

2021

Feature Selection on Permissions, Intents and APIs for Android Malware Detection

Fred Guyton

Follow this and additional works at: https://nsuworks.nova.edu/gscis_etd



Part of the [Computer Sciences Commons](#)

Share Feedback About This Item

This Dissertation is brought to you by the College of Computing and Engineering at NSUWorks. It has been accepted for inclusion in CCE Theses and Dissertations by an authorized administrator of NSUWorks. For more information, please contact nsuworks@nova.edu.

Feature Selection on Permissions, Intents and APIs
for Android Malware Detection

by

Fred Guyton

A dissertation submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy
in
Information Assurance

College of Computing and Engineering
Nova Southeastern University

2021

We hereby certify that this dissertation, submitted by Fred Guyton conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.



Wei Li, Ph.D.
Chairperson of Dissertation Committee

4/21/2021

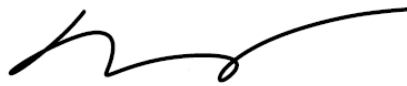
Date



Ajoy Kumar, Ph.D.
Dissertation Committee Member

04/21/2021

Date



Ling Wang, Ph.D.
Dissertation Committee Member

4/21/2021

Date

Approved:



Meline Kevorkian, Ed.D.
Dean, College of Computing and Engineering

4/21/2021

Date

College of Computing and Engineering
Nova Southeastern University

An Abstract of a Dissertation Submitted to Nova Southeastern University in Partial
Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Feature Selection on Permissions, Intents and APIs for Android Malware Detection

by
Fred Guyton
April 2021

Malicious applications pose an enormous security threat to mobile computing devices. Currently 85% of all smartphones run Android, Google's open-source operating system, making that platform the primary threat vector for malware attacks. Android is a platform that hosts roughly 99% of known malware to date, and is the focus of most research efforts in mobile malware detection due to its open source nature. One of the main tools used in this effort is supervised machine learning. While a decade of work has made a lot of progress in detection accuracy, there is an obstacle that each stream of research is forced to overcome, feature selection, i.e., determining which attributes of Android are most effective as inputs into machine learning models.

This dissertation aims to address that problem by providing the community with an exhaustive analysis of the three primary types of Android features used by researchers: Permissions, Intents and API Calls. The intent of the report is not to describe a best performing feature set or a best performing machine learning model, nor to explain why certain Permissions, Intents or API Calls get selected above others, but rather to provide a holistic methodology to help guide feature selection for Android malware detection.

The experiments used eleven different feature selection techniques covering filter methods, wrapper methods and embedded methods. Each feature selection technique was applied to seven different datasets based on the seven combinations available of Permissions, Intents and API Calls. Each of those seven datasets are from a base set of 119k Android apps. All of the result sets were then validated against three different machine learning models, Random Forest, SVM and a Neural Net, to test applicability across algorithm type.

The experiments show that using a combination of Permissions, Intents and API Calls produced higher accuracy than using any of those alone or in any other combination and that feature selection should be performed on the combined dataset, not by feature type and then combined. The data also shows that, in general, a feature set size of 200 or more attributes is required for optimal results. Finally, the feature selection methods Relief, Correlation-based Feature Selection (CFS) and Recursive Feature Elimination (RFE) using a Neural Net are not satisfactory approaches for Android malware detection work.

Based on the proposed methodology and experiments, this research provided insights into feature selection – a significant but often overlooked issue in Android malware detection. We believe the results reported herein is an important step for effective feature evaluation and selection in assisting malware detection especially for datasets with a large number of features. The methodology also has the potential to be applied to similar malware detection tasks or even in broader domains such as pattern recognition.

Acknowledgements

The journey to getting a Ph.D. is never done alone and I would like to acknowledge those people that were key to getting me through the process at NSU.

First, I would like to extend my thanks to Dr. Wei Li for serving as my dissertation committee chair. He challenged me in my coverage of the topic and provided excellent guidance as the document took shape, always finding the next spot that needed clarity and pushing me to make this dissertation the best it could be. I would also like to thank the other two members of my committee, Dr. Ling Wang and Dr. Ajoy Kumar. Their advice and guidance through this process were invaluable.

I would like to thank my loving wife Kristin for her encouragement and support, and for her patience with my obsessive focus and constant work on this over the past few years.

Finally, I would like to thank my parents, my mother Ernestine Guyton, and my father Clyde Guyton who passed away in 2018. The example they set with their work ethic is something I strive to match each day.

Table of Contents

Abstract ii

List of Tables vii

List of Figures ix

Chapters

1. Introduction 1

Background 1

Problem Statement 19

Dissertation Goal 20

Research Questions 22

Relevance and Significance 22

Assumptions, Limitations and Delimitations 23

Definition of Terms 23

List of Acronyms 29

Summary 31

2. Review of the Literature 33

Android as a Target 33

A Maturing Research Stream 37

A Static Analysis Survey 47

Survey Analysis 51

Survey Summary 54

3. Methodology 55

Notations 55

Feature Scope 56

Feature Selection Algorithms 57

Machine Learning Algorithm Selection 62

Experiment Design 63

Validation Analysis 79

Resources	82
4. Results	85
Data Description	85
Feature Subset Selection Process	86
Ranking Features	92
Data Repeatability	110
Validating Feature Subsets	113
Feature Summary	126
5. Conclusions, Implications and Recommendations	129
Conclusions	129
Implications	131
Recommendations	132
Summary	133
Data and Code Repositories	138
Appendices	
A. References for Table 4	139
B. Experiment Feature List	142
C. Heatmap Data of Feature Ranking by Dataset	149
D. Heatmap Data of Feature Ranking by Algorithm	156
E. Dataset Evaluation by Feature Selection Method and Classifier	163
F. Top 200 Permissions, Intents and API Calls	166
References	171

List of Tables

Tables

1. Caruana and Niculescu-Mizil Performance Study 13
2. Kotsiantis, Zaharakis and Pintelas Comparison 14
3. Summary of Android Features Used by Publication 19
4. Machine Learning Technique by Publication 49
5. Summary of ML Technique by Publication 51
6. Dataset Malware Sources 52
7. Dataset Sizes 53
8. Dataset Percentage Malware 53
9. Symbols 55
10. Feature Selection Algorithms Text Matrix 57
11. ML Classifiers Test Matrix 63
12. Datasets Test Matrix 66
13. Experiment Measurements 80
14. Summary of Measurement Criteria 82
15. Dataset Options 84
16. Datasets Test Matrix - Observed Feature Count 86
17. Phase 2 - Feature Count by Dataset and Feature Category. 87

18. Phase 3 - Feature Count by Dataset and Feature Category. 88
19. Feature Selection Algorithm Effectiveness 89
20. Overall Algorithm Effectiveness 90
21. Effectiveness by Feature Selection Type. 91
22. Compute Time per Feature Selection Method. 91
23. Compute Time per Feature Selection Type. 92
24. Top 10 Selected Features by Feature Selection Method 105
25. Selected Repeatability Data 111
26. Average Repeatability (Standard Deviation) 113
27. Classifier Test Matrix: Feature Subset Size by Feature Selection Method. 114
28. Maximum Accuracy Across Test Matrix 121

List of Figures

Figures

1. Android Architecture 3
2. Feature Set Evaluation Framework 20
3. Preprocessing 64
4. Binary Encoded Dataset Fragment 66
5. Feature Subset Selection Phases 67
6. Feature Subset Selection Process 68
7. Algorithmic Selection Results Example 69
8. Performance Selection Results Example 73
9. K-based Selection and Validation 79
10. Experiment Confusion Matrix. 80
11. Heatmap of Feature Ranking by Dataset 94
12. Heatmap of Feature Ranking by Feature Selection Algorithm 96
13. Combined to Single Set Membership Example - Exact Match 98
14. Combined to Single Set Membership Example - Union 99
15. Combined to Single Set Membership Example – Euclidean Distance 100
16. Similarity of Permissions: Combined Dataset vs Permissions Only Dataset 102
17. Similarity of Feature Types: Combined Dataset vs Singular Only Datasets 103

18. Similarity of Univariate Filter Methods 106
19. Similarity of Multivariate Filter Methods 107
20. Similarity of Wrapper Methods 108
21. Similarity of Embedded Methods 109
22. Similarity of All Eleven Feature Selection Methods 110
23. Data Repeatability for Feature Selection Methods 112
24. Dataset Comparison of Accuracy with Chi-Square and Random Forest 115
25. Dataset Comparison of Accuracy with RFE–NN and Random Forest 116
26. Dataset Comparison of Accuracy with Ridge Regression and Random Forest 116
27. Dataset Comparison of Accuracy with Chi-Square and SVM 117
28. Dataset Comparison of Accuracy with Chi-Square and Neural Network 117
29. Dataset Comparison of TPR with Chi-Square and Random Forest 119
30. Dataset Comparison of FPR with Chi-Square and Random Forest 119
31. Dataset Comparison of Precision with Chi-Square and Random Forest 120
32. Dataset Comparison of F-measure with Chi-Square and Random Forest 120
33. Variation in Accuracy Across Classifier Algorithm 122
34. Variation in Accuracy Across Feature Selection Methods 123
35. Percentage Decrease in Accuracy Compared to Random Forest 123
36. Curve Fit Example for Convergence Analysis 125

- 37. Convergence Data for All Feature Sets Using Dataset PIA 127
- 38. Convergence Versus Feature Set Size 128

Chapter 1

Introduction

Background

Smartphones have become ubiquitous during this past decade and have come to store more and more of their owner's personal data, going well beyond simple contact and social media data but including much more sensitive and private information such as usernames and passwords for access to financial sites as well as payment information itself such as credit card and bank account numbers. During this time the predominant mobile platforms have resolved down to two systems: Android and Apple (iOS).

Android, owned by Google, now has 85% of the world market share according to IDC, an industry leading technology research firm (Scarsella, Reith, Chau, & Shirer, 2018)

Apple's iOS has the remaining 15% market share with all other platforms being negligible.

Android

Google's mobile operating system was originally developed by a company of the same name, Android, started by Andy Rubin, a former engineer at Apple (Welcome to the definitive Android Central take on the history of Google's OS, 2015). It was built based on the Linux kernel for convenience of getting device drivers, memory management, process management, networking and security with minimal effort. Google purchased Android (the company) in 2005 and later released the first commercial device running Android in 2008, the T-Mobile G1 built by HTC, just over a year after the debut

of the first iPhone. In a step that turned out to be extremely important, Google also open-sourced it as the Android Open Source Project (AOSP).

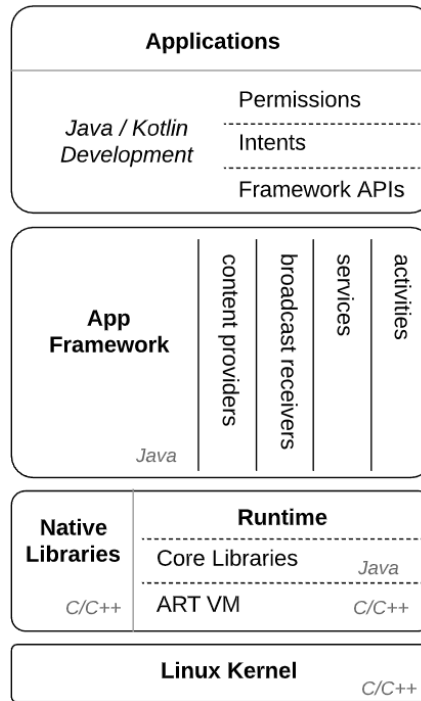
At the time, the dominant mobile phone operating system was Symbian at almost 50% market share, used by Samsung, Motorola, Sony Ericsson, Nokia and other smaller players (Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017, 2018). In second place was Blackberry OS by RIM at 20% and third place was a tie between Windows Phone by Microsoft and iOS on Apple's iPhone, both at 10%. With Android being open source, it was adopted by several manufacturers and started to experience phenomenal growth. By mid-2010 it overtook iOS and today is the dominant player at 85% share of the mobile market.

Applications are the essence of what makes smartphones "smart". Google, Apple and other third-party developers use "app stores" as convenient online sources for free and paid apps, easily accessible by users who can download and install apps on their device with no outside assistance. The official Google app store is Google Play. As of the end of the first quarter of 2020, there were over 2.9 million apps available on Google Play (Number of Android applications, 2020).

As mentioned earlier, Android is built on top of a modified Linux kernel. As shown in Figure 1 there are three additional architectural layers, *Native Libraries and Runtime* environment, the *App Framework*, and *Applications*.

The Linux kernel acts as the hardware abstraction layer and provides the basic computing infrastructure. The next layer up has two segments. The *Native Libraries* which provide key services with examples being media, audio and database management. The *Runtime* environment contains the ART VM (Android Runtime Virtual Machine)

Figure 1
Android Architecture



which is the equivalent of a JVM (Java Virtual Machine) except written specifically for Android. It runs Dex files, which are the byte code files that come from compiling Java classes and JAR files for Android. ART replaced the Dalvic VM from earlier versions of Android. Also in the Runtime environment are the core libraries which provide the basic Java package and associated utilities.

The next level up is the *App Framework* which provides all the service “managers” for applications, such as the Telephony Manager and Location Manager. These are built around four major components: *activities*, *services*, *broadcast receivers* and *content providers*.

Activities are essentially a user interface (UI). It represents a single screen, so an app may have many different activities with each being independent from the others.

Activities are launched using Intents, a concept covered in a later section.

Services are for keeping an app running in the background to perform long-running operations such as playing music while the user does something else on the device. Services do not have UIs (activities) and as with activities, services are launched using Intents.

Broadcast Receivers are used to broadcast messages to the system or other apps allowing them to react to events such as a text message being received, or the screen being turned off. Like services they have no UI and are launched using Intents.

Content Providers manage a shared set of app data (or data-store) for situations where apps want to share their data with other apps, such as Contacts being available to the email app or phone app. UIs are handled by the app making use of the content provider (Faruki, et al., 2015; Application Fundamentals, 2020).

The top layer in the Android stack (Figure 1) is that of *Applications*. Apps are written in the programming languages Java and/or Kotlin. To access the lower layers of the stack, apps use *Intents*, as discussed above, as well as *Permissions*, with all being programmed using the exposed Android APIs.

Permissions are a key part of Android often playing a significant role in malware detection. The purpose of a Permission is to protect the privacy and data of the user. Apps must request permission to access certain sensitive user data such as text messages and contacts, as well as various system components such as the camera, microphone, and network. Depending on the Permission level, Android might grant the permission automatically or might prompt the user to approve or disapprove the request.

Android categorizes all Permissions into four categories: *normal*, *dangerous*, *signature* and *special*.

Normal - Permissions that have minimal risk to the user, system or device and are granted by default at install time. They protect access to APIs that can cause no harm such as installing a shortcut.

Dangerous - Permissions considered as high risk given their capability of accessing private data and important sensors on the device. Users must accept or decline an app's ability to use a dangerous Permission as the app is being installed. These Permissions protect access to APIs that could cause harm, like those related to spending money or collecting private data. Examples would include the ability to read text messages or record audio.

Signature - Permissions that are granted only if the requesting app is signed by the same certificate authority as the app that defined the permission. Given that security check, they are granted automatically at the install time and are available with the system apps. APIs that require such permission level include accessing voicemail, NFC and VPN, among others.

