User Feedback Analysis for Mobile Malware Detection

Tal Hadad, Bronislav Sidik, Nir Ofek, Rami Puzis and Lior Rokach
Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Be'er Sheva, Israel
{*tah, nirofek*}*@post.bgu.ac.il, slavik.sid@gmail.com,* {*puzis, liorrk*}*@bgu.ac.il*

Keywords: Mobile Malware, Malware Detection, User Feedback Analysis, Text Mining, Review Mining.

Abstract: With the increasing number of smartphone users, mobile malware has become a serious threat. Similar to the best practice on personal computers, the users are encouraged to install anti-virus and intrusion detection software on their mobile devices. Nevertheless, their devises are far from being fully protected. Major mobile application distributors, designated stores and marketplaces, inspect the uploaded application with state of the art malware detection tools and remove applications that turned to be malicious. Unfortunately, many malicious applications have a large window of opportunity until they are removed from the marketplace. Meanwhile users install the applications, use them, and leave comments in the respective marketplaces. Occasionally such comments trigger the interest of malware laboratories in inspecting a particular application and thus, speedup its removal from the marketplaces. In this paper, we present a new approach for mining user comments in mobile application marketplaces with a purpose of detecting malicious apps. Two computationally efficient features are suggested and evaluated using data collected from the "Amazon Appstore". Using these two features, we show that feedback generated by the crowd is effective for detecting malicious applications without the need for downloading them.

1 INTRODUCTION

The use of mobile devices such as smartphones, tablets, and smartwatches is constantly increasing. According to the International Data Corporation (IDC) (IDC, 2016), smartphone companies shipped a total of 1.43 billion units in 2015, representing a 10.1% increase compared to the previous year and setting a new record for the highest number of smartphones sold in a year. According to Statista (Statista, 2016), in 2015 there were more than 3.7 million applications available from the three largest application stores: Google Play, iOS AppStore, and Amazon Appstore, a 3% increase compared to the previous year.

Unfortunately, with the penetration of mobile devices and their applications into our lives, the number of security threats targeting mobile devices has increased as well. In fact, malicious users, hackers, and even manufactures of mobile devices and applications, take advantage of the growing capabilities of mobile devices, careless and unaware users, and vulnerabilities in the design of standard security mechanisms, in order to develop mobile-specific malware.

Malicious applications aim to exploit the system and application software for purposes such as the exposure of personal information, identity theft, launching unwanted pop-ups and browser redirects to download malicious files, and encrypting a victims personal information to demand money (ransom) in exchange for a decryption key.

Lookout[1] reports the presence of malicious applications in official application stores. For example, malware from the Brain Test malware family and the FruitSMS Trojan were detected in Google Play in October and December 2015, respectively.

Despite their effectiveness, antivirus engines and website scanners occasionally provide different conclusions regarding the same suspected file. Therefore, it is not always clear to users whether a suspicious application is in fact malicious or not. What do people do when they are not sure? – ask a friend or consult an expert. In this paper we bring the former option to an extreme by automatically mining the user feedbacks on any particular application.

In recent years, many academic studies have focused on detecting undesired behavior of mobile applications using static and dynamic analysis methods. Static analysis usually involves inspection of an applications code in order to identify sensitive capabilities or potentially harmful instructions. Dynamic analy-

[1] https://blog.lookout.com/blog/category/alerts/

Hadad, T., Sidik, B., Ofek, N., Puzis, R. and Rokach, L.
User Feedback Analysis for Mobile Malware Detection.
DOI: 10.5220/0006131200830094
In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 83-94
ISBN: 978-989-758-209-7
Copyright © 2017 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

sis monitors system activity (behavior) and classifies it as normal or abnormal. This classification is based on heuristics or rules and attempts to detect irregular behavior (Wang and Stolfo, 2004).

In contrast to conventional static and dynamic approaches, in this study we propose a new approach that analyzes customers reviews, using text mining and machine learning techniques.

User-generated content comes in a number of forms, including designated review sites (e.g., TripAdvisor and Yelp), and purchase/review sites (e.g., Amazon and Travelocity), and these sites provide fertile ground for the analysis of such content which can be highly useful for decision making (Blair-Goldensohn et al., 2008). Extracting and aggregating this information across opinion-rich resources is mainly used as follows. First, it allows a close look at online communities without having to directly survey the population, which is a time-consuming and expensive task (Portier et al., 2013). Second, it provides the ability to collect subjective information about a product or service in order to obtain information (the wisdom of the crowd, which is a well-known phenomenon today). Such approaches have become the de-facto standard for assessing the quality of products and services (Ofek et al., 2016)

The main contributions of this paper are as follows:

- We introduce a novel approach for malware detection that uses text analysis to analyze feedback generated by users.
- The proposed method is capable of identifying suspicious applications which can be further analyzed by static and dynamic approaches; thus the method can play a role in executing more complex and time-consuming methods on large domains such as application stores.
- Because our method uses independent data, it can also be used as a tool, serving as a tie breaker in cases in which other techniques have obtained ambiguous results.

2 RELATED WORK

To the best of our knowledge, there is no in depth work that focuses on malware detection based on text mining techniques applied to user feedback. Thus, the discussion of related work is divided into two sections: related work concerning text classification and work in the area of malware detection.

2.1 Text Classification

Methods for text classification can be based on feature focused algorithms that propose new features, and model focused algorithms that propose new classification models. Like our work, the bulk of the research in text classification focuses on feature focused algorithms and concentrates on creating new features that enable new perspectives of the analyzed data. Since raw text is a special type of data that: (1) should be pre-processed (e.g., tokenize sentences into words or normalize words) (Ofek et al., 2014), and (2) can be represented in many ways given the versatility of human language (Katz et al., 2015), various approaches for feature engineering have been proposed. These features are then fed to commonly used classifiers in order to classify the class of the text. Research based on model focused algorithms is aimed at proposing new algorithms and classification models. Approaches for feature-engineering presented in the literature are diverse, proposing both text-based features and the integration of information from multiple sources.

