asset Packaging Tool* (aapt) tool to extract the information from the file. This tool, provided by Android SDK, decrypts the file. Thereafter, we dump the result and process it. From all the information available, we only use the following features:

- "uses-permission": All the permissions that the application needs to work are defined under this tag.
- "uses-feature": This tag shows which are the features of the device the application uses.

We have removed the rest of the information that the AndroidManifest file stores because of its dependency on specific devices.

Listing 1. AndroidManifest.xml example.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />
     ....
</manifest>
```

Subsequently, we decompile the file, using the *dedexer* decompiler[4]. This tool reads DEX format files (i.e., binary format of Android applications) and turns them into Java's bytecode like format. This tool generates a directory structure with the classes that has identified. To extract every string, we search the operational code "const-string", that identifies the strings of the application.

We process the strings using Term Frequency (TF) and Inverse Document Frecuency (IDF) [9]. TF is a weight widely used in information retrieval and text mining that measures how important each word is to a document in a collection of documents, and it is compensated by the frequency of the word in the collection, which is IDF. The importance of each word increases proportionally to the number of times that appears in the document but is compensated by the frequency of the word in the collection.

To extract the information from the market, we used an open-source non-official API, called *android-market-api*[5]. We gather the following features from the market:

- *Number of ratings* the application has obtained.
- *Size of applications installed*. Developers provide this value.
- *Rating* that users give to the application.

III. Machine Learning Classifiers

Machine-learning is an active research area within *Artificial Intelligence* (AI) that focuses on the design and development of new algorithms that allow computers to reason and decide based on data (i.e., computer learning) [10].

Machine-learning algorithms can commonly be divided into three different types: *supervised learning*, *unsupervised learning* and *semi-supervised learning*. For supervised algorithms, the training dataset must be labelled (e.g., the category of an application) [11]. Unsupervised learning algorithms try to determine how data are organised into different groups named *clusters*. Therefore, data do not need to be labelled [12]. Finally, semi-supervised machine-learning algorithms use a mixture of both labelled and unlabelled data in order to build models, improving the accuracy of unsupervised methods [13].

Because android applications can be properly labelled, we use supervised machine-learning; however, in the future, we would also like to test unsupervised methods for automatic categorisation of applications.

A. Bayesian Networks

Bayesian Networks [14], which are based on the *Bayes Theorem* [15], are defined as graphical probabilistic models for multivariate analysis. Specifically, they are directed acyclic graphs that have an associated probability distribution function [16]. Nodes within the directed graph represent problem variables (they can be either a premise or a conclusion) and the edges represent conditional dependencies between such variables. Moreover, the probability function illustrates the strength of these relationships in the graph [16].

The most important capability of Bayesian Networks is their ability to determine the probability that a certain hypothesis is

[4]http://dedexer.sourceforge.net/

[5]http://code.google.com/p/android-market-api/

true (e.g., the probability of an application to be of a certain category) given a historical dataset.

B. Decision Trees

Decision Tree classifiers are a type of machine-learning classifiers that are graphically represented as trees. Internal nodes represent conditions regarding the variables of a problem, whereas final nodes or leaves represent the ultimate decision of the algorithm [17].

Different training methods are typically used for learning the graph structure of these models from a labelled dataset. We use *Random Forest*, an ensemble (i.e., combination of weak classifiers) of different randomly-built decision trees [18], and *J48*, the WEKA [19] implementation of the *C4.5* algorithm [20].

C. K-Nearest Neighbour

The *K-Nearest Neighbour* (KNN) [21] classifier is one of the simplest supervised machine learning models. This method classifies an unknown specimen based on the class of the instances closest to it in the training space by measuring the distance between the training instances and the unknown instance.

Even though several methods to choose the class of the unknown sample exist, the most common technique is to simply classify the unknown instance as the most common class amongst the K-nearest neighbours.

D. Support Vector Machines (SVM)

SVM algorithms divide the n-dimensional space representation of the data into two regions using a *hyperplane*. This hyperplane always maximises the *margin* between those two regions or classes. The margin is defined by the farthest distance between the examples of the two classes and computed based on the distance between the closest instances of both classes, which are called *supporting vectors* [22].

Instead of using linear hyperplanes, it is common to use the so-called *kernel functions*. These kernel functions lead to non-linear classification surfaces, such as polynomial, radial or sigmoid surfaces [23].