Special - Permissions are system level and labeled "Not for use by 3rd party apps." Examples include the APIs granting access to system alerts and writing settings (Felt, Chin, Hanna, Song, & Wagner, 2011; Tam, Feizollah, Anuar, Salleh, & Cavallaro, 2017; Permissions Overview, 2019).

Intents are another important piece of the Android OS. An Intent is a messaging component for requesting an action from another app component. Three fundamental use cases are: starting an activity, starting a service and initiating a broadcast. They are used extensively for inter-application and intra-application communication. Android restricts

who may send certain intents to prevent apps from mimicking them (Felt, Chin, Hanna, Song, & Wagner, 2011).

There are two types of Intents: *explicit* and *implicit*.

Explicit – Intents that specify precisely which app will be used. Known as early binding, this type is typically used to start a component within the same app such as starting a new activity in response to a user action, or start a service.

Implicit – Known as late binding, it does not name a specific component but rather declares an action to perform for which a component from another app can manage. An example is an app that wants to display a location on a map might use an implicit Intent to request that some other app capable of performing that function show the specified location on a map (Intents and Intent Filters, 2019).

Android's tremendous marketshare and open architecture make it a popular target for malicious apps. Security firm AV-Test reports that over the last three years there was an average of 5.9 million new malicious apps per year specifically targeting Android (AV-Test Malware Statistics, 2019). The malware covers a large range of nefarious processes aimed at mobile devices, from stealing users' most sensitive information to one of the most popular today - running botnets for crypto-currency mining (Samani & Davis, 2019).

The popularity also means that Google Play, the app store, is a key threat vector. Google took steps to address the problem in 2012 by deploying to Google Play a scanning tool called Bouncer to help detect malware, but according to Hou (2012) there was minimal impact due to the limited scope of the scan. Five years later in 2017, Google rebranded Bouncer under the name "Play Protect" and included on-device

protection (Amadeo, 2017) but according to an analysis by Computerworld, all the services announced as part of Play Protect were already part of Play Store and Android for some time past (Raphael, 2017), so once again there was minimal impact. A recent comparison of different antivirus products on Android by an independent test lab rated Play Protect last among twenty products tested (The best antivirus software for Android, 2018).

Threat Detection

In classic intrusion detection as well as mobile malware detection, approaches are classified as either *signature-based* or *anomaly-based*. As described by Garcia-Teodoro, Diaz-Verdejo, Maciá-Fernández and Vázquez (2009), signature-based systems look for defined patterns, or signatures within the app and compare to a signature database of known attacks. Anomaly-based systems attempt to model the “normal” behavior of a system and generate an anomaly alert whenever the difference between an observation and the normal behavior exceeds some predefined threshold.

Within the world of anomaly-based detection, another stratification of techniques relates to how the analysis is accomplished. Detection of malware happens by either examining its code or by executing it in a safe environment (Gandotra, Bansal, & Sofat, 2014). The former is known as *static analysis*. With static analysis, the executable app (in Android referred to as an APK file) first has to be decompiled into source code and that code is analyzed for patterns that indicate malware or not. The latter is known as *dynamic analysis*. Dynamic analysis occurs with the app running and monitors things like information flow, function calls, etc., with the goal of observing what the app does and detecting if it is malware or not based on its actions.

Garcia-Teodoro, et al., further categorize anomaly-based systems as either: 1) statistical-based, which is focused on stochastic behavior, 2) knowledge-based, which requires availability of prior knowledge and/or data, or 3) machine learning-based, which is essentially a categorization of patterns. The third one is of interest here and is explored in more detail in the next section.

Machine Learning

Machine learning techniques can be mapped to three primary scenarios: 1) situations where examples of data are available that provide the input and the output, 2) situations where only the input data is available and 3) situations where only the input data is available but there is a feedback loop indicating the quality of the prediction (Lison, 2015). The three cases are generally referred to as: supervised learning, unsupervised learning and reinforcement learning, respectively.

Considering the problem of malware detection with respect to these three, reinforcement learning has yet to be shown as an effective tool given the nature of the life cycle of malware. “Reinforcement” would require someone identifying that they had installed malware and feeding that back into the detection engine, which implies the event to be prevented, installing malware, had already occurred and that the user would know that and report it. Interestingly it is being used to attack malware detection as described by Anderson, Kharkar, Filar, Evans and Roth (2018), and Fang, Wang, Li, Wu, Zhou and Huang (2019), where they demonstrate the use of reinforcement learning against dynamic analysis and static analysis respectively.

Unsupervised learning has low applicability as a stand-alone approach to malware detection due to the need for positive identification of malware as well as the need for

low false positives. However, it is often used in feature engineering where the goal is to understand the structure of various features in an application (Machine Learning for Malware Detection, 2019). This understanding is then applied to feature selection to be used in supervised models such as demonstrated by Verma and Muttoo (2016) who used the unsupervised technique of K-means clustering to limit the number of features to be analyzed by a supervised approach using Decision Trees.

Supervised learning as described by Shalev-Shwartz and Ben-David (2014) is "using experience to gain expertise," by having a training example that has key information that is missing from other test data, or data of interest in the wild. This key information, typically referred to as labels on the data, is used to teach or supervise the learner by providing the answers to how the model should predict based on input. The model is then applied to unlabeled data to make predictions without having the answers a priori.

In supervised learning, target predictions can be characterized as either quantitative or qualitative, where the quantitative variables have numerical values and qualitative variables have values in one of n different classes, or categories (James, Witten, Hastie, & Tibshirani, 2017). These two approaches are generally referred to as *regression* versus *classification* respectively. Regression approaches aim to predict values along a continuous output variable. Examples include a stock price, the value of a house, and a person's income. Classification approaches aim to predict values of discrete output variables. Examples of classification variables include email type (spam or not spam), whether a person will default on a loan (yes or no), whether a patient has cancer (yes or no), and of most importance here, whether an application is malware or benign.

There are a number of approaches, or algorithms, for supervised machine learning. Following is a list and brief description of the supervised machine learning classifiers that are most widely used today. (James, Witten, Hastie, & Tibshirani, 2017; Shalev-Shwartz & Ben-David, 2014)

Support Vector Machine (SVM): A linear method of classification in which data is defined as points in an n -dimensional space (where n is the number of features). The model defines a hyperplane that optimally separates (classifies) all of the data points onto one side or the other of the hyperplane.

Bayesian Networks: An algorithmic applications of Bayes theorem resulting in a probabilistic graphical model representing the relationships between several random variables, or features.

Naïve Bayes: Another probabilistic classifier based on Bayes theorem but with a significant restriction. The “naïve” moniker is due to the fact that it assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Decision Trees: Classification models in the form of a tree structure where data points are broken down into smaller and smaller subsets as an associated decision tree is incrementally developed as an “if-then” rule set. The resulting structure has *branches* and *leaves* where branches represent a decision node and leaves represent a classification node.

K-nearest Neighbor: An instance-based algorithm in which data is defined as points in an n -dimensional space. When a new data point arrives, the closest k

number of instances (nearest neighbors) to that point are analyzed and the most common class of those are the prediction or classification of the new data point.

Logistic Regression: Regardless of the regression term in the name, it is a classification algorithm. It transforms n features using the logistic function, most often using a sigmoid curve, to a probability output from 0 to 1 which can then be interpolated into binary results.

Artificial Neural Networks (ANN): Models that are inspired by the structure of biological neural networks. They consist of a set of connected input/output units in two or more layers where each connection has a weight associated with it that gets tuned in the training phase to adapt the network to the particular problem at hand.

Ensemble methods are machine learning techniques that combines several models such as just described in order to produce an even better predictive model. The following four techniques are ensemble methods used in conjunction with one or more of the previously described algorithms.

Random Forest: An ensemble technique for supervised learning classification that works by constructing a multitude of decision trees. The final classification is the mode of the classes (classification) of the individual trees.

Adaptive Boosting: Also known as AdaBoost, it is used with many different machine learning algorithms but typically with only one type per model. It is a sequential ensemble method where the output of those algorithms, referred to as weak learners, are combined into a weighted sum that represents the final output of the boosted classifier.

Bootstrap Aggregation: Also known as Bagging, this technique can also be used with many different machine learning algorithms but typically with only one type per model. It is a parallel ensemble method that uses bootstrap sampling of the dataset to construct many independent models using the weak learners and aggregating those results into a final classification.

Stacking: An ensemble method that uses multiple different machine learning algorithms in a single model. The various algorithms use the same dataset, and their output is fed into yet another machine learning algorithm as a meta-model. This meta-model will take the outputs of the weak learners as inputs and develop a final classification.

When trying to decide between machine learning approaches (algorithms) one obvious candidate for including in the selection criteria is performance. Caruana and Niculescu-Mizil (2006) performed a large-scale empirical comparison of several supervised learning algorithms using eight performance criteria. Comparing machine learning techniques in general can be problematic given the tendency for some techniques to perform better in certain problem domains, so this study used eleven different classification problems and associated datasets. The datasets were diverse including census data, medical imaging, particle physics, text recognition and biological data.

The machine learning techniques under study were Decision Trees, SVMs, Logistic regression, Naïve Bayes, ANN and K-nearest Neighbor along with three ensemble methods on Decision Trees: Random Forests, Bagged Decision Trees and Boosted Decision Trees. The performance criteria included threshold metrics, ordering/rank metrics and probability metrics. The first group included accuracy, F-score

and lift. The second included area under the ROC curve (Receiver operating characteristic), average precision, and precision/recall breakeven point. The probability metrics were squared error (Root Mean Square) and cross-entropy.

A summary of the results of their testing is presented in Table 1. It is interesting to note that for all performance metrics, which includes all the problem domains under study, the three ensemble methods dominated the top three positions. Considering non-ensemble techniques, SVMs and Neural Nets performed best, again across all performance metrics. Conversely, the lowest performers were consistently Logistic Regression and Naïve Bayes.

Kotsiantis, Zaharakis and Pintelas (2007) analyzed a number of supervised learning techniques that also included Decision Trees, SVMs, Naïve Bayes, ANN and K-nearest Neighbor, but no ensemble methods. A normalized summary of their data is presented in Table 2. This study has more qualitative commentary and performance criteria related to speed compared to the prior study, but the first column does address accuracy. In that metric, SVMs and Neural Nets are cited as the top performers, which directly coincides with the results of the Caruana and Niculescu-Mizil work when not

Table 1
Caruana and Niculescu-Mizil Performance Study

Algorithm	Accuracy	F-score	Lift	Area under Curve	Avg Precision	Precision/Recall Break Even Point	Root Mean Square	Cross-entropy
Random Forest	0.87	0.79	0.93	0.96	0.93	0.93	0.83	0.83
Bagged DT	0.85	0.78	0.94	0.96	0.94	0.92	0.85	0.87
Boosted DT	0.83	0.82	0.94	0.96	0.94	0.93	0.60	0.61
SVM	0.82	0.80	0.90	0.94	0.90	0.91	0.51	0.47
ANN	0.80	0.76	0.91	0.94	0.89	0.90	0.81	0.82
KNN	0.76	0.73	0.89	0.92	0.87	0.87	0.73	0.72
Decision Tree (DT)	0.65	0.64	0.82	0.84	0.76	0.78	0.56	0.61
Logistic Reg.	0.64	0.55	0.82	0.85	0.74	0.73	0.62	0.65
Naïve Bayes	0.50	0.56	0.78	0.83	0.74	0.74	0.35	-0.63

Table 2
Kotsiantis, Zaharakis and Pintelas Comparison

Algorithm	Accuracy in general	Speed of learning	Speed of classification	Tolerance to irrelevant attributes	Tolerance to redundant attributes	Tolerance to highly interdependent attributes	Dealing with danger of overfitting	Model parameter handling
SVM	1.00	0.25	1.00	1.00	0.75	0.75	0.50	0.25
ANN	0.75	0.25	1.00	0.25	0.50	0.75	0.25	0.25
Decision Trees	0.50	0.75	1.00	0.75	0.50	0.50	0.50	0.75
KNN	0.50	1.00	0.25	0.50	0.50	0.25	0.75	0.75
Naïve Bayes	0.25	1.00	1.00	0.50	0.25	0.25	0.75	1.00

considering ensemble techniques. Both studies also have Naïve Bayes as the lowest performer.

Feature Selection

In machine learning, features are individual independent measurable properties or characteristics that act as input into the model. For example, a machine learning model used to predict the probability of rain might have features such as the outside temperature, wind speed and humidity. Complex models can contain large numbers of input features, hundreds and even thousands.

Feature selection is the process of selecting a subset of relevant features for use in construction of machine learning models. In their seminal work on the subject, Guyon and Elisseeff (2003) described three reasons to focus on improving feature selection: 1) higher model accuracy, 2) faster model performance and 3) better understanding of the model itself. They went on to suggest that researchers with sufficient time and computational resources should “compare several feature selection methods, including [...] new idea[s]” and approaches.

According to Liu, et al. (2005), the goals of feature selection in machine learning are:

- Reducing dimensionality,

- Removing irrelevant and redundant features,
- Reducing the amount of data needed for learning,
- Improving algorithms' predictive accuracy, and
- Increasing the constructed models' comprehensibility

There are a number of approaches to feature selection. One that can be best described as *exhaustive search* is exemplified by the work of Sung and Mukkamala (2003) in which to select the most effective features of the classic KDD dataset (Lippmann, Haines, Fried, Korba, & Das, 2000) they removed variables one-by-one and reran their models each time determining the accuracy based on all features minus the one removed. This technique inherently assumes complete independence of attributes.

Another approach is to use *heuristics*, such as that by Li, Ge and Dai (2015) where they relied on the Android categorization of Permissions, using only the ones labeled as “Dangerous Permissions” (Permissions Overview, 2019). Using heuristics requires domain knowledge. In this instance, the authors understand how Google labels groups of permissions and decided that based on their domain knowledge the use of that subgroup of Permissions was appropriate.

A third, and the most common approach to feature selection, is called *filter methods*, which use a statistical measure to assign a score to each feature and who's ranking ultimately determines if they are used or not. The methods are often univariate and consider the features independently. As examples, Chan and Song (2014) used *information gain theory* in their feature selection of Android Permissions and API calls. Tsang, Kwong and Wang (2007) performed a comparison of analysis techniques including *information gain theory*, *gain ratio*, *Chi-square* and *Relief-F*.

A fourth technique is called *wrapper methods* which treat feature selection as a search problem. Some predictive model is used to evaluate a combination of features and assign a score based on model accuracy. Various search techniques can be employed. As an example, Wang, Wang, Feng, Liu, Han and Zhang (2014) used *forward selection* which is an iterative approach with each iteration adding the feature which best improves the model.

A feature selection approach referred to as *embedded methods* attempts to learn which features best contribute to the accuracy of the model while the model is being created. Of these approaches, the most common are regularization methods that introduce additional constraints into the optimization of a predictive algorithm in order to bias the model towards lower complexity. Nezhadkamali, Soltani and Seno (2017) used one of the more popular regularization algorithms, LASSO or L1 regularization, in their comparison with various filter methods.

A variation on feature selection is often referred to as feature learning. The goal with this approach is to allow a system to automatically discover what feature representations are needed for accurate classification and more specifically some number of representations that is significantly less than the original number of features in the raw data. One of the more popular techniques for this dimensionality reduction approach is *Principle Component Analysis (PCA)*. PCA is an unsupervised algorithm that creates linear combinations of the original features ranked in order of variance allowing a subset, amounting to the most important, to be selected thereby reducing the number of features. With respect to malware detection, Zhao, Fang and Wang (2014) used PCA with a

feature set of Android Permissions to define the attributes for input into a Support Vector Machine as the primary classifier.