A number of works generate a lexicon of terms to represent each class, which then can be used to determine the class of a test instance, for example, by counting terms (Hu and Liu, 2004) or computing their joint probability (e.g.,utilizing the Naive Bayes approach) (Ofek and Shabtai, 2014). Other feature focused approaches attempt to bridge the gap between lexicon-based and learning-based approaches by dynamically setting weights to sets of predefined terms.

Works in this area include (Choi and Cardie, 2008), which uses machine learning with voting, and (Balahur et al., 2013), which emphasizes the text surrounding special entities in the analyzed text.

A simple approach involves representing a fragment of text as a term frequency (TF) vector and feeding this into a classifier (Ye et al., 2009). Such representation can be augmented with additional information about specific phrases (Mullen and Collier, 2004).

Since this approach is highly effective (Ofek et al., 2015), it has been used as a baseline in our work as part of the latent Dirichlet allocation (Blei et al., 2003).

2.2 Malware Detection

In recent years a great deal of academic research has been published proposing a wide range of methods for malware detection. We mention the academic research that, in our opinion, has provided the most significant contribution to this field.

2.2.1 Behavior-based Analysis

In (Xie et al., 2010), the authors proposed a behavior-based malware detection system (pBMDS) that correlates the users input and output with system calls in order to detect anomalous activities such as unsolicited SMS/MMS and email messages. Like other research, the authors rely on kernel calls that require root privileges on devices. (Portokalidis et al., 2010) presented the Paranoid Android framework that aims to detect viruses and run-time attacks. The authors also introduce cloud-based computation for Android systems. The authors do not refer to the privileges that their program operates with, however they discuss user-level mode. However, based on their use of the ptrace utility it seems that they need root privileges.

(Burguera et al., 2011) presented Crowdroid which is a machine learning-based framework for dynamic behavior analysis that recognizes Trojan-like malware on Android smartphones. The framework analyzes the number of times each system call has been issued by an application during the execution of an action that requires user interaction. In addition, the authors make use of the strace program that can only be operated on rooted Android devices.

In (Shabtai et al., 2014), the authors presented a new behavior-based anomaly detection system for detecting meaningful deviations in a mobile applications network behavior. The main goal of the proposed system is to protect mobile device users and cellular infrastructure companies from malicious applications by: (1) the identification of malicious attacks or masquerading applications installed on a mobile device, and (2) the identification of republished popular applications injected with a malicious code (i.e., repackaging).

Most of the research proposing application behavior analysis suffers from one of the following drawbacks. First, many of the studies make use of a secure environment, which is problematic, since their conclusions may be misleading in real-life settings, particularly considering the wide range of environments that exist. Second, many studies explore malicious behavior at the kernel level (e.g., system calls, real-time permission monitoring using the Android intent system). In such solutions the monitoring system requires root permissions to access necessary information, whereas most of the malicious applications (and potentially malicious applications) do not require special permissions (snoopwall, 2014). Moreover, a security application that requires root permissions itself becomes a threat to the security of the mobile device.

2.2.2 Permission-based Analysis

In (Aung and Zaw, 2013), the authors proposed a framework that is based on machine learning methods for the detection of malicious applications on Android. This framework is designed to detect malicious applications and enhance the security and privacy of smartphone users.

In (Zhang et al., 2013), the authors presented a new concept based on learning permissions used by applications according to user behavior. They also developed the VetDroid framework, which is a dynamic analysis platform for reconstructing sensitive behaviors in Android applications from the permission perspective.

2.2.3 Static Code based Analysis

Most static malware detection techniques suffer from inability to detect vulnerabilities introduced only at run-time. In addition, attackers have developed various techniques that are particularly effective against static analysis (Moser et al., 2007).

(Rastogi et al., 2013) proposed code transformation procedures for the Dalvik virtual machine (VM) and ARM (two engines for virtual machines on Android). The assumption behind this line of research is that malware writers rely on similar code transformation procedures to evade static signatures. Therefore, evaluating malware detectors against mutants is a good estimation of their robustness to future malware.

In (Yang et al., 2013), the authors presented a static analysis tool for privacy leaks in Android applications. This tool analyzes the intention of privacy leaks and can distinguish between intended and unintended data transmission. However, the proposed tool cannot analyze other types of undesirable behavior such as stealthily sending SMSs and cant examine the internal logic of sensitive behavior.

Zheng et al. (Zheng et al., 2013) propose the DroidAnalytics framework for static analysis which is designed to detect obfuscated malware on the Android platform. The proposed framework generates signatures in order to detect obfuscated code and repackaged malware. The detection of obfuscated and repackaged code procedure is performed on three levels: (1) the method level, (2) class level, and (3) application level.

3 THE PROPOSED APPROACH

3.1 Architectural Overview

Figure 1 presents an overview of the proposed malware detection systems architecture. The components in the figure correspond to the steps of the algorithm (pseudo code).

The systems inputs are application reviews and a malware related textual corpus, i.e., computer and network security books such as (Dunham, 2008). The output is a statistical classifier which can detect and classify malicious and benign applications based only on application reviews.

The system performs the detection and classification in three main steps: (1) the generation of a domain-specific lexicon; (2) extraction of application features based on reviews and the domain lexicon; and (3) generation of a classification model using supervised learning.

3.2 Textual Corpora

In this paper, we deal with two different types of textual corpora: (1) a domain-specific corpus, and (2) a corpus of application reviews provided by general users. These corpora are presented in natural language form, which imposes additional processing difficulties as described in (Baron, 2003). Therefore, text normalization is required in order to reduce language diversity, including transformation to canonical form for further processing.