IV. Empirical Validation

To validate our method for automatic categorisation of applications, we used the dataset described previously. We extracted the different features of the instances and then performed a feature selection step using Information Gain (IG) [7]. We discarded any feature with an IG value of 0. In this way, we compose the final dataset with the reduced number of features.

To evaluate the performance of machine-learning classifiers, *k-fold cross validation* is usually used in machine-learning experiments [10]. This technique assesses how the results of the predictive models will generalise to an independent data set. It involves partitioning the sample of data into subsets, performing the training step with one subset (called the training set) and validating with the remaining dataset (called the test set). To reduce the variability, cross validation performs multiple rounds with different partitions, which are defined with the parameter k. The results of each round are averaged to estimate the global measures of the tested model. Thereby, for each classifier we tested, we performed a k-fold cross validation [24] with $k = 10$. In this way, our dataset was split 10 times into 10 different sets for learning (90% of the total dataset) and testing (10% of the total data).

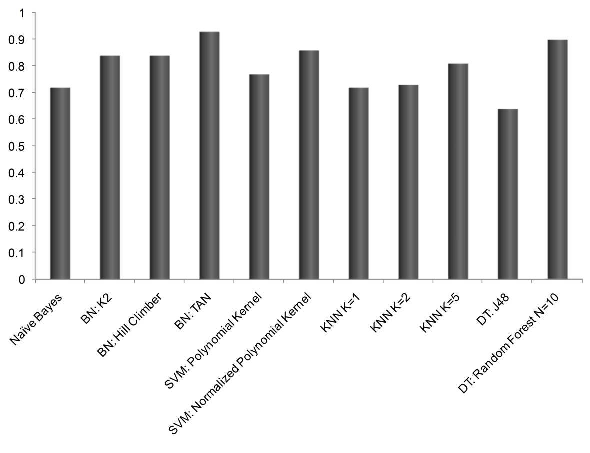

Fig. 1. The results in terms of AUC.

For each validation step, we conducted the learning phase of each algorithm with each training dataset, applying different parameters or learning algorithms depending on the concrete classifier. The algorithms uses the default parameters in the well-known machine-learning tool WEKA [19]. Specifically, we used the following four models:

- *Decision Trees (DT):* We used *Random Forest* [18] and *J48* (Weka's *C4.5* [20] implementation).
- *K-Nearest Neighbour (KNN):* We performed experiments for $k = 1$, $k = 2$ and $k = 5$ to train KNN.
- *Bayesian networks (BN):* We used several structural learning algorithms; *K2* [25], *Hill Climber* [26] and *Tree Augmented Naïve* (TAN) [27]. We also performed experiments with a *Naïve Bayes* classifier [28].
- *Support Vector Machines (SVM):* We used a *Sequential Minimal Optimization* (SMO) algorithm [29] and performed experiments with a *polynomial kernel* [23] and a *normalised polynomial kernel* [23].

To evaluate each classifier's capability, we measured the *Area Under the ROC Curve* (AUC), which establishes the relation between false negatives and false positives [30]. The ROC (Receiver Operator Characteristics) curve is obtained by plotting the TPR against the FPR.

Figure 1 shows the obtained results for the different classifiers. Specifically, Bayes TAN was the best classifier obtaining an Area Under the ROC Curve of 0.93. Random Forest was the second best classifier with an AUC of 0.9. The worst classifier was the Decision Tree trained with the J48, which only obtained a 0.64 of AUC.

V. Discussion and Conclusions

In this paper we show a method for classifying Android applications using machine-learning techniques. To generate the models, we have extracted several features from several applications: (i) data from "AndroidManifest.xml", where all permissions and features that an application needs to work are defined; (ii) data from Android Market, where users evaluate and comment the applications and (iii) strings contained in applications.

In order to validate our method, we have collected 820 samples of Android applications classified into 7 different categories. Then, we have extracted the aforementioned features for each application and we have trained the models which have been evaluated using the Area Under ROC Curve (AUC). We have obtained a 0.93 of AUC using the Bayes TAN classifier. Nevertheless, there are several considerations regarding the viability of this method.

First, a better classification of the applications used for training should be done. Despite we use the category in which the applications are stored in the Android Market, there are several applications that belong to several categories. Applications are categorised according to the developers' criterion, who sometimes prefer categories that have more visibility (e.g, "tools" category rather than "communication").

Second, although Market features enhance the results, they introduce an increase of the processing time required that may be avoidable using other features from the .apk file. This could allow us to improve the performance overhead of our approach.

Finally, there are other features from the applications that could be used to improve the detection ratio. Forensic experts are developing reverse engineering tools over Android applications, from which researchers could extract new features to enhance the data used to train the models.