K-means clustering is another popular approach for dimensionality reduction. This unsupervised technique tries to find k -clusters of the data by minimizing the square error function. The new attributes are then represented by the centroids of the clusters. K-means can be computationally expensive, so in an interesting variation, Napoleon and Pavalakodi (2011) used PCA on their high-dimensional dataset first and then applied K-means clustering for further reductions.

An approach that has gained significantly in popularity of late is the use of Neural Networks. This is often referred to as *deep learning*, but some use the term deep learning simply to imply the use of multi-layer, deep, Neural Networks.

There are a number of variations in the use of Neural Networks for dimensionality reduction. In one such, with respect to Android malware detection, Su, Zhang, Li and Zhao (2016) used a Neural Network based on a Deep Belief Network (DBN) for feature learning against Permissions and API calls with the selected features then used for input into a Support Vector Machine for classification. They reported improved results to other published ML techniques but had no direct comparison. Nix and Zhang (2017) also used a Neural Net, specifically a Deep Neural Network (DNN) to automatically learn features on Android APKs. They restricted their feature types to API calls only and on sequences thereof as opposed to existence. Results were compared to the use of SVM and Naïve Bayes on the same dataset and showed a higher accuracy.

Duc and Giang (2018) used a Neural Network model based on a Multilayer Perceptron (MLP) to learn features from Permissions, Intents (Intents and Intent Filters,

2019), API calls, app components, hardware used and URLs. The results, compared to prior published results on the same dataset using an SVM, were mixed showing a higher precision but a lower recall. In a similar study Wang, Zhao and Wang (2018) used variations of a Convolution Neural Network (CNN) for feature learning also including the feature types: Permissions, Intents, API calls and hardware used. Their results were compared to the same dataset run on an SVM, Decision Tree, Random Forest and K-nearest Neighbor and showed an increase in accuracy.

Android Features

Feature selection starts by defining what set or type of features will be used in the model. For example, with respect to Android malware detection using static analysis, the input could be limited to just Android Permissions with the idea being that the selection process will determine, of all the Permissions available in the Android OS, which ones should be used in the machine learning classifier.

A survey was conducted by this author on research related to Android malware detection using static analysis and machine learning. Over the 71 publications reviewed, there was a total of eleven different Android features used, some using just a single feature type, others using multiple. Table 3 shows a metadata analysis of features used. By far, the most common attribute type used are Permissions, with 82% of the publications describing their use. Second is API calls with just over 50% including that feature, and third are Intents at 24%. The remaining eight features are used much more sparsely as shown in the table.

It is also interesting to note that most researchers do not detail the features used. There are currently a total of 158 Permissions in the Android OS so it would be of value

Table 3
Summary of Android Features Used by Publication

Android Feature	# Papers Using Feature	% Papers Using Feature	% in Use
Permissions	58	82%	41%
API Calls	36	51%	25%
Intents	17	24%	12%
Hardware	8	11%	6%
App Components	7	10%	5%
Control Flow Graph	6	8%	4%
URLs	5	7%	4%
Ops Codes	2	3%	1%
Call logs	1	1%	1%
Data flow	1	1%	1%
Directories Accessed	1	1%	1%

to know precisely which ones are important. The same can be said for other features such as Intents of which there are 295 Intents defined in Android, and there are hundreds of variations on API calls given the hierarchical nature of API definitions.

Problem Statement

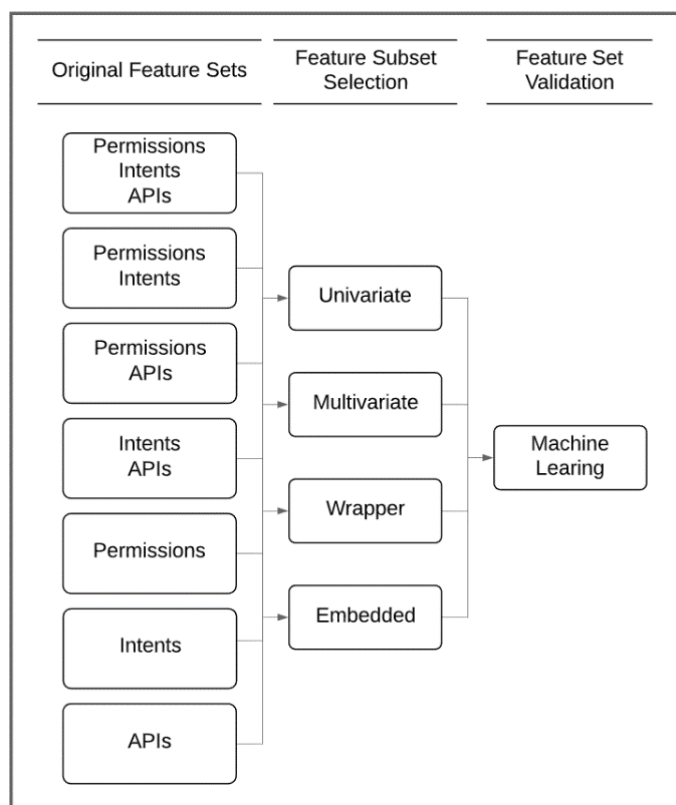
This variety in approaches to static analysis points to a problem in the academic and commercial arena of Android malware detection; there is a lack of agreement regarding what Android feature set is most effective for detecting zero-day attacks with machine learning techniques. Not only is there significant disagreement on the categories of Android features that are effective (Permissions, API Calls, etc.) but there are hundreds of discrete attributes in those categories and there is no consistency in which discrete attributes of the categories are effective. Using Permissions as an example, there are 158 different Permissions defined in Android. Some researchers just use the subset that Google classifies as "dangerous". In other cases, the Permissions subset was hand

selected based on the researchers' domain knowledge. And often, the dataset in question is simply run through some selection method such as Information Gain, and results are blindly accepted as the most important subset to apply. In large part this inconsistency in approach is due to the fact that there is no definitive study to provide guidance to researchers who desire to get past the task of feature selection and work on innovative next-step approaches in detection.

Dissertation Goal

The goal of this research is to advance the state of the industry's knowledge on feature sets used for Android static analysis malware detection. Figure 2 presents the evaluation framework of the approach. The column on the left labeled as *Original*

Figure 2
Feature Set Evaluation Framework



Feature Sets represents all possible combinations of the three groups of features under evaluation: Permissions, Intents and APIs. Given that the research community uses various combinations of these, as demonstrated earlier, then using each combination as a starting feature provides the broadest possible domain coverage.

The second column, labeled *Feature Subset Selection*, depicts the categories of feature selection techniques used: two filter method approaches, univariate and multivariate, wrapper methods and embedded methods. Each category represents one or more algorithms employed to create a feature subset for each of the original feature sets from the first column.

This provides a broad set of top-level feature subsets representing every possible combination of the original feature sets and the feature selection techniques. In terms of concrete numbers, there are seven original feature sets and eleven feature selection algorithms across the four defined categories which equals 77 different feature subsets. Within these top-level subsets, feature importance is determined by the weights assigned by each algorithm in addition to comparison to random columns of data included in each dataset. The point of this latter aspect is that regardless of a feature's reported weight, if it cannot predict better than features consisting of random data, it should be discarded. Finally, subsets of the important features (top five, top ten, etc.) are created to look at the effect of feature count. As detailed later, this resulted in 615 unique feature subsets.

The third column, *Feature Set Validation*, is where each of the 615 feature subsets are used as input into three different different machine learning algorithms. Thus, there are 1,845 total test cases evaluated and analyzed.

This approach significantly adds to a very limited amount of feature selection knowledge in the Android space by providing a robust analysis of the key feature types and providing detailed result sets naming the specific attributes that prove to be most relevant for the various machine learning algorithms.

Note that it was not a goal of the research to find a best performing feature set or a best performing machine learning model, nor to explain why certain Permissions, Intents or API Calls get selected above others.

Research Questions

1. How does feature ranking vary when Permissions, Intents and API Calls are selected separately versus combined?
2. How does feature ranking vary across feature selection algorithms?
3. How does machine learning model accuracy vary across machine learning algorithms and feature selection algorithms?
4. How does feature set size affect model accuracy across feature selection methods?
5. Among Permissions, Intents and API Calls, what are the important features?

Relevance and Significance

As discussed in the section Feature Selection, the importance of selecting the most appropriate features for machine learning algorithms cannot be overstated. To date, no one has taken a systematic approach to evaluation of features in the Android malware detection arena. This research is the first.

The shear variation in approach by the research community as documented previously in Android Features and more specifically Table 3 shows the depth of the

problem. In addition, most feature selection studies publish the results strictly in terms of malware detection capability, not the actual list of features, just the category used. This research not only publishes the performance results, but the details of which features were selected by the various methods providing a significant reference to future researchers.

Assumptions, Limitations and Delimitations

1. The scope of the investigation is limited to Android Permissions, Intents and API Calls.
2. The scope of the investigation is limited to eleven feature selection methods (detailed in the Methodology section).
3. The scope of the investigation is limited to three machine learning algorithms (detailed in the Methodology section).

Definition of Terms

Accuracy: The fraction of predictions a machine learning model predicts correctly.

Android application package (APK): The package file format used by the Android operating system for distribution and installation of applications.

Application Programming Interface (API): A set of definitions and protocols for integrating software components or applications.

Area Under the Curve (AUC): Measures the quality of a machine learning model's predictions irrespective of the classification threshold, with the curve being the *Receiver Operator Curve (ROC)*.

Artificial Neural Network (ANN): Models that are inspired by the structure of biological neural networks consisting of a set of connected input/output units in two or more layers where each connection has a weight associated with it that gets tuned in the training phase to adapt the network to the particular problem at hand.

Classification and Regression Tree (CART): A decision tree that can perform classification and regression.

Classifier: A machine learning model that is trained to classify its input into n number of distinct classes.

Control Flow Graph (CFG): A graphical representation of all execution paths possible for a running software application.

Convolutional Neural Network (CNN): A Neural Network algorithm which can input an image, assign importance to various aspects in the image and be able to differentiate one from another.

Correlation-based Feature Selection (CFS): A feature selection algorithm that takes a basis set of feature correlations and compares inter-feature correlation and each features' correlation with the class label vector to select a feature subset.

Crossover: An operator in evolutionary computing, inspired by the concept of sexual reproduction in which two genomes combine traits to produce children containing a mixture of both sets of traits.

Dataset: For supervised learning, it is a collection of input values (X) and the labeled output values (y). The dataset is typically divided into training and validation subsets.

Decision Tree (DT): A machine learning model in the form of a tree structure where data points are broken down into smaller and smaller subsets as an associated decision tree is incrementally developed as an “if-then” rule set. The resulting structure has branches and leaves where branches represent a decision node and leaves represent a classification node.

Deep Belief Network (DBN): An unsupervised Neural Network that uses probabilities to produce outputs. They consist of multiple layers of latent variables, with connections between the layers but not between components within each layer.

Deep Learning: Neural network architectures that uses multiple layers (three or more) to extract higher level features from the initial input data.

Deep Neural Network (DNN): A neural network with three or more layers.

Evolving Clustering Method (ECM): A machine learning distance-based clustering method for dynamic clustering of an input stream of data in a single pass.

F-measure: Also known as F1 score or F-score is the harmonic mean of *Recall* and *Precision*.

False Negative (FN): In the case of malware detection, an outcome where a malicious app (positive) is classified as benign (negative).

False Positive (FP): In the case of malware detection, an outcome where a benign app (negative) is classified as malicious (positive).

False Positive Rate (FPR): The proportion of actual negative outcomes that a machine learning model predicts incorrectly.

Feature Engineering: The technique of creating new features from the original features by applying one or more mathematical transformations.

Feature Importance: A mathematical representation of the importance of an individual feature in a machine learning model relative to the other features.

Feature Selection: The process of choosing the most important features of a machine learning model and creating a feature subset.

Feature Vector: The set of inputs to a machine learning model expressed as a vector.

Feature: An individual value from the feature vector.

Fuzzy C-means Method (FCM): A machine learning clustering algorithm that allows one piece of data to belong to two or more clusters.

Genetic Algorithm (GA): An evolutionary computing model inspired by biological systems, specifically evolution. It develops a group of possible solutions to a classification problem and evolves them using a fitness function until an optimized solution resolves.

KDD dataset: A network intrusion dataset from the 1999 KDD Cup annual competition hosted by the Association for Computing Machinery (ACM).

K-nearest Neighbor (KNN): An instance-based algorithm in which data is defined as points in an n -dimensional space. When a new data point arrives, the closest k number of instances (nearest neighbors) to that point are analyzed and the most common class of those are the prediction or classification of the new data point.

Latent Semantic Indexing (LSI): A technique in *Natural Language Processing (NLP)* analyzing relationships between documents and the strings they contain.

Layer: A collection of related neurons in a neural network such as the input layer or output layer, or some layer in between.

Least Absolute Shrinkage and Selection Operator (LASSO): A regression technique that performs L1 regularization, meaning it penalizes the L1 norm of the feature weights which ultimately will force some of the weights to zero.

Multilayer Perceptron (MLP): An *Artificial Neural Network (ANN)* that has an input layer, an output layer, and one or more hidden layers between.

Mutation: An operator in evolutionary computing, where during the crossover process some random mutation of the traits from the parents occurs in the child, mimicking what happens in human reproduction.

Natural Language Processing (NLP): Machine learning algorithms designed to understand human language.

Nearfield Communication (NFC): A set of networking standards used to establish communication between mobile devices in very close proximity.

Partial Decision Trees (PART): A *Decision Tree* that contains branches to undefined sub trees.

Particle Swarm Optimization (PSO): A population-based stochastic optimization technique inspired by intelligent collective behavior of animals such as flocks of birds or schools of fish.

Precision: The proportion of positive predictions from a machine learning model that were actually correct.

Preprocessing: The process that prepares a dataset to be ready as input for a machine learning model.

Principal Component Analysis (PCA): An unsupervised algorithm that creates linear combinations of the original features ranked in order of variance allowing a

subset, amounting to the most important, to be selected thereby reducing the number of features.

Random Forest (RF): An ensemble technique for supervised learning classification that works by constructing a multitude of *Decision Trees*. The final classification is the mode of the classes (classification) of the individual trees.

Recall: Also known as True Positive Rate (TPR) is the proportion of actual positive outcomes that a machine learning model predicts correctly.

Recursive Feature Elimination (RFE): An iterative procedure using backward feature elimination incorporating a classifier for ranking the features in each iteration.

Receiver Operating Curve (ROC): A curve plotting *Recall* against *False Positive Rate (FPR)* for various thresholds.

Regression: A machine learning model that predicts values along a continuous output variable.

Selection: In evolutionary computing, the process that chooses fit individuals for creating the next generation through evolutionary operations such as crossover and mutation.

Singular Value Decomposition (SVD): A matrix decomposition method used for feature dimensionality reduction.

Support Vector Machine (SVM): A linear method of classification in which data is defined as points in an n -dimensional space (where n is the number of features). The model defines a hyperplane that optimally separates (classifies) all of the data points onto one side or the other of the hyperplane.

True Negative (TN): In the case of malware detection, an outcome where a benign app (negative) is correctly predicted as benign (negative).

True Positive (TP): In the case of malware detection, an outcome where a malicious app (positive) is correctly predicted as malicious (positive).

Virtual Machine (VM): A software emulation of a physical computer on an actual physical computer.