In this paper we obtain language diversity reduction by performing textual normalization as presented in Algorithm 1. The main steps are: (1) removing numbers, punctuation, and stop words to remove noise as described in (Onix, 2016) ; (2) character replacement, including: character continuity (i.e., a character which repeats itself more than three times will be reduced to two times, e.g., goooood will be replaced by good), slang words as described in (Twitter, 2016) predetermined spelling mistakes and expressions (e.g., helpfull will be replaced by helpful), predetermined missing apostrophe (e.g., dont will be replaced with dont), and predetermined apostrophe expansion (e.g., dont will be replaced with do not); and (3) stemming each word to its root, as described in (De Marneffe et al., 2006).

```
Algorithm 1. Textual Normalization.
Input: textual document.
Output: normalized textual document.
Steps:
1. Convert characters into lower case.
2. Remove punctuation.
3. Remove numbers.
4. Remove stop words.
5. Replace continuity.
6. Replace slang terms.
7. Add apostrophe.
8. Replace apostrophe.
9. Spelling correction for known mistakes.
10. Word stemming to its root.
```

Domain-specific corpus gives us the ability to generate a domain-specific lexicon as a prior step to natural language processing (NLP). In this work we focus on the domain of cyber-security, and the extraction of relevant information has been applied on computer and network security books such as (Dunham, 2008). We chose to use books as a resource for the cyber-security domain corpus, since they contain reliable and conventional terminology for this domain.

An application review corpus is required for training a malicious application detector (i.e., classifier). Unfortunately, as far as we know, there is no publicly available corpus (or dataset) for malicious applications and their corresponding reviews. Therefore, in order to collect such data, we applied a crawler on the Android application store (Amazon, 2016). In total, we collected 2,506 applications along with their 128,863 user reviews. Each review consists of the following information: (1) textual content, (2) authors name, and (3) review rating (1-5 stats).

Additional application information is collected for each application: (1) metadata such as: size, permissions, sellers ranks, average rating, operating system, release date, update date, etc.; and (2) Android application package files (APKs), which constitute the binary representation of an application on the Android platform.

3.2.1 Domain Specific Corpus

Using the domain-specific corpus we generate domain-specific lexicon, referred to as the Domain Lexicon (DL). As a first step, the textual corpus has been extracted from computer and network security books and transformed to canonical form by applying textual normalization as presented in Algorithm 1. As a second step, from the normalized corpus, unigrams and bigrams (phrases) have been extracted along with their frequencies in the corpus. As a final step, the most frequent phrases have been selected and included in the DL.

The DL representation is a list of phrases that represent the p percent (p is an independent variable which will be determined in Section 4) of the most frequent unigrams and bigrams in the normalized domain corpus, as presented in Algorithm 2.

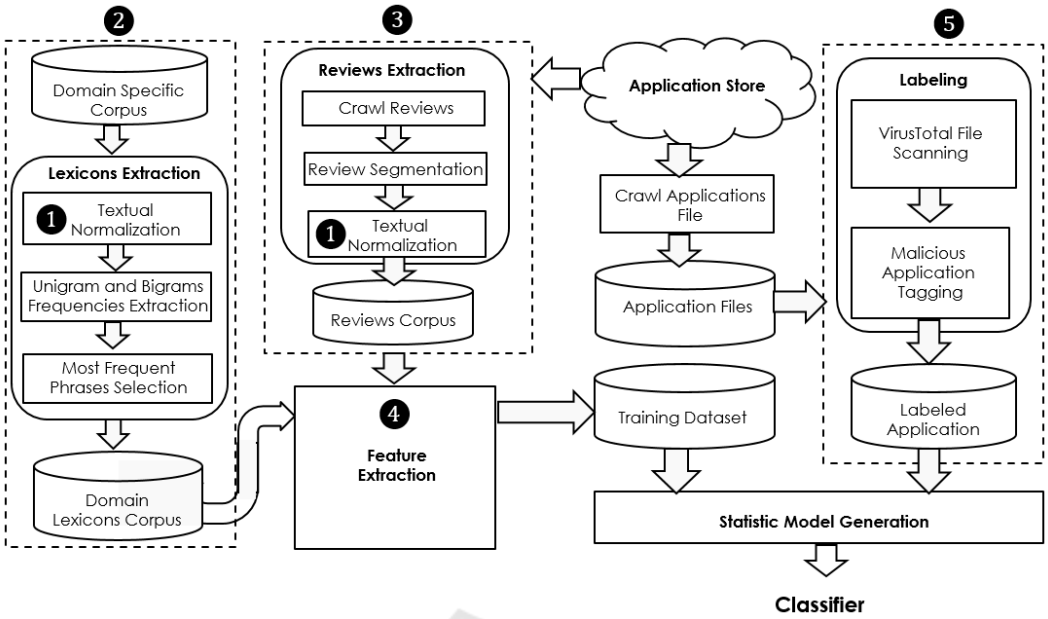

Figure 1: Flowchart of the presented malware detection system's architectural overview.

```
Algorithm 2. Creating Domain-Specific.
Lexicon
Input: security books, p.
Output: domain lexicon.
Steps:
1. Perform textual normalization on security
books.
2. Phrases <- extract unigrams and bigrams,
along with their frequencies from the
normalized security books.
3. Domain lexicon <- select the p percent of
the most frequent phrases.
```

3.2.2 Application Review Corpus

In this paper, only application reviews have been used in order to extract features, without any additional information. For this purpose we apply Algorithm 3. As a first step we crawl application reviews from an application store (*applications' URLs*). As a second step, each review is segmented into separate sentences (De Marneffe et al., 2006). This step requires in order to prevent the presence of bigrams composed of words from different sentences. Finally, each sentence is transformed to canonical form by applying textual normalization as presented in Algorithm 1.

```
Algorithm 3. Creating Application Review.
Corpus
Input: applications' URLs.
Output: reviews corpus.
Steps:
1. Crawl all reviews found in applications'
URLs pages.
2. Segment each review to sentences.
3. Perform textual normalization on each
sentence.
```

3.3 Model Generation

The proposed method uses NLP techniques for the identification of linguistic phrases that correspond to the basic phrases of a malicious applications domain. The input to the method is a set of phrases (DL) which describes the malicious applications language (as described in Section 3.2.1) and application reviews (as described in Section 3.2.2). Once those two inputs have been obtained by Algorithm 2 and Algorithm 3 respectively, the creation of an intermediate representation model proceeds, reflecting the relationships between the DL and an applications reviews. The intermediate representation model is a dataset, in which an instance represents an application, and features represent statistical information. The final stage of the method consists of generating a classification model using supervised learning, which used to classify malicious applications based only on application reviews.