Regarding future work, despite our dataset is composed of goodware, similar methods using machine-learning techniques trained with extracted features from the binary have been successfully applied on PC environments [1], [2]. Therefore, we will train models with both goodware and malware samples as soon as we obtain enough samples of malicious software.

References

[1] M. Schultz, E. Eskin, F. Zadok, and S. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001, pp. 38–49.

[2] I. Santos, C. Laorden, and P. G. Bringas, "Collective classification for unknown malware detection," in *Proceedings of the 6^{th} International Conference on Security and Cryptography (SECRYPT)*, 2011, in press.

[3] Y. Ye, D. Wang, T. Li, and D. Ye, "Imds: Intelligent malware detection system," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2007, pp. 1043–1047.

[4] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *Proceedings of the 2008 Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Springer, 2008, pp. 108–125.

[5] R. Tian, L. Batten, R. Islam, and S. Versteeg, "An automated classification system based on the strings of trojan and virus families," in *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*. IEEE, 2009, pp. 23–30.

[6] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated Static Code Analysis for Classifying Android Applications Using Machine Learning," *2010 International Conference on Computational Intelligence and Security*, pp. 329–333, Dec. 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5696292

[7] J. T. Kent, "Information gain and a general measure of correlation," *Biometrika*, vol. 70, no. 1, pp. 163–173, 1983.

[8] G. Holmes, A. Donkin, and I. H. Witten, "Weka: a machine learning workbench," August 1994, pp. 357–361.

[9] G. Salton and M. McGill, *Introduction to modern information retrieval*. McGraw-Hill New York, 1983, vol. 1.

[10] C. Bishop, *Pattern recognition and machine learning*. Springer New York., 2006.

[11] S. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," in *Proceeding of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, 2007, pp. 3–24.

[12] S. Kotsiantis and P. Pintelas, "Recent advances in clustering: A brief survey," *WSEAS Transactions on Information Science and Applications*, vol. 1, no. 1, pp. 73–81, 2004.

[13] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-supervised learning*. MIT Press, 2006.

[14] J. Pearl, "Reverend bayes on inference engines: a distributed hierarchical approach," in *Proceedings of the National Conference on Artificial Intelligence*, 1982, pp. 133–136.

[15] T. Bayes, "An essay towards solving a problem in the doctrine of chances," *Philosophical Transactions of the Royal Society*, vol. 53, pp. 370–418, 1763.

[16] E. Castillo, J. M. Gutiérrez, and A. S. Hadi, *Expert Systems and Probabilistic Network Models*, erste ed., New York, NY, USA, 1996.

[17] J. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[18] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[19] S. Garner, "Weka: The Waikato environment for knowledge analysis," in *Proceedings of the 1995 New Zealand Computer Science Research Students Conference*, 1995, pp. 57–64.

[20] J. Quinlan, *C4. 5 programs for machine learning*. Morgan Kaufmann Publishers, 1993.

[21] E. Fix and J. L. Hodges, "Discriminatory analysis: Nonparametric discrimination: Small sample performance," *Technical Report Project 21-49-004, Report Number 11*, 1952.

[22] V. Vapnik, *The nature of statistical learning theory*. Springer, 2000.

[23] S. Amari and S. Wu, "Improving support vector machine classifiers by modifying kernel functions," *Neural Networks*, vol. 12, no. 6, pp. 783–789, 1999.

[24] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial Intelligence*, vol. 14, 1995, pp. 1137–1145.

[25] G. F. Cooper and E. Herskovits, "A bayesian method for constructing bayesian belief networks from databases," in *Proceedings of the 1991 conference on Uncertainty in artificial intelligence*, 1991.

[26] S. J. Russell and Norvig, *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.

[27] D. Geiger, M. Goldszmidt, G. Provan, P. Langley, and P. Smyth, "Bayesian network classifiers," in *Machine Learning*, 1997, pp. 131–163.

[28] D. Lewis, "Naive (Bayes) at forty: The independence assumption in information retrieval," *Lecture Notes in Computer Science*, vol. 1398, pp. 4–18, 1998.

[29] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," *Advances in Kernel Methods-Support Vector Learning*, vol. 208, 1999.

[30] Y. Singh, A. Kaur, and R. Malhotra, "Comparative analysis of regression and machine learning methods for predicting fault proneness models," *International Journal of Computer Applications in Technology*, vol. 35, no. 2, pp. 183–193, 2009.