Virtual Private Network (VPN): A networking technology that encrypts communication over an unsecured public network so that it can be used as if it were a secure private network.

List of Acronyms

ANN: Artificial Neural Network

API: Application Programming Interface

APK: Android application package

ART: Android Runtime

AUC: Area Under the Curve

CART: Classification and Regression Tree

CFG: Control Flow Graph

CFS: Correlation-based Feature Selection

CNN: Convolutional Neural Network

DBN: Deep Belief Network

DNN: Deep Neural Network

DT: Decision Tree

ECM: Evolving Clustering Method

FCM: Fuzzy C-means Method

FN: False Negative

FP: False Positive

FPR: False Positive Rate

GA: Genetic Algorithm

GUI: Graphical User Interface

KNN: K-nearest Neighbor

LASSO: Least Absolute Shrinkage and Selection Operator

LSI: Latent Semantic Indexing

ML: Machine learning

MLP: Multilayer Perceptron

NFC: Nearfield Communication

PART: Partial Decision Trees

PCA: Principle Component Analysis

PSO: Particle Swarm Optimization

RAM: Random Access Memory

RF: Random Forest

RFE: Recursive Feature Elimination

ROC: Receiver Operating Curve

OS: Operating System

SVD: Singular Value Decomposition

SVM: Support Vector Machine

TB: Terabyte

TN: True Negative

TP: True Positive

TPR: True Positive Rate

UI: User Interface

VM: Virtual Machine

VPN: Virtual Private Network

Summary

As smartphones become more and more integral to peoples' daily lives, both personal and business, the need to secure those systems grows. With 85% of the worlds' mobile phones running Google's Android operating system, that OS has become the key threat vector for bad actors trying to compromise those systems. There is a rich history of research in threat detection on computers dating back into the 1970s, but smartphones are a relatively new phenomena and Android itself was only released in late 2008, so

there has only been a decade of research on Android protection. In addition to the relative newness of Android, Google continues to modify it, ostensibly for improvements in user experience and security, and threat actors continue to improve their methods of defeating its security. It is an arms race.

The ultimate goal in threat detection is identifying and stopping zero-day attacks. There are a number of research streams towards that end in the Android community, and one such approach is using static analysis and machine learning. The concept is to analyze the source code of an Android app and using some machine learning algorithm, make the prediction as to if that app is benign or malicious. Those predictions need to be highly accurate, not giving off too many false alarms; otherwise in a real-life situation it would tend to get ignored by the user.

A key aspect of any machine learning model is the input data, or features. In the decade of work in Android security, many different feature types and specific features have been used by researchers, and the variation in feature use continues today. The most used feature types are components of the OS known as Permissions, Intents, and API calls, used individually or in various combinations. But those are feature types. There are actually 158 different Permissions, 295 different Intents and several hundred API calls depending on how they are grouped.

This study focuses on providing insight into feature importance for those three feature types and hundreds of specific features. It was accomplished by testing all combinations of Permissions, Intents and API calls, using various feature ranking and subset selection techniques to determine the most important ones, and then validating which of those result sets work best using various machine learning models.

Chapter 2

Review of the Literature

Android as a Target

Android first appeared on the market in September 2008. Early on there was little interest from threat actors due to the minimal volume of targets. The first reported malware came two years later in August 2010 when Kaspersky Labs reported the discovery of the first SMS Trojan for Android, called “FakePlayer” and Symantec reported finding location spyware consisting of a modified version of the classic “snake” video game. (Castillo, 2011)

Even though the first threats in the wild were not detected until the third quarter of 2010, security researchers started investigating Android as soon as it was released. In one early work Enck, et al. (2010) reported on a dynamic analysis tool they created called TaintDroid. It used a variation of information flow analysis called taint analysis where “tainted data” is injected into specified data flows in the running app, such as GPS location and contacts, then traced and analyzed it to determine if the app exposed private user information.

Burguera, Zurutuza, and Nadjm-Tehrani (2011) reported on their crowd-source-based malware detection app and system named Crowdroid. This was an app that users ran on their phone and fed non-personal but app behavior data (system calls) over the Internet to central servers. The servers would perform the malware analysis using a K-means clustering algorithm and report results back to the user(s). While the authors did have the system running and reported some success in malware detection, the main

contribution was this early demonstration of using machine learning in Android threat detection.

Zhao, Zhang, Ge and Yuan (2012) created another tool using dynamic analysis they referred to as RobotDroid which incorporated a Support Vector Machine (SVM) for detection using system calls as the feature vector. There was limited testing performed but for those tests a detection rate of 90% to 93% was reported with a false positive rate of 3% to 5%.

In an assessment approach by Sahs and Khan (2012), their goal was to determine the effectiveness of using control flow graphs (CFGs) and Permissions via static analysis based on a Support Vector Machine as the classifier. The system was tested on a dataset of 2,272 apps of which 4% were malicious. The authors' determination was that Permissions and CFGs appeared to be a correct approach in terms of input features for the SVM, but that the CFGs needed to be more detailed in order to improve the systems' accuracy.

In one early use of a large dataset consisting of over 200,000 apps, Zhou, Wang, Zhou and Jiang (2012) created a dual detection engine where in one side, they defined known malware signatures based on Permissions, and in the other used dynamic analysis looking for behaviors they considered likely to be used by malware, such as downloading code from the Internet to run. The accuracy based on testing was high but the percentage of malware in the dataset was very small, less than one percent, so the results are unclear.

Wu, Mao, Wei, Lee and Wu (2012) developed a host-based tool they named Droidmat to perform static analysis using Permissions, Intents, API calls and component (Activity, Service, Broadcast Receiver). It used K-means clustering as well as

Expectation Maximization (EM) clustering for feature selection and then K-nearest-neighbor for classification. An accuracy of 98% was reported on a test database of just over 1,700 apps with 13% malware.

Xu, Zhang and Zhu (2013) created an assessment tool they referred to as Permlyzer to be a general-purpose Permission analysis framework. The system used dynamic analysis to create a map, or call stack, that provided fine-grained information on Permissions use at runtime. Permlyzer would actually takes control of the app (requiring modification to the base Android operating system) and execute all possible code paths. Their statistical findings based on analyzing over 100K apps showed significant overlap between Permission requests from malicious apps versus many non-malicious apps that use common third-party libraries indicating that using Permission only as features for malware detection could be problematic.

Peiravian and Zhu (2013) investigated the using Permissions and API calls for Android malware detection. Of those features, the authors created custom selection lists for attribute input to the classifiers. They used three different machine learning approaches using the Weka library suite (Weka 3: Data Mining Software in Java, 2018). The three classifiers were Support Vector Machines, Decision Trees and Bagging Predictor. The dataset contained 2,510 apps of which about 50% were malicious. Comparing the three approaches across multiple scenarios, Bagging was the top performer followed by Support Vector Machines and then Decision Trees with accuracy averaging around 95%.

In research similar to above described, Huang, Tsai and Hsu (2013) investigated the effectiveness of multiple machine learning algorithms also using the Weka library.

The key differences were: 1) Huang, et al., only used Permissions (all Permissions) and did not include API calls, 2) a Support Vector Machine was one of the algorithms but the other techniques were Decision Trees, Naïve Bayes and AdaBoost, and 3) their dataset was much larger at over 100K apps. The results showed an average accuracy of 81% with the top performer being the Support Vector Machine, followed by Naïve Bayes and AdaBoost. While the authors indicated satisfaction with the accuracy, clearly their research indicated detection using only Permissions was not likely to be successful.

Yerima, Sezer, McWilliams and Muttik (2013) created a tool to scan Android app stores such as Google Play to help expunge them of malware. They used Permissions and API calls as the features, Information Gain for feature selection and a Bayesian classifier. Experiments were run against a dataset of 2,000 apps with 50% malware using various sets of the features selected ranging from 5 – 20 attributes. Their results were that with 15 – 20 of the selected features, a predictive capability at a 90% to 92% could be achieved.

Aung and Zaw (2013) also used Information Gain theory for feature selection based on Permissions only. The selected features were evaluated using K-means clustering for segmentation and classification used Decision Tree and Random Forests algorithms. Testing was conducted on two datasets of 200 and 500 apps. The top performer was Random Forests with accuracy just under 92%. The lowest was one of the Decision Tree algorithms at 85% accuracy.

Glodek and Harang (2013) used a novel approach of feature engineering when they created a system starting with Permissions, Intents, Broadcast Receivers and the presence of embedded applications in the native code (yes/no). The occurrence

frequency of these in a set of malware apps were determined and a set of rules were created based on combinations observed, and these rules then became the real feature vector. These were applied to a Random Forest algorithm from the Scikit-learn library (scikit-learn - Machine Learning in Python, 2010) using a custom dataset of benign apps combined with an existing malware dataset from the Malware Genome Project of Zhou and Jiang (2012). The results of the experiments were an average accuracy of 81%.

A Maturing Research Stream

In a well-known study by Arp, et al. (2014) they created a system they referred to as DREBIN. It used a broad feature set consisting of Permissions, Intents, API calls, app components, hardware components and URLs providing a feature vector of 545,000 attributes. These were the input into a Support Vector Machine for classification. A dataset was created of 129,013 apps that also incorporated the Malware Genome Project malware dataset, ending up with 4% malware content. Their results showed an accuracy of 94%, which was high for the time. Interestingly, DREBIN also output information related to the input features detailing why it classified a specific app as malicious or benign.

In an approach reminiscent of Enck, et al. (2010), Arzt, et al. (2014) used taint analysis for classification, but in their research used static instead of dynamic analysis. The system builds a model based on control flow graphs extracted from an app and uses tainted variables inserted into the flow to follow throughout all possible flow paths. Custom rules then determine if data is being leaked, indicating malware. The number of apps tested was small but they did compare results to two commercially available systems from the time and out-performed them with a 93% accuracy.

In an interesting study of Permission usage in Android apps, Moonsamy, Rong, and Liu (2014) looked at what they termed “required” Permissions versus “used” Permissions. In every Android app there is an xml file named, `AndroidManifest.xml`, that contains information about what the package uses and specifically Permissions that are requested. Often researchers just use this list of Permissions (defined as “required” in this work) because they are easily attainable. However, app developers can use other Permissions that are not in the manifest file. To extract these the byte code has to be decompiled and the source code parsed to find which Permissions were “used.” Moonsamy, et al., extracted both for a dataset that contained over 1,200 benign apps plus the malware apps from the Genome Project and performed a statistical analysis looking at required Permissions for benign apps, required Permissions for malicious apps, used Permissions for benign apps and used Permissions for malicious apps. Their results indicated there was enough difference between required and used, that used Permissions should be considered in malware detection as opposed to only required ones.

Chan and Song (2014) investigated the detection accuracy of using Permissions and API calls compared to using Permissions only. Information Gain theory was used for feature subset selection. The test dataset consisted of 800 apps with 21% malicious. They tested Permissions only versus Permissions and API calls using seven different machine learning algorithms from the Weka library: Naïve Bayes, Support Vector Machine, two Neural Nets (RBF and MLP), Liblinear, Decision Tree and Random Forest. The average accuracy for Permissions only was 88.8% versus 89.3% with API calls indicating no difference. The highest performer was Random Forest at 92%. Interestingly, adding the API calls caused Naïve Bayes and RBF Network to decrease in

accuracy. While the results indicated there was no reason to add API calls as a feature, this is inconsistent with other published research. The small dataset and feature subset selection could have had adverse effects on the answers.

Sharma and Dash (2014) investigated feature selection approaches using Permissions and API calls. The two feature selection methods were Correlation-based Feature Selection (CFS) using Pearson's correlation and Information Gain theory. They started with 35 features (19 Permissions and 16 API calls) that were preselected as important. Based on the feature ranking of the two algorithms, various numbers of the features, top 5, top 10, etc. up to the full 35, were fed into both a Naïve Bayes algorithm and a K-nearest Neighbor (KNN) algorithm using a dataset of 2,000 apps at 50% malware. Overall, there was minimal difference in the results coming through CFS versus Information Gain. Interestingly, smaller numbers of features improved Naïve Bayes while more features improved KNN, and in general KNN performed better with up to 96% accuracy versus 94% with Naïve Bayes.

Another feature selection investigation was performed by Zhao, Fang and Wang (2014) in which using only Permissions as the feature, they incorporated Principal Component Analysis (PCA) for dimensionality reduction. Initially, variations in PCA settings were used with a Support Vector Machine (SVM) to determine the optimum PCA configuration, which in their case resulted in a reduction down to 41 attributes. These features were then used with the a dataset of 454 apps of which 220 were malware for validation with seven machine learning algorithms: Bayesian Networks, Naïve Bayes, NB Tree, CART (Classification and Regression Tree), Random Tree, Decision Tree and an SVM. Overall, the SVM provided the highest accuracy at 90% will all others being

below 87%, however, this result could be because the PCA parameters were tuned using the SVM. No other combinations were reported.

Seo, Gupta, Sallam, Bertino and Yim (2014) created a static analysis tool that looked at “suspicious” Permissions and API calls and performed keyword searches of the source code looking for strings that the authors identified as often being used in malware. The Permissions, API calls and keyword lists were developed based on a statistical analysis of the Malware Genome Project dataset. The tool would then compare what it found in the app source code to the lists and provide a risk ranking. Testing was performed on 76 apps that were selected because they fit the model, but accuracy results were not reported.

In a similar vein, in research by Kang, Jang, Mohaisen and Kim (2015), they identified certain malicious behaviors like command usage with root privilege, hiding SMS notification, and collecting sensitive information as possibly indicating malware. Their system collected data from app source code as to if the app contained any of those behaviors. In the next level their system looked for a usage of what they defined as “critical” Permissions. These features were fed into a Naïve Bayes classifier and tested on a dataset of over 55,000 apps with 8% malware. The accuracy was reported at 98% but given the hand-tooled nature of the features it is uncertain how generalizable this technique might be.

Sun, Li, Yan, Srisa-an, and Pan (2016) created a three tiered feature selection method for reducing the dimensionality of a Permissions feature vector. In the first level they used a forward selection custom ranking system to create the first feature subset. The second level used a filter method based on support to create yet another feature

subset. In the final tier they removed highly correlated features, ending up at a final subset of just 22 Permissions out of the starting 135 Permissions. Validation used a dataset of approximately 5,500 apps with 18% malware and six different machine learning algorithms from the Weka library: Random Committee, Rotation Forest, Functional Tree, Decision Tree (PART), Random Forest and SVM. The top performer was Functional Tree with 96% accuracy.

Two methods of feature selection were investigated by Qiao, Sung and Liu (2016) as means of reducing dimensionality across the set of all Permissions and API calls. The first was a filter method, ANOVA, and the second was a wrapper method, Recursive Feature Elimination (RFE) using an SVM. Validation was performed using three machine learning algorithms: Random Forest, Artificial Neural Network and Support Vector Machine on a dataset of 5,000 custom collected benign apps and malware from the Genome Project. Experiments were run with all features versus the selected features, and then with only Permissions, only API calls, and with both. The results showed that using API calls or API calls and Permissions performed a little better than just Permissions (95% vs 93% on Random Forest). The results also showed there was no appreciable difference between the two feature selection methods or in using all features versus incorporating feature selection. The biggest factor in difference in accuracy was which machine learning algorithm was used with Random Forest and Neural Net performing about the same with 94% accuracy and SVM coming in at 82%.