The proposed method is composed of three main steps:

Feature Extraction - based on both corpora and the detection and counting of DL occurrences in an applications reviews.

Feature engineering - based on the extracted features, we generate two features for each application, which serve as input data for supervised learning (i.e., input dataset).

Supervised Learning - in this paper, performed for generating a model which is a classifier for malicious applications.

3.3.1 Features Extraction

As mentioned previously, in this paper we only use an application review corpus for feature extraction. Therefore, as a first step we apply the DL on application reviews, in order to detect and count the occurrences of each phrase $DL_i \in$DL in each application's reviews. We focus on one-star ratings, because complains on malicious behavior are usually provided along with negative feedback. This step provides a list of phrases $DL_i \in$DL and their associated occurrences in the applications one-star reviews, information that is required for the feature engineering process that follows.

3.3.2 Features Engineering

The feature engineering process is used to create two statistical features that will be used to generate a statistical model (i.e., classifier) as shown in Algorithm 4. Therefore, as a first step we denote that for review r, a review weight $w(r)$ will be the occurrences summary of each phrase $DL_i \in$DL in r as follows:

$$w(r) \leftarrow \sum_{DL_i \in DL} DL_i \; occurrences \; in \; r \qquad (1)$$

This weighting function is been used to generate the features *Review's DL* and *User's DL*.

Review's DL - represents the average number of DL occurrences found in an application reviews. Therefore, the computation of this feature is performed as a normalization by the amount of one-star reviews (R_1).

$$Review's \; DL \leftarrow \frac{\sum_{r \in R_1} w(r)}{|R_1|} \qquad (2)$$

User's DL - represents the average number of DL occurrences that have been used by a unique reviews authors. The computation of this feature is performed similarly to the preceding computation, except for the division by the amount of unique one-star reviews authors (U_1).

$$User's \; DL \leftarrow \frac{\sum_{r \in R_1} w(r)}{|U_1|} \qquad (3)$$

```
Algorithm 4. Feature Engineering.
Input: reviews of application (R1), DL.
Output: dataset instance.
Steps:
1. DL aggregation <- DL occurrences in reviews.
2. Reviews number <- amount of reviews.
3. Authors number <- amount of unique authors.
4. Create instance's features as follow:
4.1 Dataset(application).Review's DL <-
(DL aggregation)\(reviews number).
4.2 Dataset(application).User's DL <-
(DL aggregation)\(authors number).
```

3.3.3 Supervised Learning

In order to generate the model in this paper we use supervised learning algorithms. There are several approaches for supervised classification, however all of them require known labels (classes) for the training, testing, and validation stages. Therefore, the identification of labels for the presented dataset is required. (Ranveer and Hiray, 2015) research have tested static and dynamic state of the art methods for malware detection. Their results show that static analysis methods can provide accuracy between 92% to 99%, and dynamic analysis methods accuracy are between 91% to 96%. For this reason, we select static approach to serve as a gold standard for the proposed method, by labeling applications as malicious using VirusTotal (Total, 2016), as described in Algorithm 5.

For each application's APK (*APK*) we obtain labels from several antivirus (AV) vendors that provide information such as whether the given application has a label $B = \{\text{benign application}\}$ or label $M = \{\text{malicious application}\}$. In label M cases, most AVs provide additional information regarding the specific type of malicious application. Furthermore, due to the diversity and accuracy of AVs, different AVs can associate the APK with different malicious families.

Additionally, out of the more than 40 AVs hosted by VirusTotal usually only a few provide positive classification (i.e., M) for the same *APK*.

Due to non-zero false positive classification of some AVs, their results cannot be trusted when only one AV reports that an *APK* is malicious. Therefore, we need to carefully choose the minimal number of AVs required to declare an *APK* as malicious (M).

For no specific reason, related works (Šrndic and Laskov, 2013; Nissim et al., 2014) usually set $t = 5$. In our dataset we manually reviewed the applications that were classified as malicious by more than two AVs and identified no false positives. Thus, in this research we set $t = 1$ as described in Section 4.

```
Algorithm 5. Labeling.
Input: application's APK, t.
Output: application label.
Steps:
1. Positive reports <- VirusTotal scan results
of application's APK.
2. If positive reports > t,
application label = M.
```

```
3. Else, application label = B.
```

4 EXPERIMENTAL EVALUATION

In this section, we describe our evaluation settings, which include the dataset, results, accuracy, and independent variables. We start by describing the dataset used for the evaluation, including the malicious family type distribution. Furthermore, we describe the motivation behind our decision to use unigrams and bigrams rather than higher order n-gram models to generate the DL and describe independent variables p and t, where p is the percent of the most frequent phrases in the domain authors number the corpus selected to generate the DL, and t is the threshold of the AVs positive reports for the ground truth labeling approach.

Finally, we evaluate the proposed methods accuracy, based on multiple AV reports, and compare the results to two baseline methods: (1) bag-of-words (BOW) (Harris, 1954) and (2) latent Dirichlet allocation (LDA) (Blei et al., 2003), which are popular textual classification methods.

4.1 Datasets

As mentioned before, to the best of our knowledge, there is no publicly available corpus (or dataset) for malicious applications along with their end user reviews. Malicious applications whose official reviews appear in news reports are immediately removed from application stores and markets, and therefore we cannot obtain their end user reviews for in depth analysis.

In order to collect available (free) applications, a crawler was applied on an Android application store (Amazon, 2016) for a two month period (October through November 2015). Due to the large number of applications available, we randomly selected a subset of applications in order to generate the classifiers. In this paper, a single crawling session has been preformed, extracting a single version for each application. In total, we collected 2,506 applications APKs along with their 128,863 user reviews as shown in Table 1.