Another investigation of Permissions requested versus Permissions used was undertaken by Wang, Wang and Zhu (2016) but also included API calls. Feature selection was employed using both Information Gain theory and Correlation-based

Feature Selection using Pearson's correlation (CFS). A number of combinations were run with each feature selection technique including just with used Permissions, just with requested Permissions, only API calls and then used Permissions with API calls. The dataset involved had 2,375 apps with 50% malware. Five machine learning algorithms from the Weka suite were used for validation: Decision Tree, Random Forest, K-nearest-neighbor, Support Vector Machine and AdaBoost. Results showed minimal difference in accuracy from feature selection using Information Gain versus CFS, except when using only API calls in which case Information Gain in general was about 2% better. Across the full test matrix, the top performer was AdaBoost with a reported accuracy of 99.8%, which seems high compared to other published results using the same features.

Verma and Muttoo (2016) used Intents in addition to Permissions in their detection scheme incorporating Information Gain for feature selection. Interestingly, the methodology treated the two features vectors separately, analyzing each app using Intents and then using Permissions. Apps were labeled as malicious or benign if both processes agreed, otherwise it was labeled "suspicious." A dataset of 1,470 apps with 42% malware was used with three classifiers from the Weka library: K-means clustering and two different Decision Trees, ID3 and J48. Results showed that J48 performed best with an accuracy of 94% compared to 92% for ID3 and 74% for the K-means algorithm.

Using a dynamic analysis approach, Yang, Wang, Ling, Liu and Ni (2017) built a customized version of Android as a "behavior inspection platform." It ran a taint tracking program on apps recording their API calls and Permissions use. Experiments were run against a dataset of 3,934 apps with 27% malware. Two classifiers were employed: a Support Vector Machine (SVM) and Naïve Bayes. The SVM was marginally better in

accuracy 98% versus 97% for Naïve Bayes, but false negative rate for Naïve Bayes was very high at 44%.

In an interesting use of dynamic analysis, Mahindru and Singh (2017) ran apps through an emulator to extract Permissions used, which were then vectorized for use by static analysis. They used a dataset of 11,000 apps with five different classifiers from the Weka suite: Naïve Bayes, Decision Tree, Random Forest, Simple Logistic and Lazy Instance-based learner (K^*). Reported results show an accuracy of 99% for Decision Tree, Random Forest and Simple Logistic which seems to be an outlier for using only Permissions as a feature set.

In a novel approach to Permissions analysis for malware detection Shahriar, Islam and Clincy (2017) investigated using natural language processing, specifically Latent Semantic Indexing (LSI) to map the association of related Permissions to malicious apps based on the text in the Permission name or description. The map was created using a custom malware dataset in one dimension and Dangerous Permissions in the other dimension. Dimensionality reduction is then accomplished using Singular Value Decomposition (SVD). Results from various tests showed successful identification of malware ranging from 76% to 89% but there could be an overfitting problem given the overlap of training and test apps.

Nezhadkamali, Soltani and Seno (2017) reported on their research using Permissions, Intents and API calls that also used a concept they referred to as *Feature Pockets*, when two or more features from the three separate feature sets overlap with each other, i.e., have the same resource. A feature refinement process was performed to reduce the dimensionality of the model inputs followed by three different feature

selection methods, tested separately: Information Gain, Gini Impurity and LASSO (L1). The dataset used consisted of the 1,260 malicious apps from the Genome Project plus an additional 498 benign apps. Classification was performed with three machine learning algorithms: Decision Tree, Support Vector Machine and Random Forest. The test matrix included all combinations of feature selection methods and classification methods plus using Permissions only, Permissions and API calls, and Permissions, API calls and Intents. The results showed that using all three features, Permissions, API calls and Intents was more accurate than using just Permissions or Permissions and API calls. In terms of feature selection methods, Information Gain proved more accurate across all classifiers, and lastly, Random Forest, was the highest performing classifier with 99% accuracy. As with the prior study by Shahriar, et al., the small size of the dataset makes the relevance of the high accuracy results somewhat suspect.

Research performed by Altaher (2017) used just Permissions as the feature input with feature selection based on Information Gain theory. For classification, a neuro-fuzzy system was created by modifying an evolving cluster method (ECM) system for generating the fuzzy rules, which then feed into a neural network. A dataset of 500 apps were used, half being malware. Results showed an accuracy of 90% which is comparable to other works using Permissions only. Follow-up work was performed by Altaher and BaRukab (2017) again using Permissions and Information Gain for feature selection. Instead of using ECM to generate the fuzzy rules, they used a Fuzzy c-means (FCM) clustering algorithm. The testing dataset consisted of the 1,260 apps of the Malware Genome Project. Results were marginally better than the prior work reporting 91% accuracy.

Wang, Li, Wang, Liu and Zhang (2018) used a set of features for their study that included Permissions, Intents, API calls and hardware used. A feature selection wrapper method based on a Support Vector Machine (SVM) was used for dimensionality reduction. A dataset of over 116,000 apps with 7% malware was used for testing. Five different machine learning algorithms were used in addition to an ensemble method. The five were: SVM, Random Forest, Naïve Bayes, K-nearest Neighbor (KNN) and Classification and Regression Tree (CART). The ensemble approach was a simple voting scheme among the five. Interestingly the accuracy of all but Naïve Bayes was 98% to 99%, and it was significantly lower with 76% accuracy.

Shang, Li, Deng and He (2018) used a two-phase feature selection approach in their Permissions only research. The first step was to use Pearson's correlation and create the first feature subset by eliminating Permissions below a specified p-value. In the second phase Information Gain theory is used to create a second subset and provide weighting input for the Naïve Bayes classifier used. Tests were run on a dataset of 2,670 apps with 65% malware. Results showed the Naïve Bayes accuracy at 86% which is on par for Permissions only detection methods.

In research by Alswaina and Elleithy (2018), Permissions were the sole feature vector under consideration. Feature selection was performed by a wrapper method using Extremely Randomized Trees for creating a feature subset. The dataset used was that of the Malware Genome Project which was tested using five machine learning classifiers plus one ensemble method: Support Vector Machine, Neural Network, Decision Tree, K-nearest-neighbor (KNN) and Random Forest. The ensemble method was Bagging with an unspecified base classifier. Results showed the top performers were Random Forest

(itself an ensemble method) and Neural Network at 96% accuracy followed closely by KNN at 95%. The numbers appear a bit high for Permissions only, which possibly points back to the small size of the dataset.

Firdaus, Anuar, Karim and Ab (2018) used directories accessed, and system commands in addition to Permissions and API calls as their features. Next they used a wrapper method for feature selection incorporating a Genetic Algorithm from the Weka library. Interestingly, the resulting feature set consisted of only six attributes: three Permissions, two services and one directory, but no system commands. The dataset used consisted of 6,105 apps of which 5,555 were malicious. Validation was performed with five classifiers: Naïve Bayes, Neural Net, Functional Tree, Random Forest and Decision Tree. Testing showed an accuracy of roughly 95% for all five classifiers. This odd result may indicate the GA culled the feature set too far, not providing a broad enough feature vector.

In an interesting study of feature selection techniques for Permissions-based malware detection, Bhattacharya, Goswami and Mukherjee (2019) proposed Particle Swarm Optimization (PSO) using rough set theory and compared results against nine other feature selection techniques, including six filter methods and three wrapper methods. The filter methods were: Pearson correlation, Information Gain, Gain Ratio, Chi-squared, One Rule and Relief. The wrapper methods were: Forward Selection, PSO (not incorporating rough set theory) and Genetic Algorithm. Testing was done with two different datasets and a Decision Tree classifier. Their proposed PSO method improved over the second best methods by 1%, with those second best being Gain Ratio and Relief.

The results also showed all the filter methods being better than any of the wrapper methods.

A Static Analysis Survey

Android security and privacy research has grown tremendously during its ten years of existence. Acar, et al. (2016) surveyed the state of research and categorized it into five major categories: 1) Permission-based access control, 2) app webification issues (the integration of web content into mobile apps), 3) programmer-induced leakage, i.e., poor programming practices, 4) software distribution channels (trust worthiness of app markets), and 5) vendor customization leading to OS fragmentation. Of the publications cited, over 50% were in the Permission-based access control grouping.

A more recent survey by Talal, et al. (2019) took a different approach in research categorization. They presented four broad categories: 1) survey and review, 2) security solutions, subdivided into malware protection techniques and malware detection techniques, 3) malware studies, ranging from data collections to social science models, and 4) ranking and classification, i.e., classifying malware according to their families or security risk level. The majority of research reported on was in the second grouping, security solutions. Of those, approximately 75% were focused on detection versus 25% on protection.

Tam, Feizollah, Anuar, Salleh and Cavallaro (2017) describe malware detection techniques as falling into one of three categories: 1) static analysis which evaluates an app without executing any code, 2) dynamic analysis which executes the app and observes the results, and 3) hybrid analysis which combines techniques of static and dynamic approaches. Within their surveyed work 56% used static analysis only, with the

remaining split between dynamic and hybrid analysis. These numbers align with another review by Sufatrio, Tan, Chua and Thing (2015) in which the publications they cited represented 63% static analysis with again, the remaining split evenly between dynamic and hybrid analysis.

A significant amount of static analysis research uses machine learning (ML). In their survey, Talal, et al. (2019), describe fifteen different ML techniques in use among the work cited. Approximately two-thirds make use of multiple techniques with the rest using a single ML technique. The most common ones in use, and all about the same level were, Naïve Bayes, Decision Tree, Random Forest and Support Vector Machine.

Table 4 presents the results of a survey conducted by this author cataloging 71 publications in which machine learning techniques were used for Android malware detection.

In the collection of research, there were 31 different machine learning techniques used across all of the works, shown along the top row of the table. The columns representing which technique each effort used are ordered by most used on the left with lesser used techniques going to the right.

Survey Analysis

A metadata analysis is presented in Table 5. In this cataloging, the five most used techniques were Support Vector Machine first at 31% of the publications incorporating it,

Table 5
Summary of ML Technique by Publication

ML Technique	# Papers Using Technique	% Papers Using Technique	% in Use
SVM	22	31%	16%
Random Forest	20	28%	15%
NN	16	23%	12%
Decision tree	16	23%	12%
Naive Bayes	11	15%	8%
Bayesian	5	7%	4%
K-nearest neighbor	5	7%	4%
K-means clustering	4	6%	3%
AdaBoost	3	4%	2%
Functional Tree	3	4%	2%
Genetic Algorithm	3	4%	2%
K-Star	3	4%	2%
PART	3	4%	2%
ANFIS	2	3%	1%
Bagging	2	3%	1%
Markov Chains	2	3%	1%
RIPPER	2	3%	1%
Biclustering	1	1%	1%
Conformal Prediction	1	1%	1%
Decision table	1	1%	1%
ECM	1	1%	1%
EMC	1	1%	1%
Fuzzy c-means	1	1%	1%
LSI	1	1%	1%
Particle Swarm	1	1%	1%
Prism	1	1%	1%
Random Committee	1	1%	1%
Randomized trees	1	1%	1%
Rotation forest	1	1%	1%
Simple Logistic	1	1%	1%
SMO	1	1%	1%

second was Random Forest, followed by Neural Networks, Decision Tree and Naïve Bayes. Each of the other techniques are below 10% going from a count of five down to one. The most techniques evaluated by a single source was seven while 34 publications, almost half, cited using a single technique.

In the community of Android malware detection research there are two datasets that are often cited and or used either in whole or in part. Those datasets are the Android Malware Genome Project (Zhou & Jiang, 2012) which contain 1,260 malware samples, and the Drebin dataset (Arp, et al., 2014) which contains 129,013 total samples of which 5,560 are malware and the rest benign. Interestingly, the 5,560 malware samples in Drebin include the 1,260 samples from the Genome Project. For creating custom datasets, there are various places to get benign apps, the main one being Google Play; however, the availability of verified malware samples is much more limited, and rightly so. Two major sources of malware samples are VirusShare (VirusShare, 2020) and Contagio Malware Dump (Contagio Malware Dump, 2020).

Of the work presented above, only three confined themselves to solely using one of these two referenced datasets. All others create a custom dataset but often using samples from one or both of the two referenced datasets. Of the 71 citations, 51 provided information on the source of their dataset's malware samples. Table 6 is a metadata analysis of those sources. Given that the Drebin dataset includes the Genome

Table 6
Dataset Malware Sources

Malware Source	# Cited	%
Genome Project	20	39%
Drebin	14	27%
VirusShare	7	14%
Contagio	10	20%

Project samples, those two represent 66% of the source with VirusShare and Contagio providing the rest at 14% and 20% respectively.

Also of interest is dataset size. In machine learning, more data for testing and training is always better, but assembling a custom dataset can require a lot of work. So it is not surprising to see a wide range of dataset sizes, i.e., the number of APK files used, within the referenced work. The range goes from a low of 106 samples up to a maximum of 206,264 samples. As shown in the metadata analysis in Table 7 around 65% of the research was conducted with datasets of 10,000 or less and only 15% used 100,000 or more.

A final point of interest related to the datasets used is the percentage of APKs in the dataset that is malware. Table 8 presents a metadata analysis of that metric. Note that there are two modalities in evidence: 20% and below being the first and 41 – 60 % being the second. None of the work reported testing with various percentages of malware using the same dataset, so the impact of this distribution is to be determined.

Table 7
Dataset Sizes

Dataset size (# APKs)	# Cited	%
>= 100k	9	15%
10k-100k	12	20%
1k-10k	30	49%
<= 1k	10	16%

Table 8
Dataset Percentage Malware

% Malware	# Cited	%
0-20 %	18	30%
21-40 %	5	8%
41-60 %	30	49%
61-80 %	5	8%
81-100 %	3	5%

Survey Summary

Compared to the computer industry overall, mobile operating systems are a relatively new component of the technology landscape. The convergence to only two major players, Android and iOS, is just over a decade old. New operating systems are always followed by new threat vectors and Android was no different. With just over ten years of research in Android malware detection, there has been significant progress, but there remain gaps that need to be addressed.

One major void is an exhaustive analysis of what Android features are most indicative for malware detection. While there has been significant work in the industry on feature selection for machine learning in general, there is a definite lack of such a focus with respect to Android, with ample evidence shown in the above survey.

Chapter 3

Methodology

Notations

Table 9 lists the notations to be used for the remainder of this text. The format uses bold uppercase characters for matrices (e.g. \mathbf{X}), bold lowercase characters for vectors (e.g., \mathbf{y}) and italicized, uppercase fonts for sets (e.g. F).

The first step in designing the experiments is to select the library to be employed.

Table 9
Symbols

Notations	Description
\mathbf{E}	Expected values
\mathbf{O}	Observed values
n	Number of instances in the dataset
d	Number of all features in the dataset
k	Number of selected features
\mathbf{X}	Binary data matrix with n instance and d or k features
x_{ij}	Feature values for i^{th} instance and j^{th} feature
\mathbf{Y}	Binary class label vector for all n instances
y_i	Class value for i^{th} instance
\mathbf{W}	Feature weight vector for k features
w_j	Feature weight value for j^{th} feature
\mathbf{F}	Original feature set with d features
\mathbf{S}	Selected feature set with k selected features
\mathbf{V}	Contrast variables matrix, n instances with 3 features
\mathbf{T}	Set of feature selection methods

This notation closely follows that used by Li, et al., (2017).

Scikit-Learn (scikit-learn - Machine Learning in Python, 2010) was selected as the library to use due to it being open source, written in Python and having a broad list of available algorithms.

Feature Scope

The breadth of feature types used in static analysis for Android malware detection is presented in Table 3. For this research, the feature types under test were the top three of interest in the community: Permissions, API calls and Intents. The next two in the list, hardware and app components, are essentially redundant to the first three since to use either, an API call is required and possibly even a Permission. The remaining features, such as CFGs, URLs, etc., may be important in some domains but are believed to be less important in this research given that every Android app, malware and benign, will make use of a variety of Permissions, API calls and Intents providing a broad detection surface. Adding other feature sets is unlikely to increase that detection surface area enough to justify the increase in computational time and therefore in detection time.

Given the diversity of the combinations of the three in use, all seven combinations are used as starting feature sets in the experiment as shown in Figure 2. This includes using the three individually as well as all together as a single feature set in addition to the other various combinations. Each combination occurs prior to any subset selection. Given that there are 158 Permissions, 295 Intents and 581 significant API calls, the starting feature sets range from as low as 158 attributes (Permissions only) to 1,034 attributes (Permissions, Intents and API calls).

Feature Selection Algorithms

As described in the section Feature Selection, there are three main objective approaches to feature selection:

1. Filter methods
2. Wrapper methods
3. Embedded methods

Given that the focus of this research work is to provide an expansive view of feature importance, multiple feature selection techniques were used from each of the above categories in order to compare and contrast the selected features with representative approaches across the spectrum. Table 10 lists the eleven selected techniques: three univariate filter methods, three multivariate, three wrapper methods and two embedded methods. Univariate filter methods assume independence of the features from one another and only select based on the correlation with the class. These three univariate algorithms were selected due to the popularity of those methods in feature

Table 10
Feature Selection Algorithms Text Matrix

FS #	Method	Category
1	Chi-Square	Filter - Univariate
2	Information gain	Filter - Univariate
3	Relief	Filter - Univariate
4	CFS with Chi-Square	Filter - Multivariate
5	CFS with Information gain	Filter - Multivariate
6	CFS with Relief	Filter - Multivariate
7	RFE with NN	Wrapper
8	RFE with Random Forests	Wrapper
9	RFE with SVM	Wrapper
10	Lasso regression (L1 regularization)	Embedded
11	Ridge regression (L2 regularization)	Embedded

selection and therefore it would be informative to see the results they produce compared directly.

Univariate Filter Methods

Chi-Square

The Chi-Square test is used in statistics to test the independence of two events, or in the case of feature selection to test whether the occurrence of a specific feature value and the occurrence of a specific class are independent. It can be expressed as:

$$\chi^2(\mathbf{X} | \mathbf{y}) = \sum_{i=1}^r \sum_{j=1}^d \frac{(\mathbf{O}_{ij} - \mathbf{E}_{ij})^2}{\mathbf{E}_{ij}} \quad (1)$$

where r is the number of different values in a given feature vector, d is the number of features in the dataset, O is the count of observed values and E is the expected frequency (Li, et al., 2017).

Information Gain

Information Gain (also known as Mutual Information) is a statistical method that measures the amount of information shared between a feature and its class labels. It is based on the concept of information entropy from information theory. Information Gain is defined as:

$$I(\mathbf{X} | \mathbf{y}) = \sum_{x_i \in X} \sum_{y_i \in Y} P(x_i, y_i) \log \frac{P(x_i, y_i)}{P(x_i)P(y_i)} \quad (2)$$

where $P(x_i)$ is the probability of x_i over X , $P(y_i)$ is the probability of y_i over Y , and $P(x_i, y_i)$ is the joint probability of x_i and y_i (Li, et al., 2017).

Relief

The Relief algorithm (Kononenko, 1994) estimates the quality of attributes according to how well their values distinguish between instances that are near to one another. Given a randomly selected instance vector \mathbf{x}_i , it searches for the two nearest neighbors: one from the same class, and the other from a different class, and then updates the quality estimate for all the features, depending on the values for \mathbf{x}_i .

Relief can be expressed as:

$$R(\mathbf{X}|\mathbf{y}) = \frac{Gini' \times \sum_{x \in X} P(x)^2}{(1 - \sum_{y_i \in Y} P(Y)^2) \sum_{y_i \in Y} P(Y)^2} \quad (3)$$

where $Gini'$ is a modified version of the Gini-index which is highly correlated with Information Gain covered earlier, $P(x)$ is the probability of the values of vector \mathbf{x} , and $P(y)$ is the probability of the classes in the labeled set.

Multivariate Filter Methods

Multivariate methods select feature importance based on higher correlation with the class, just like univariate methods, but then look for low correlation between features. CFS (Correlation-based Feature Selection) will be used to take the same three sets of correlations and consider the interaction among features

Correlation-based Feature Selection (CFS)

Correlation-based Feature Selection (CFS) is based on the thesis that “A good feature subset is one that contains features highly correlated with (predictive of) the class, yet uncorrelated with (not predictive of) each other” (Hall, 1999). It requires an initial correlation analysis on which to base the selection algorithm. While this initial technique is required it is not subscribed and any approach can be used. For this research, the three

previously discussed univariate methods were used: Chi-Square, Information Gain and Relief.

CFS uses a concept of “Merit” as a heuristic by which to compare inter-feature correlation. It is given by:

$$M_S = \frac{k \overline{r_{cf}}}{\sqrt{k + k(k - 1) \overline{r_{ff}}}} \quad (4)$$

where S is the feature subset with k number of selected features, $\overline{r_{cf}}$ is the mean feature-class correlation and $\overline{r_{ff}}$ is the mean feature-feature correlation (Hall, 1999).

Wrapper Methods

Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) is an iterative procedure using backward feature elimination. However, as opposed to eliminating just one feature at a time, it allows for evaluating and eliminating (or keeping) feature subsets (Guyon, Weston, Barnhill, & Vapnik, 2002). RFE incorporates a classifier for ranking the features in each iteration. RFE can be described by:

$$\mathcal{F}^d \rightarrow S^k \rightarrow \boxed{\text{classifier}} \rightarrow S^{k-n} \quad (5)$$

↑
while $k > 1$

where \mathcal{F}^d is the original feature set with d features, S^k is a selected feature set with k selected features and n is the subset reduction value. Guyon and collaborators originally developed RFE using an SVM as the classifier, but it has since been used with different models among researchers, most notably Random Forest, as exemplified by Ustebay, Turgut, and Aydin (2018) as well as a Neural Network as demonstrated by Peterson and Coleman (2005).

Embedded Methods

Ridge regression

Ridge regression, also known as Tikhonov regularization, is a multiple regression technique using a cost function based on the residual sum of squares. It adds a regularization (or penalty) function that is an L2 norm, i.e., based on Euclidean distance. Optimization is based on minimizing the cost function, which is defined as:

$$\sum_{i=1}^n (y_i - \sum_{j=1}^d x_{ij} w_j)^2 + \lambda \sum_{j=1}^d w_j^2 \quad (6)$$

|— regression model —| |— penalty function —|

where \mathbf{y} is a vector of the class observations for n number of instances in the dataset, \mathbf{x} is a matrix of the model predictor variables for n instances by d number of features, and \mathbf{w} is the vector of weights (or model coefficients) corresponding to each feature.

To control the regularization function, λ is used as a tuning parameter that increases or decreases the size of the penalty, which for Ridge, is the sum of the squares of the weight coefficients. This implies that for $\lambda = 0$ the coefficients are the same as simple linear regression and as $\lambda \rightarrow \infty$ all coefficients are driven towards zero, but never actually reach zero (James, Witten, Hastie, & Tibshirani, 2017).

LASSO regression

LASSO, which stands for *Least Absolute Shrinkage and Selection Operator*, was developed to improve on Ridge regression by performing L1 regularization using Manhattan distance instead of using L2. The cost function is similar to Ridge except for the penalty term and is defined as:

$$\sum_{i=1}^n (y_i - \sum_{j=1}^d x_{ij} w_j)^2 + \lambda \sum_{j=1}^d |w_j| \quad (7)$$

|— regression model —| |— penalty function —|

where \mathbf{y} is a vector of the class observations for n number of instances in the dataset, \mathbf{x} is a matrix of the model predictor variables for n instances by d number of features, and \mathbf{w} is the vector of weights (or model coefficients) corresponding to each feature.

As with Ridge, λ is a tuning parameter that controls the size of the penalty, but for LASSO, the penalty is the sum of the absolute value of the weight coefficients. This implies that for $\lambda = 0$ the coefficients are the same as simple linear regression and as $\lambda \rightarrow \infty$ all coefficients approach zero and some actually equal zero implicitly selecting features to be eliminated, which is the desired improvement of LASSO over Ridge (Tibshirani, 1996).

Machine Learning Algorithm Selection

The goal of this research was to show the variation in feature selection inputs and indicate how that variation reflects in different machine learning models. It was not a goal to find a best performing feature set, nor to find a best performing machine learning model.

The classifier selection was based on common usage among other researchers so as to make replication and comparison easy. In the same vein, with the exception of Random Forest, ensemble techniques were eliminated, such as boosting and stacking. The final consideration was performance. It was desired to have models that have consistently exhibited high accuracy in various domains.

Based on the data presented earlier in Tables 1, 2 and 5, the three classifiers selected were Random Forest, Support Vector Machine and Neural Network. Specifically, the Neural Network classifier was a Perceptron, chosen because the problem at hand is one of binary classification and with a single layer would be expected to have high performance in terms of speed to convergence. The list of selected algorithms is presented in Table 11.

Experiment Design

Preprocessing

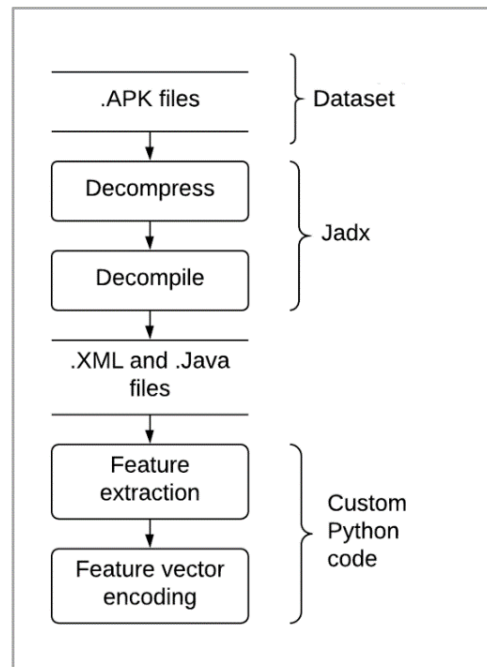
A high-level view of the preprocessing procedure is presented in Figure 3. The malware dataset used is a custom collection of over 119,000 Android apps with an eight percent malware component. See the section [Datasets](#) for more details.

There are many tools available for reverse engineering Android apps. Four requirements were identified for tool selection for this project. First, it needs to be open source in order to provide a level of transparency as well as be affordable. Second, the open source project needs to be actively maintained. Given that Android itself changes from time to time and is customized by different vendors, it is critical to use a tool that stays up to date with those changes. Third, the tool needs to have a command line interface to support automation. A good test database of APKs consists of thousands of

Table 11
ML Classifiers Text Matrix

ML #	ML Algorithm
1	Random Forest (RF)
2	Support Vector Machine (SVM)
3	Neural Network (NN)

Figure 3
Preprocessing



files, too many to use a GUI one file at a time. And finally, the decompiled files need to be in Java simply due to the author's expertise in that language versus Android assembly language.

Jadx (Dex to Java decompiler, 2019) was selected. It meets all the stated requirements and is a popular tool among Android researchers. Firdaus, Anuar, Karim and Ab (2018) used Jadx in their research on Android feature selection. Pauck, Bodden and Wehrheim (2018) presented a new approach for comparing taint analysis tools and used Jadx to prep the data. It was also used as part of the development of a new classifier approach to get around obfuscation by Martin, Menendez and Camacho (2017). And Chen, Fan, Chen, Su, Li, Liu and Xu (2019) actually incorporated Jadx as part of their Storydroid tool for use in app development.

Jadx has a graphical user interface as well as a command line interface, the latter making it convenient for integrating with other tools. Its main function is decompressing APK's resulting in .xml files and .dex files. It then decompiles the .dex files into Java source code.

From the resulting .xml and .java source files, the features of interest are then parsed using custom Python code. For this experiment, those features are Permissions, Intents and API Calls. To simplify data management each feature is assigned a unique identifier consisting of the first letter of the type, P for Permissions, I for Intents and A for API Calls, followed by the concatenation of an integer value ranging from 1 to the maximum number of features available. As an example, Permission IDs went from P001 to P158 and the Permission `READ_EXTERNAL_STORAGE` was assigned an ID of P100.

A separate module of the code then performs binary encoding of the extracted features to account for the existence (1) or non-existence (0) of each feature in each APK file, an example of which is depicted in Figure 4 as a dataset fragment of the Permissions file. The encoding creates three datasets, one for each feature type as well as a dataset containing just the target variable indicating malware (1) or benign (0). Keeping each of the four datasets in separate files makes the mechanics of concatenating them for the test dataset of interest a simple exercise. Each of these four files contains 119,183 rows of data, one row per APK.

Figure 4
Binary Encoded Dataset Fragment

APK	P004	P005	P006	P008	P010	P012	P013	P015	P016	P025	P028	P031	P034	P037	P041	P043	P044	P045	P050	P053	P055	P058	P059	P060	P061	...	P157
1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	
8	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
9	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
10	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
12	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
15	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
18	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
20	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
⋮																											⋮
119183	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0

These feature vector datasets segmented as Permissions, Intents and API Calls can then be used individually or combined into larger feature vectors by concatenating the segments. Table 12 shows the seven combinations of the three feature types and associated feature counts possible for each combination.

Feature Subset Selection

Feature subset selection is the process of selecting a subset of relevant features for use in model construction. The goal is to remove features that are either redundant or

Table 12
Datasets Test Matrix

DS #	Dataset Features Types			DS ID	Count
1	Permissions	Intents	API Calls	PIA	1,034
2	Permissions	Intents		PI	453
3	Permissions		API Calls	PA	739
4		Intents	API Calls	IA	876
5	Permissions			P	158
6		Intents		I	295
7			API Calls	A	581

irrelevant. For this experiment, there are four phases (as depicted in Figure 5) encompassed in an 8-step process (shown in Figure 6). Phase 1 starts with all Permissions, Intents and API calls. The next three phases, *Algorithmic Selection*, *Performance Selection* and *K-based Selection* are described in the following sections.

Algorithmic Selection

Algorithmic selection refers to the feature selection method actually eliminating non-relevant features as part of its process. Of the eleven feature selection algorithms used, eight perform this type of subset selection. The three that do not are Information Gain, Chi-squared and Ridge Regression. For example, if 100 features are submitted to the Information Gain algorithm, it will return the weights for all 100 features. However if 100 features are submitted to the LASSO Regression algorithm, it will return weights only for the features the algorithm deems significant, so that would be some number less than 100 most likely. In other words, in this case, LASSO Regression has algorithmically selected a subset of features.