Each applications APK was scanned by *VirusTotal* (Total, 2016), which aggregates different antivirus products that provide an online scan engines and presents a comprehensive report regarding whether a given APK is malicious or not, including the malicious threat. In our case, the *VirusTotal Mass API* has been used, which is available for researchers to perform malicious file detection.

Note, that due to the diversity and accuracy of the AVs, scanning by different AVs can associate a single APK with different malicious families. Thus, the evaluated dataset contains many malicious types, and each application can belong to several different malicious type families, as shown in Figure 2.

As seen in the Venn diagram in Figure 2, our dataset includes the following types of malicious threats: Trojan, Adware, Viruse, Spyware, Riskware, and other less familiar malicious threats. As is known, several types of malicious threats, particularly in the mobile domain, can be classified by VirusTotal with different labels, for example Spyware, Adware, and Trojan. However, most of the applications receive a single label.

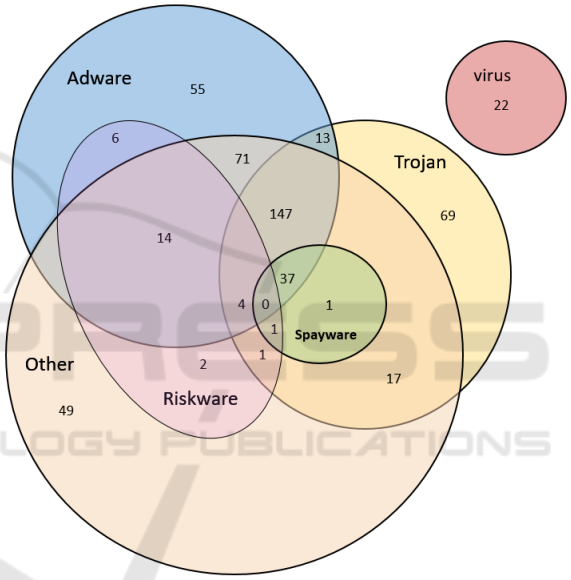

Figure 2: Malicious family distribution of 510 malicious applications.

4.2 Independent Variables

The method presented has two independent variables: (1) t, the threshold of the AVs positive reports for ground truth labeling approach; and (2) p, the percentage of the most frequent phrases in the domain corpus selected to generate the DL.

In order to determine the value of t we perform a scan of 2,506 application and summarize (Total, 2016) the scan reports using a histogram that is presented in Figure 3. In this figure Positive reports aggregate the number of AVs that label each APK as malicious. The majority (1,996 applications) were labeled as non-malicious by all of the AVs. As can be seen in Figure 3, different threshold values can be used for evaluation, as was performed by (Šrndic

Table 1: Dataset volume for evaluating the classification task.

Total Applications	Total Reviews	Malicious Applications	Benign Applications
2,506	128,863	510	1,996

and Laskov, 2013; Nissim et al., 2014). However, in this paper the authors selected a single threshold value without justification. In our paper, we perform a set of experiments in order to select the most suitable value for this parameter starting at 0, i.e., an application was labeled as *M* when at least a single AV labeled it as malicious (t>0), each evaluation differ by increase in 1 ($t \in \{0,1,2,3,4,5\}$).

In order to determine parameter p parameter we start our evaluation with p=10 up to p=50 ($p \in \{10,20,30,40,50\}$), i.e. top p% of the most frequent domain unigrams and bigrams. We used unigrams and bigrams rather than higher order n-gram models to represent the text, due to the following studies: (1) (Pang et al., 2002) which shows that unigrams beat other higher order n-grams, and (2) (Dave et al., 2003) which shows experimentally that trigrams and higher failed to show significant improvement.

The evaluation results were performed for different values of t and p, simultaneously, and are presented in Section 4.4.

4.3 Classifiers

In this paper, we used supervised machine learning methods for the classification of Android applications into malicious and benign classes. For this purpose, we have used *Waikato Environment for Knowledge Analysis* (WEKA) (WEKA, 2016).

The evaluation was performed on different classification models based on the following approaches: (1) decision tree, (2) ensemble, and (3) regression.

For each approach we use the following classification algorithms: (1) C4.5 decision tree learner (Quinlan, 1993); (2) random forest (Ho, 1998); and (3) logistic (Walker and Duncan, 1967), respectively. A brief explanation of these three algorithms follows.

1. **C4.5 decision tree learner:** The algorithm for the induction of decision trees uses the greedy search technique to induce decision trees for classification.
2. **Random forest:** An ensemble of 100 unpruned classification trees, induced from bootstrap samples of the training data, using random feature selection in the tree induction process. The prediction is made by aggregating (majority vote) the predictions of the ensemble.
3. **Logistic regression:** This algorithm allows prediction of a discrete outcome, such as group membership, from a set of variables that may be continuous, discrete, dichotomous, or a mix of any of these. In our case, we used logistic regression from a set of continuous variables. Since logistic regression calculates the probability of class M over the probability of class B, the results of the analysis are in the form of an odds ratio.

4.4 Results

Our domain has an imbalanced class distribution, namely there are many more genuine applications than malicious, due to the nature of application stores which contain a small percentage of malicious applications.

To evaluate the performance of machine learning classifiers, k-fold cross-validation is usually used (Bishop, 2006). Therefore, for each classifier that has been used, we apply k-fold cross-validation (Kohavi et al., 1995) with $k = 10$, i.e., the dataset is partitioned ten times into ten different sets. This way, each time we use 90% of the data for training and 10% for testing.

In order to measure the performance of each classifier, we measured the true positive rate (TPR):

$$TPR = \frac{TP}{TP+FN} \tag{4}$$

Where TP is the number of malicious applications correctly classified (true positives) and FN is the number of malicious applications misclassified as benign (false negatives).

In addition, we measured the false positive rate (FPR):

$$FPR = \frac{FP}{FP+TN} \tag{5}$$

Where FP is the number of benign applications incorrectly detected as malicious and TN is the number of benign applications correctly classified.