Figure 5
Feature Subset Selection Phases

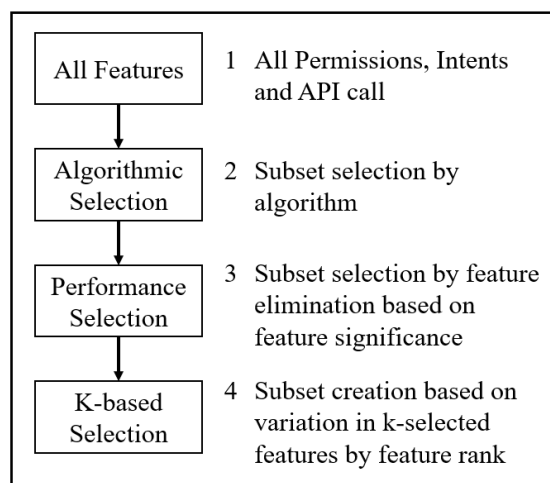
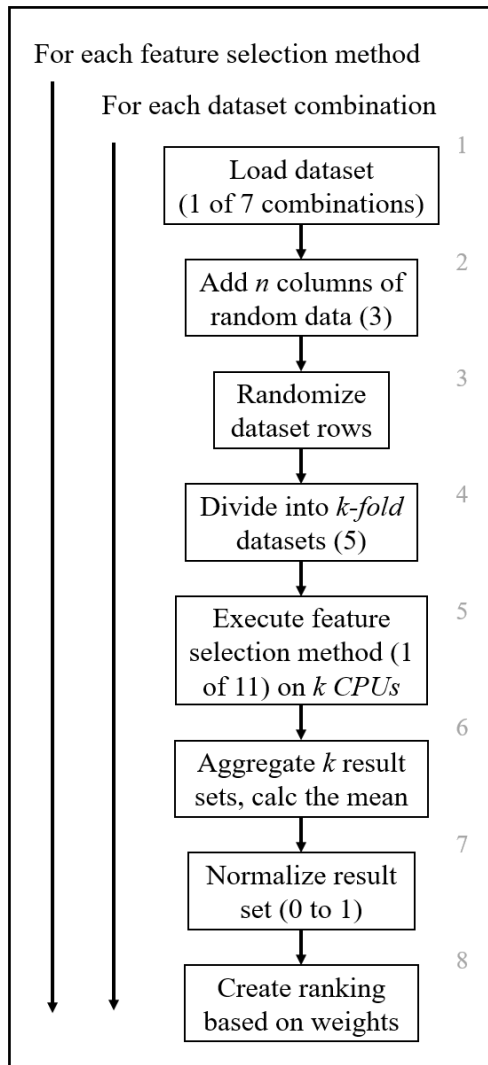


Figure 6
Feature Subset Selection Process



In many library implementations the operator can specify the number of features to return, so in this example, one could submit 100 features to Information Gain and request some arbitrary number of features in return, say the top 10. In that case, the algorithm would return only the top 10 weights, but that is arbitrary subset selection based on the operator's experience, domain knowledge or simple guess, but having nothing to do with the calculations of the algorithm.

Algorithmic selection steps are 3 - 7 in Figure 6. They include randomizing the rows of the dataset, then dividing it into k -fold datasets, specifically five, resulting in five datasets of approximately 24k rows, and each is then submitted to one of the eleven feature selection algorithms. The five results are then aggregated and the mean is calculated. These results are then normalized so that all weights are in the range of zero to one.

Figure 7 is an example using dummy data of what a resulting subset of 25 features, P01 – P25, might look like, with each feature having a weight assigned by the feature selection algorithm.

At this point in the process there are two distinct types of results, one where all features fed into the feature selection algorithm are returned with weights, and a second where the only features and weights returned are those deemed significant by the selection algorithm. These two cohorts will be referred to as C1 and C2 respectively.

For C1, no features have been eliminated. Put conversely, no features have been selected. A process is needed to select the significant features. For the C2 cohort, there are feature subsets as selected by the respective algorithms, but there has been no process to evaluate the quality of those subsets. The performance selection process is used to address these two issues.

Figure 7
Algorithmic Selection Results Example

25 features (all or a subset)																								
P01	P02	P03	P04	P05	P06	P07	P08	P09	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24	P25
0.827	0.842	0.335	0.342	0.568	0.235	0.727	0.327	0.473	0.624	0.057	0.213	0.733	0.540	0.805	0.755	0.949	0.222	0.830	0.395	0.664	0.792	0.591	0.495	0.005

Performance Selection

Performance selection refers to analyzing the performance of the feature selection algorithm and further subsetting by elimination of features deemed irrelevant based on an independent measure of the feature selection algorithm's output. From the weights assigned by algorithmic selection, rank order of importance can be determined, however, from this ranking of features, it is not evident how to threshold the ranking to incorporate only important attributes and to exclude others as noise. As an example, assume a feature selection algorithm returns a subset of 400 features, each having an associated real number weight. One could consider the attribute with the highest weight value as the most important feature and the attribute with the smallest weight value as the least important feature, but how does one determine if all 400 features are relevant? Rather, is it the top ranked 25, or 50, or 100 that are relevant?

That is the question addressed by Tuv, Borisov and Torrkols (2006) when they introduced the use of "artificial contrast variables." To quote their introduction of the concept:

In order to determine a cut-off point for the importance scores, there needs to be a contrast variable that is known to be truly independent of the target. By comparing variable importance to this contrast (or several), one can then use a statistical test to determine which variables are truly important.

In their experiments using contrast variables ensembled with a Random Forest classifier, Tuv, Borisov and Torrkols (2006) show that the technique outperformed RFE and CFS for feature selection.

In a work on determining causality, Guyon, Janzing and Scholkopf (2010) refers to these contrast variables as “probes” similar to those used in statistics for ranking by comparing the index of ranked variables to the index of hypothetical variables from a null distribution set of irrelevant variables. They further describe the use of probes as “...relatively straightforward for regular feature selection” and even reference their use in a previous work by Guyon, et al. (2008).

Kursa and Rudnicki (2011) used a Random Forest classifier for setting feature importance in their gene expression dataset and contrast variables combined with a Boruta algorithm for feature subset development. Lin, et al. (2012) used contrast variables combined with RFE wrapping an SVM for feature selection in order to subset a high dimension database from chromatography–mass spectrometry systems. In another medical application, Paja and Pancierz (2017) used Information Gain for weighting and then contrast variables for thresholding to develop their final feature subset.

In using contrast variables, one has to consider the possible impact of the added columns on feature ranking and on performance. This could range from zero (no contrast variables) to some number greater than the number of real features. In terms of a percentage of the total number of real features, in practice researchers report successful results with ranges from 0.5% to 2.5%. Considering an example dataset with 400 features this would equate to adding from 2 to 10 artificial attributes.

For this work, thresholding was also accomplished using artificial contrast variables. They are treated as additional features, just as the Permissions, Intents and API Calls. The major difference is that instead of consisting of data extracted from actual Android apps, the contrast features are populated with data from a random number

generation algorithm, in this case, 0 or 1 due to the binary nature of the datasets. To the feature selection algorithm, these contrast features are just additional features, no different than the Permissions, Intents or API Calls, and the algorithm assigns weights to the contrast features just like all other real features.

In that these experiments evaluate all features in each set and run time was not important, the number of contrast variables used was not significant. The number ultimately selected and incorporated into each set was three. Early prototypes were run with a single contrast variable and it was observed that some algorithms provided inconsistent results (weights) when repeated, so three were tested and using the average of the weights made the observed variation much smaller. No runs were made with more than three since it was judged there would be diminishing return.

The process then is to contrast the weights (or more precisely the mean of the weights) of these features of random data to the weights of the features from algorithmic selection and determine if further feature reduction is indicated.

For this step, let \mathbf{F} represent the original matrix of feature vectors:

$$\mathbf{F}^m = \{\text{Permissions, Intents ...}\} \text{ (see Table 12)}$$

where $m = 7$ (the number of instances).

$$\mathbf{F}^V = \mathbf{F}^m + \mathbf{V}$$

where \mathbf{V} is the matrix of contrast variables added to the original matrix of attributes for each dataset. A matrix of candidate feature sets, \mathbf{F}^T is created by applying each feature selection technique T where:

$$\mathbf{T}_r = \{\text{Chi-Square, Information Gain, ...}\} \text{ (See Table 10)}$$

and $r = 11$ (the number of techniques to be employed) onto each original feature set \mathbf{F}^V :

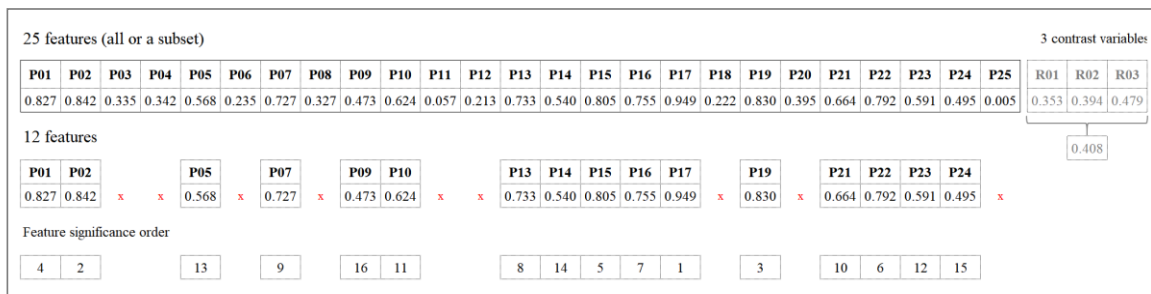
$$F^T = T_r(F^V)$$

The size n of each candidate feature set will vary based on the results of the selection algorithm. For each feature set in F^T , the values of the weights vector, w , are determined by T which equates to a feature importance measure. Let w^F be the weights matrix for the original features F^m , and let w^V be the weights matrix for the contrast variable features V . Then the preliminary subset S' of significant features is built from the features of F whose weights are greater than the mean of the weights of the features of V ($\overline{w^V}$), such that:

$$S' \subseteq F^m$$

Getting back to the previous example, Figure 8 shows the weights of the contrast variables to the right of the weights of the original features, labeled as R01, R02 and R03, where the label 'R' stands for *random*. They are separate in the figure to help visualize the process, but the feature selection algorithm just sees them as features no different than P01 - P25 and assigns weights similarly. (See also step 2 in Figure 6.) In this example, the mean of the weights of the contrast variables is 0.408. The weight of each feature, P01 - P25, is then compared to that mean, and if the weight of that feature is less than 0.408, it is eliminated. In the figure this is represented in the second row which shows 9

Figure 8
Performance Selection Results Example



of the 25 features removed. Then, as shown on the third row of the figure, an importance order is inferred based on the weights, resulting in a feature significance order for this instance with the range 1 - 16.

With respect to the C1 cohort where all features have a weight value, the population of \mathcal{S}' is trivial. However, for C2, there are three variations possible. The first is where none of the feature weights are greater than $\overline{\mathbf{w}^V}$:

$$\mathbf{w}_{max}^F \leq \overline{\mathbf{w}^V}$$

meaning the algorithm that selected those features could not distinguish between real features that are truly significant and random data. The second scenario is the opposite, where all of the feature weights from the algorithmic selection process are greater than $\overline{\mathbf{w}^V}$:

$$\mathbf{w}_{min}^F > \overline{\mathbf{w}^V}$$

This result has the performance selection process in complete agreement as to the significance of all the features selected. The third scenario is a mix of these two extremes where some of the feature weights are greater than $\overline{\mathbf{w}^V}$ and some are not:

$$\mathbf{w}_{min}^F < \overline{\mathbf{w}^V} \geq \mathbf{w}_{max}^F$$

For comparing this effect, let the number of features returned from a given algorithmic selection process be n_{AS} and the number of features returned from the performance selection process as n_{PS} , then we can define the effectiveness of that algorithm for feature selection as:

$$\text{eff}_i = \frac{n_{PS}}{n_{AS}} \quad (8)$$

where i is the feature selection method in Table 10. In such a rating, 100% effectiveness would be the case in which performance selection was in full agreement with the algorithmic selection results and 0% effectiveness is the case where the feature selection algorithm could not distinguish between significant features and random data.

Feature Ranking

As described by Bolon-Canedo, Sanchez-Marono and Alonso-Betanzos (2013), there are two main approaches to evaluating feature selections, *individual evaluation*, which provides an ordered ranking of features, and *subset evaluation*, which provides a candidate feature subset. In that the goal of this research was to analyze the importance of all Android features within the three categories of Permissions, Intents and API Calls, as opposed to selecting a single, best performing feature set, the individual evaluation approach was selected which provides an ordered ranking of features.

In order to evaluate performance of specific features across a range of subset creation methods an ensemble voting scheme was chosen based on quartile membership of the ordered set and across two data stratifications: dataset combination and feature selection method. There is a long history of quartile analysis as an evaluative technique going as far back as Tukey (1977) in his seminal work *Exploratory Data Analysis* although he used the term *hinges*. As defined by Langford (2006), the simplest way to delineate a quartile is to find the median of a dataset, the number which puts at least half of the data values at that number or below and at least half of the data values at that number or above, and then to define the first quartile (Q1) to be the median of the bottom half, and the third quartile (Q3) to be the median of the top half. Data below the Q1 point is in the first quartile and data above the Q3 point is in the fourth quartile.

Quartiles are often used in statistical outlier identification (Rousseeuw & Hubert, 2011) where the aim is to minimize the impact of outliers. While that was not the goal desired here, the identification task is the same. The approach is generic in that it spans many research streams. As examples, Shih and Liu (2016) used quartile analysis for threshold determination in their work in image processing, and Lee and Sumiya (2010) employed it to geo-locate social event occurrence based on Twitter data.

Final determination to use quartiles came after data statistics were computed using half deciles, deciles, the selected quartile, second quartile and percentiles. The goal of the analysis is to provide a meaningful representation of importance. It was determined based on the range of statistical calculations that half deciles and deciles filtered too much of the results. Conversely, second quartile did not adequately bring higher performers to the top and percentiles simply did not provide a clear demarcation point.

Cluster analysis was another alternative considered but rejected. While it could show the groups, theoretically some being significant and others not, but there is no inherit ranking such as

$$Q1 < Q2 < Q3 < Q4$$

as naturally available in quartile analysis.

Ranking by dataset combination

Define ρ_{di} to be the number of times the order of feature f_i is in the first quartile for the d^{th} combination among the m dataset combinations used. Then the range of ρ_{di} can be given as:

$$0 \leq \rho_{di} \leq |T|$$

where T is the set of feature selection methods used with $|T| = 11$, and

$$1 \leq i \leq |F|$$

where F are the original feature sets based on m combinations. To be more concrete, if $\rho_{di} = 11$ then the i^{th} feature of F was ranked in the top 25% by each feature selection method, i.e., it is selected as an important feature. Conversely if $\rho_{di} = 0$ then none of the feature selection methods ranked that feature as important. (Note that F does not contain the contrast variables as their usefulness was for performance selection and are not to be treated as real features for the ranking process.)

For comparison of ρ_{di} across the data stratifications, a permutation (reordering) of features can be defined by:

$$\langle \pi(d1), \pi(d2), \dots, \pi(d, |F|) \rangle$$

with

$$\rho_{\pi(d1)} \geq \rho_{\pi(d2)} \geq \dots \geq \rho_{\pi(d, |F|)}$$

so that sequences can be mapped.