Furthermore, we measured the accuracy (the total number of the classifiers successful detection divided by the number of instances in the dataset):

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \tag{6}$$

Moreover, we measure the area under the ROC curve (AUC) which establishes the relation between false negatives and false positives (Singh et al., 2009). The ROC curve is obtained by plotting the TPR against the FPR.

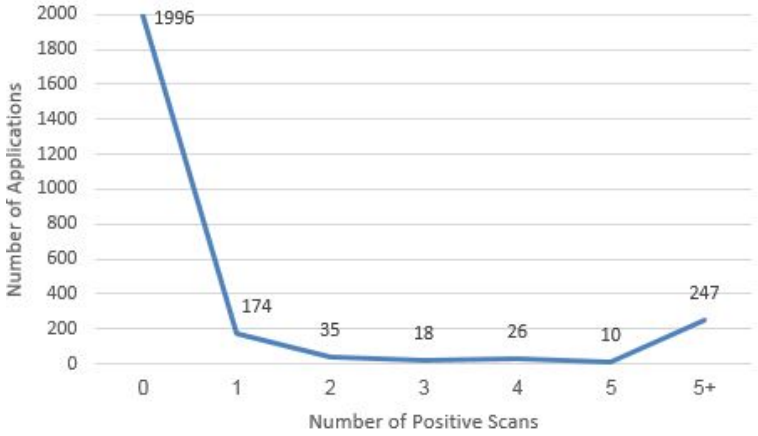

Figure 3: Histogram of 2,506 VirusTotal applications scan results.

AUC measure is independent of prior probabilities of class distributions, and therefore is not overwhelmed by the majority class instance. (Oommen et al., 2011) indicate that AUC is more robust over other measures and is not influenced by class imbalance or sampling bias.

Figures 4-6 summarizes the results of the evaluation performed for the independent variables t and p, with the three classification models mentioned in Section 4.3.

The AUC precision of our method drops slightly as the values of p and t increase by more than ten and one respectively, thus our method achieves the best performance when $p = 10$ and $t = 1$.

Table 2 presents the results obtained with the most efficient independent variable settings ($p = 10$ and $t = 1$). Using these settings, the methods achieved accuracy rates higher than 86%. In particular, the best classifier, in terms of accuracy, was logistic regression with an accuracy rate of 89%. Regarding the TPR results, random forest trained with 100 trees was the best classifier with a TPR of 38%. TPR results show that it is not feasible for using our method as a sole detection method, and further research should be performed in this aspect. In terms of AUC, logistic regression was the best classifier with an AUC of 86.4%.

Table 2: Results of different classifiers.

Classifier	TPR	FPR	AUC	Accuracy
C4.5	0.27	0.02	0.81	0.87
Random Forest	0.38	0.06	0.83	0.86
Logistic	0.23	0.02	0.86	0.89

4.5 Baselines

Popular textual classification methods such as bag-of-words (BOW) (Harris, 1954) and latent Dirichlet allocation (LDA) (Blei et al., 2003) was used to evaluate the proposed method. Table 3 present the obtained results. The BOW methods hypothesis is that the frequency of words in a document tends to indicate the relevance of the document to other documents. If documents have similar column vectors in a term document matrix, then they tend to have similar meanings. The hypothesis expresses the belief that a column vector in a term document matrix captures (to some degree) an aspect of the meaning of the corresponding document, i.e., what the document is about. However, this method generates a mathematical model of all words (perhaps with the exception of a list of high frequency noise words), without any additional knowledge or interpretation of linguistic patterns and properties.

LDA can also be cast as language-modeling work. The basic idea is to infer language models that correspond to unobserved factors in the data, with the hope that the factors that are learned represent topics. However, as (Hong and Davison, 2010) shows this method does not work well on classifying short text documents such as tweets or costumers reviews.

Table 3: Results of different baseline methods.

Method	TPR	FPR	AUC	Accuracy
Proposed	0.23	0.02	0.86	0.89
BOW	0.26	0.06	0.61	0.84
LDA	0.15	0.03	0.62	0.81

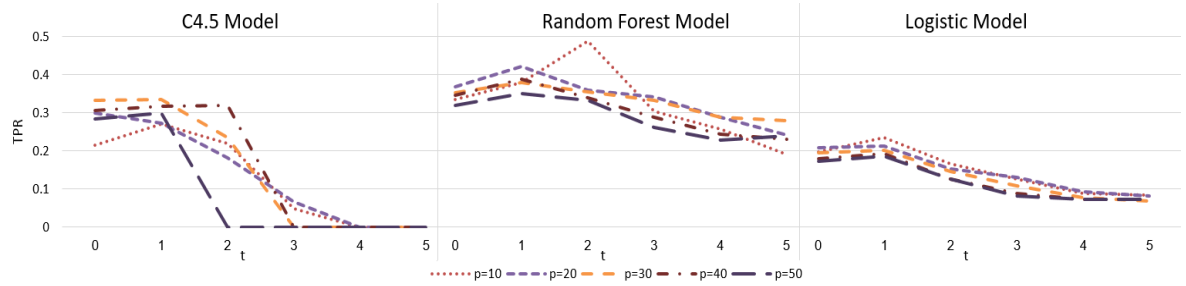

Figure 4: TPR results for three classification models.

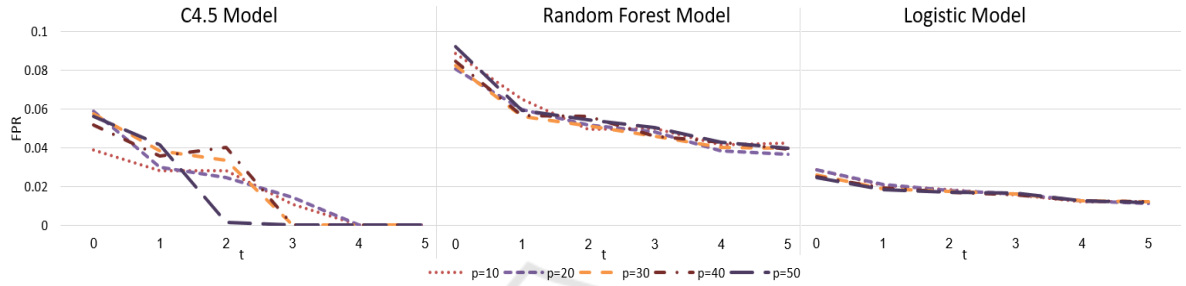

Figure 5: FPR results for three classification models.

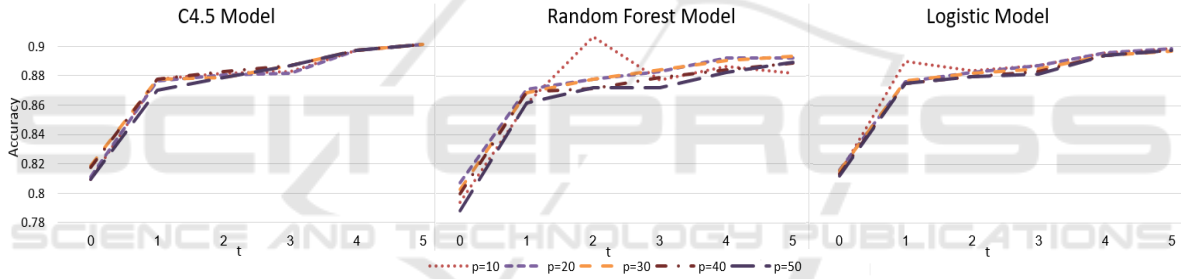

Figure 6: Accuracy results for three classification models.

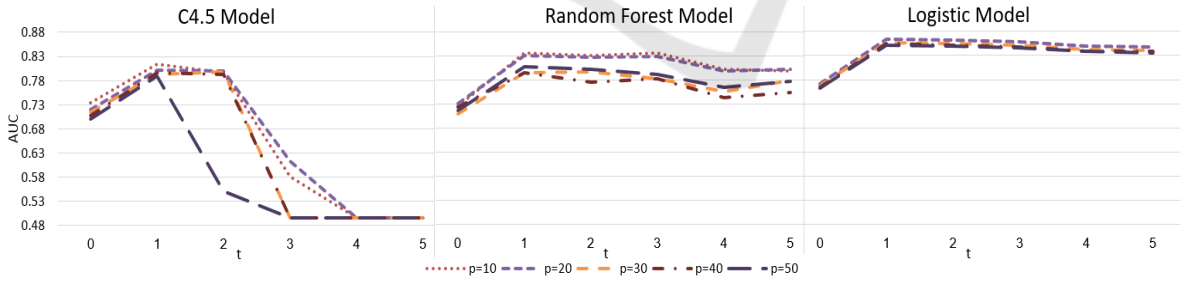

Figure 7: AUC results for three classification models.

5 CONCLUSIONS

This paper presents method for constructing effective classifiers for malicious applications in applications stores. Classification is based on a set of features that are extracted from customers textual reviews. We achieved this by defining a set of significant features and extracting them from real application store dataset. Evaluation of the classifier is performed with several machine learning algorithms. The evaluation demonstrates that our model performs well in terms of accuracy and AUC measures, as presented in Section 4. The best results were obtained using the logistic regression model which achieved accuracy rates of 89% and an AUC of 86%. However, TPR results show that it is not feasible for using our method as a

sole detection method, and further research should be performed in this aspect.

The proposed method uses static approach as a gold standard, by labeling applications as malicious using VirusTotal, as mentioned in section 3.3.3. In terms of space complexity, traditional methods analyze application files (a few dozen megabytes), while the proposed method analyze text files (a few kilobytes). In our dataset, an average application file size is 18MB, while an average application reviews text file size is 7.8KB, a 99.9% space resource decrease. Time complexity for classifying a single application in the proposed methods is O(w), where w is the number of words in all of the applications reviews.

Another contribution of this paper is that we introduced a fast classifier, which by using a generative model, can classify large-scale domains, such as the three biggest application stores: Google Play, iOS AppStore and Amazon Appstore.

Moreover, our model provides better results compared to several baseline methods for textual classification such as BOW and LDA. Such results demonstrate the ability to detect malicious applications when using cyber-security domain information, such as domain lexicon, rather than general linguistic information such as word frequency.

Our research considers malicious application detection by analyzing only end users textual reviews. A possible future research direction includes an analysis of other types of user-related features such as user reputation, diversity of reviews, etc. Examples of other feature types are user reliability and professional domain knowledge. Another future direction is detection of unknown malware, based on the reported behavior of an application.

REFERENCES

Amazon (2016). Amazon appstore. http://www.amazon.com/mobile-apps/b?node=2350149011. [Online; accessed April 2016].

Aung, Z. and Zaw, W. (2013). Permission-based android malware detection. *International Journal of Scientific and Technology Research*, 2(3):228–234.

Balahur, A., Steinberger, R., Kabadjov, M., Zavarella, V., Van Der Goot, E., Halkia, M., Pouliquen, B., and Belyaeva, J. (2013). Sentiment analysis in the news. *arXiv preprint arXiv:1309.6202*.

Baron, N. S. (2003). Language of the internet. *The Stanford handbook for language engineers*, pages 59–127.

Bishop, C. M. (2006). Pattern recognition. *Machine Learning*, 128.

Blair-Goldensohn, S., Hannan, K., McDonald, R., Neylon, T., Reis, G. A., and Reynar, J. (2008). Building a sentiment summarizer for local service reviews. In *WWW workshop on NLP in the information explosion era*, volume 14, pages 339–348.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Burguera, I., Zurutuza, U., and Nadjm-Tehrani, S. (2011). Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26. ACM.

Choi, Y. and Cardie, C. (2008). Learning with compositional semantics as structural inference for subsentential sentiment analysis. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 793–801. Association for Computational Linguistics.

Dave, K., Lawrence, S., and Pennock, D. M. (2003). Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international conference on World Wide Web*, pages 519–528. ACM.

De Marneffe, M.-C., MacCartney, B., Manning, C. D., et al. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Dunham, K. (2008). *Mobile malware attacks and defense*. Syngress.

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844.

Hong, L. and Davison, B. D. (2010). Empirical study of topic modeling in twitter. In *Proceedings of the first workshop on social media analytics*, pages 80–88. ACM.

Hu, M. and Liu, B. (2004). Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM.

IDC (2016). Apple, huawei, and xiaomi finish 2015 with above average year-over-year growth, as worldwide smartphone shipments surpass 1.4 billion for the year, according to idc. http://www.idc.com/getdoc.jsp?containerId=prUS40980416.

Katz, G., Ofek, N., and Shapira, B. (2015). Consent: Context-based sentiment analysis. *Knowledge-Based Systems*, 84:162–178.

Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145.

Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of static analysis for malware detection. In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*, pages 421–430. IEEE.

Mullen, T. and Collier, N. (2004). Sentiment analysis using support vector machines with diverse information sources. In *EMNLP*, volume 4, pages 412–418.

Nissim, N., Cohen, A., Moskovitch, R., Shabtai, A., Edry, M., Bar-Ad, O., and Elovici, Y. (2014). Alpd: Ac-

tive learning framework for enhancing the detection of malicious pdf files. In *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint*, pages 91–98. IEEE.

Ofek, N., Katz, G., Shapira, B., and Bar-Zev, Y. (2015). Sentiment analysis in transcribed utterances. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 27–38. Springer.

Ofek, N., Poria, S., Rokach, L., Cambria, E., Hussain, A., and Shabtai, A. (2016). Unsupervised commonsense knowledge enrichment for domain-specific sentiment analysis. *Cognitive Computation*, 8(3):467–477.

Ofek, N., Rokach, L., and Mitra, P. (2014). Methodology for connecting nouns to their modifying adjectives. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 271–284. Springer.

Ofek, N. and Shabtai, A. (2014). Dynamic latent expertise mining in social networks. *IEEE Internet Computing*, 18(5):20–27.

Onix (2016). Onix text retrieval toolkit. http://www.lextek.com/manuals/onix/stopwords1.html. [Online; accessed April 2016].

Oommen, T., Baise, L. G., and Vogel, R. M. (2011). Sampling bias and class imbalance in maximum-likelihood logistic regression. *Mathematical Geosciences*, 43(1):99–120.

Pang, B., Lee, L., and Vaithyanathan, S. (2002). Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics.

Portier, K., Greer, G. E., Rokach, L., Ofek, N., Wang, Y., Biyani, P., Yu, M., Banerjee, S., Zhao, K., Mitra, P., et al. (2013). Understanding topics and sentiment in an online cancer survivor community. *JNCI Monographs*, 47:195–198.

Portokalidis, G., Homburg, P., Anagnostakis, K., and Bos, H. (2010). Paranoid android: versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 347–356. ACM.

Quinlan, J. R. (1993). C4. 5: Programming for machine learning. *Morgan Kauffmann*, page 38.

Ranveer, S. and Hiray, S. (2015). Comparative analysis of feature extraction methods of malware detection. *International Journal of Computer Applications*, 120(5).

Rastogi, V., Chen, Y., and Jiang, X. (2013). Droidchameleon: evaluating android anti-malware against transformation attacks. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 329–334. ACM.

Shabtai, A., Tenenboim-Chekina, L., Mimran, D., Rokach, L., Shapira, B., and Elovici, Y. (2014). Mobile malware detection through analysis of deviations in application network behavior. *Computers & Security*, 43:1–18.

Singh, Y., Kaur, A., and Malhotra, R. (2009). Comparative analysis of regression and machine learning methods for predicting fault proneness models. *International journal of computer applications in technology*, 35(2-4):183–193.

snoopwall (2014). Summarized privacy and risk analysis of top 10 android flashlight apps. http://www.snoopwall.com/threat-reports-10-01-2014/. [Online; accessed April 2016].

Statista (2016). Number of available apps in the apple app store from july 2008 to june 2015. http://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/. [Online; accessed April 2016].

Total, V. (2016). Virustotal, a free online service that analyzes files and urls enabling the identification of viruses, worms, trojans and other kinds of malicious content. https://www.virustotal.com/. [Online; accessed April 2016].

Twitter (2016). Twitter dictionary: A guide to understanding twitter lingo. http://www.webopedia.com/quick_ref/Twitter_Dictionaryx_Guide.asp. [Online; accessed April 2016].

Šrndic, N. and Laskov, P. (2013). Detection of malicious pdf files based on hierarchical document structure. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium*.

Walker, S. H. and Duncan, D. B. (1967). Estimation of the probability of an event as a function of several independent variables. *Biometrika*, 54(1-2):167–179.

Wang, K. and Stolfo, S. J. (2004). Anomalous payload-based network intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 203–222. Springer.

WEKA (2016). http://www.cs.waikato.ac.nz/ml/weka/. [Online; accessed April 2016].

Xie, L., Zhang, X., Seifert, J.-P., and Zhu, S. (2010). pbmds: a behavior-based malware detection system for cellphone devices. In *Proceedings of the third ACM conference on Wireless network security*, pages 37–48. ACM.

Yang, Z., Yang, M., Zhang, Y., Gu, G., Ning, P., and Wang, X. S. (2013). Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1043–1054. ACM.

Ye, Q., Zhang, Z., and Law, R. (2009). Sentiment classification of online reviews to travel destinations by supervised machine learning approaches. *Expert Systems with Applications*, 36(3):6527–6535.

Zhang, Y., Yang, M., Xu, B., Yang, Z., Gu, G., Ning, P., Wang, X. S., and Zang, B. (2013). Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 611–622. ACM.

Zheng, M., Sun, M., and Lui, J. C. (2013). Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 163–171. IEEE.