Ranking by feature selection method

Define $\rho_{\tau i}$ to be the number of times the order of feature f_i was in the first quartile of the τ^{th} algorithm among the n feature selection methods used. Then the range of $\rho_{\tau i}$ can be given as:

$$0 \leq \rho_{\tau i} \leq |F|$$

Note that while there are seven feature set combinations, referring back to Table 12 one can see that any given feature type can only be in four of the seven combinations so that the actual range of $\rho_{\tau i}$ is:

$$0 \leq \rho_{\tau i} \leq 4$$

Similar to the prior concrete example, if $\rho_{\tau i} = 4$ then the i^{th} feature of \mathbf{F} was ranked in the top 25% by each combination in which that feature type existed, i.e., it is selected as an important feature. Conversely if $\rho_{\tau i} = 0$ then none of the combinations had it ranked as an important feature.

As with dataset variation, for comparison of $\rho_{\tau i}$ across selection methods, a permutation can be defined by:

$$\langle \pi(\tau 1), \pi(\tau 2), \dots, \pi(\tau, |F|) \rangle$$

with

$$\rho_{\pi(\tau 1)} \geq \rho_{\pi(\tau 2)} \geq \dots \geq \rho_{\pi(\tau, |F|)}$$

so that sequences can be mapped.

K-based Selection

The final feature subset \mathcal{S} is then built by varying the number of features selected, k , where

$$k_i \in \{5, 10, 15, 20, 30, 40, 50, 75, 100 \dots d_{\mathcal{S}_i}\}, 1 \leq i \leq n$$

a set arbitrarily selected to provide robust coverage of the span of possible selected subsets, and $d_{\mathcal{S}_i}$ represents the largest number of features in the subset vector which can vary based on the prior subset selection processes and will be less than or equal to the original number of features in the dataset, i.e., $d_{\mathcal{S}_i} \leq d$.

Then

$$\mathcal{S}_i \in \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3 \dots \mathcal{S}_n\}, 1 \leq i \leq n$$

where \mathcal{S}_i is the i^{th} feature set with k_i features.

For an understanding of how \mathcal{S} is built, let us refer back to the example where after performance selection there are twelve features ranked in order of importance. As

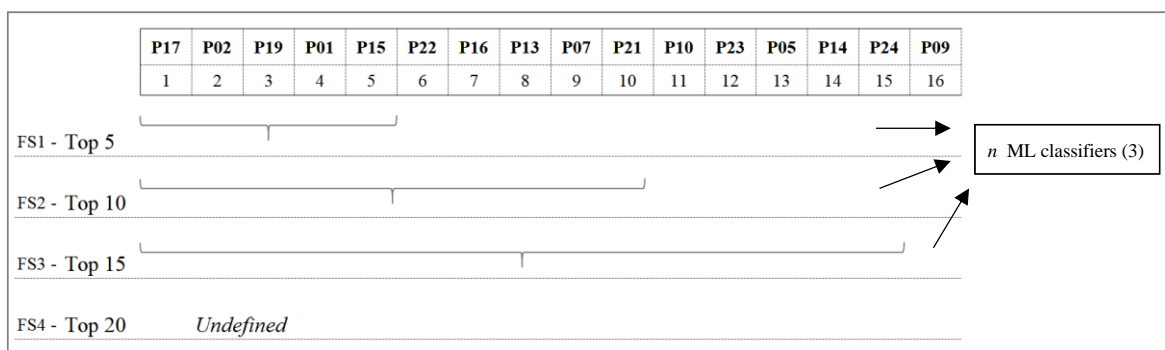
shown in Figure 9, the first k -based subset in the example is five, labeled as FS1 for feature subset one. The five top ranked features, P17, P02, P19, P01 and P15 make up the feature subset. That subset is then validated using n machine learning classifiers. Next, for FS2, the top 10 ranked features are used as a subset and that subset is run through the same n classifiers. This is followed by FS3 and the top 15 features. However, at FS4 (20 features), based on the number of features down-selected as part of the algorithmic selection and performance selection processes, there are not enough features left to create a k -based subset of 20 features, so the process ends with FS4 and all following k -based subsets, i.e., where $k \geq 20$ is undefined.

Each combination of dataset and feature selection method (e.g. Permissions only and Information Gain) can have a different number of features available going into the k -based selection phase which implies that each combination can have a different number of feature subsets used in the validation phase.

Validation Analysis

Experiments were run with the various machine learning techniques described above using the previously described malware database. The key measurements coming

Figure 9
K-based Selection and Validation



from the experiments were number of true positives, number of true negatives, number of false negatives and number of false positives as shown in the confusion matrix of Figure 10 and defined in Table 13.

From this raw data, calculations are made for: true positive rate (TPR), false positive rate (FPR), accuracy, precision and F-measure.

True Positive Rate is defined as: $TPR = \frac{TP}{TP+FN}$

False Positive Rate is defined as: $FPR = \frac{FP}{FP+TN}$

Accuracy is defined as: $Accuracy = \frac{TP+TN}{TP+FN+FP+TN}$

Figure 10
Experiment Confusion Matrix.

		Actual Condition	
		Positive (malicious)	Negative (benign)
Predicted Condition	Predicted Positive	TP	FP
	Predicted Negative	FN	TN

Table 13
Experiment Measurements

Measure		Description
TP	True Positive	# malicious apps classified as malicious apps
TN	True Negative	# benign apps classified as benign apps
FN	False Negative	# malicious apps classified as benign apps
FP	False Positive	# benign apps classified as malicious apps

Two parameters that go into the calculation of F-measure are *precision* (p) and *recall* (r).

Recall, or *sensitivity*, is defined the same as TPR in this context: $r = \frac{TP}{TP+FN}$

Precision, also known as the positive predictor value, is described in the following formula.

Precision is defined as: $p = \frac{TP}{TP+FP}$

F-measure or F1 score is then the harmonic mean of precision and recall.

F-measure is defined as: $F\ measure = 2 \cdot \frac{p \cdot r}{p+r}$

These five measurements are all standard metrics used in evaluation of machine learning tools for Android malware detection. In a survey of measurements in use, by Talal, et al. (2019), the proposed measurements align with five of the top seven as shown in Table 14. Also of note are the two time-based metrics: *Detection time* and *Training time*. Given the current experiments execute training and detection as part of the same process, *elapsed time* is the metric of interest.

Somewhat related in terms of system level performance, but not included in the table, is memory, both internal (system RAM) and external (drive space). These were not included in the observations since the size of available RAM is going to affect overall processing time due to the operating system paging and swapping to the disk drive when more memory is needed than is available, so in that sense it is already part of the metric. Drive space was not considered simply because it is very inexpensive and should not be allowed to be a bottleneck in performance.

The other three items, unchecked in the table, AOC, FNR and TNR, are all calculations from the same raw data so can easily be calculated if desired.

Table 14
Summary of Measurement Criteria.

Metric	% in Use	Current Work
Accuracy	92%	✓
TPR (True Positive Rate)	72%	✓
FPR (False Positive Rate)	52%	✓
Precision	32%	✓
AUC (Area under curve)	24%	
Detection time	24%	✓
F-measure	12%	✓
FNR (False Negative Rate)	8%	
TNR (True Negative Rate)	4%	
Training time	4%	✓

The results covered in the next chapter will incorporate all six metrics to provide consistency with other published works thus providing ample opportunity for comparison.

Resources

Systems

The system used in the research was a Windows 10 system running on an Intel i7 (8 core, 1.8 GHz) with 16 GB of RAM and 8 TB of external storage.

Libraries

As discussed in the previous section, the machine learning library used was Scikit-learn (scikit-learn - Machine Learning in Python, 2010).

Datasets

There is a maxim in machine learning that more data is always better. When dealing in the arena of malware detection, the problem is that data is not easily attained.

The first step in the process is gathering labeled applications, i.e., labeled "malware" or "benign". The next step is decompiling the apps and extracting the desired features. This can be very computationally intensive with durations in terms of days or weeks depending on the number of apps. Finally, the extracted data needs to be cleaned, encoded, and formed into datasets appropriate for input into machine learning libraries. Such required level of effort results in much malware research having smaller rather than larger datasets. Of the literature survey conducted herein, over 70% of the papers used datasets containing on the order of 10,000 instances or less, and 15% of the papers described using 1,000 samples or less. A major goal of this research was to use an exceptionally large dataset, large at least with respect to comparative research.

In their survey on the availability of forensic datasets, Grajeda, Breitingerr and Baggili (2017) listed only two Android malware datasets that meet the above criteria: Drebin (Arp, et al., 2014) and Andro-AutoPsy Lab (Jang, Kang, Woo, Mohaisen, & Kim, 2015) as shown in Table 15. The Drebin dataset is older and even contains some malware samples from an older set, that of the Genome project (Zhou & Jiang, 2012).

Andro-AutoPsy, compiled as part of the Andro-Autopsy project at the University of Korea Hacking and Countermeasure Research is newer and contains almost twice as many malware samples, all of which came from the two most respected malware repositories in the community, Contagio Malware Dump (Contagio Malware Dump, 2020) and VirusShare (VirusShare, 2020). As another level of quality check, in order to be included in the Andro-AutoPsy dataset, the malware had to have been diagnosed by at least ten different antivirus vendors. The final collection of almost 10,000 samples span a range of 30 different malware families. With respect to the 100K+ benign samples, all

were downloaded from Google Play. Many comparisons have been made to their base research with some such as Park, Seo, Han, Oh and Lee (2018) using some or all of the dataset.

Additional goals of this research related to datasets include using one based on real apps, as opposed to contrived or synthetic datasets, plus using a dataset that is available to other researchers for download. Both of the datasets in Table 15 meet those requirements. Given all of the above, Andro-AutoPsy was selected as the dataset for this research, mainly due to the quality and sample size of malware.

Table 15
Dataset Options

Dataset	Source	Total Samples	Malware Samples
Drebin Dataset	University of Gottingen, Germany	129,013	5,560
Andro-AutoPsy	University of Korea Hacking and Countermeasure Research Lab	119,183	9,990

Chapter 4

Results

Data Description

As described in the section Datasets, the raw dataset consisted of 119,183 Android applications in the form of APK files. Of those, 9,990 were malware leaving 109,193 as benign, or 8.4% malware overall (Table 15). Each APK was decompiled and Android Permissions, Intents and API Calls were extracted from the Java source code and the manifest XML file. The collection totaled over 500K Permissions, 800K Intents and 3.7M API calls.

At the time the experiments were run in January 2020, there were 158 Permissions defined in Android, 295 Intents and innumerable Java API calls available. When those three feature groups were extracted from all the APKs, only 66 distinct Permissions of the 158 defined were found used. Likewise, 139 distinct Intents of 295 were found, and a total of 203 Android Java API calls were identified. Thus, the resulting list of attributes consisted of 408 distinct Android features across the 119K Android apps. Table 16 presents an updated version of Table 12 with the observed feature counts. These numbers do not include the three features of contrast variables added as part of the performance selection phase.

As described earlier, in order to simplify data management, all features were given unique identifiers. These IDs and the feature names are presented in Appendix B.

Table 16
Datasets Test Matrix - Observed Feature Count

DS #	Dataset Features Types			DS ID	Count*
1	Permissions	Intents	API Calls	PIA	408
2	Permissions	Intents		PI	205
3	Permissions		API Calls	PA	269
4		Intents	API Calls	IA	342
5	Permissions			P	66
6		Intents		I	139
7			API Calls	A	203

* *Observed*

Feature Subset Selection Process

The feature selection process was described previously in the section Feature Subset Selection as consisting of four phases as shown in Figure 5. After obtaining *all features*, as listed in Table 16, the second step, *algorithmic selection*, consisted of running each of the eleven different feature selection methods against the seven feature type combinations. As defined in the earlier referenced section, the maximum possible feature subsets at this stage was: $\mathcal{F}^{m \times r}$ where $m = 7$ (dataset combinations) and $r = 11$ (feature selection techniques), or 77 feature subsets. The inclusion of the term *maximum* in the definition was due to the possible circumstance of some feature subsets being the same. Upon completion of the experiments creating the feature subsets, it was observed that there were no duplicates, thus, phase two in the process did result in 77 distinct feature subsets.

Also note that all algorithms were set to provide the maximum number of feature weights. For methods such as Information Gain that implies that if 408 features go into the process, one will get 408 feature weights out. However, other algorithms eliminate

features as part of their internal calculations and return only the features the algorithm calculated to be significant. As an example, CFS has a stepwise process calculating a metric defined as *merit* per feature. Each feature with a merit value equal to or greater than the previous is assumed significant. The first time a feature's merit value is less than the previous, the algorithm assumes it has determined all the significant features and the process stops. So if one were to use 408 features as input to CFS, it might return 10, 50 or 100 as significant, but most likely some number less than 408.

The feature counts for each algorithm-dataset combination from the experiments are presented in Table 17 by rows and columns respectively. Comparing the counts to the count of all features in Table 16 one can see that of the eleven feature selection algorithms, seven actually eliminated features, while four (Information Gain, Chi-Square, RFE - Support Vector Machine and Ridge Regression) provided feature weights for all features. As an example, the dataset of all three attribute types (PIA) contained a feature count of 408, as described in row one of Table 16. After algorithmic selection only the

Table 17
Phase 2 - Feature Count by Dataset and Feature Category.

FS Method	PIA	PI	PA	IA	P	I	A
InfoGain	408	205	269	342	66	139	203
Chi2	408	205	269	342	66	139	203
Relief	282	128	190	244	40	88	151
CFS-InfoGain	14	11	13	9	9	4	8
CFS-Chi2	11	10	11	6	9	4	9
CFS-Relief	14	17	10	10	13	15	9
RFE-NN	387	182	269	321	65	119	203
RFE-RF	382	191	268	319	66	132	203
RFE-SVM	408	205	269	342	66	139	203
LASSO	281	132	238	252	59	86	187
Ridge	408	205	269	342	66	139	203

subsets from Information Gain, Chi-Square, RFE-SVM and Ridge Regression still contain 408 features (first data column of Table 17).

The next phase of subset selection, *performance selection*, eliminated attributes based on feature significance. Table 18 presents the feature counts after phase three. In the table, each dataset combination now has two columns, one showing the feature counts after algorithmic selection (n_{AS}) as reported in Table 17, and a second column showing feature count after performance selection (n_{PS}).

With respect to the C1 cohort defined earlier, significant features are features with weight values greater than $\overline{w^V}$. Using Information Gain and the PIA dataset as an example, of the 408 features weighted by the Information Gain algorithm, 29 features had weights less than or equal to $\overline{w^V}$ and thus were eliminated. Or in other words, 379 features were selected by the performance selection process as significant.

For the C2 cohort and the three variations in outcomes, the first is where none of

Table 18
Phase 3 - Feature Count by Dataset and Feature Category.

FS Method	PIA		PI		PA		IA		P		I		A	
	n_{AS}	n_{PS}	n_{AS}	n_{PS}	n_{AS}	n_{PS}	n_{AS}	n_{PS}	n_{AS}	n_{PS}	n_{AS}	n_{PS}	n_{AS}	n_{PS}
InfoGain	408	379	205	172	269	257	342	321	66	65	139	111	203	203
Chi2	408	378	205	165	269	262	342	307	66	61	139	104	203	203
Relief	282	0	128	0	190	2	244	1	40	5	88	8	151	1
CFS-InfoGain	14	14	11	11	13	13	9	9	9	9	4	4	8	8
CFS-Chi2	11	11	10	10	11	11	6	6	9	9	4	4	9	9
CFS-Relief	14	3	17	3	10	4	10	3	13	13	15	15	9	5
RFE-NN	387	339	182	158	269	264	321	301	65	62	119	102	203	199
RFE-RF	382	146	191	44	268	118	319	114	66	22	132	19	203	90
RFE-SVM	408	213	205	126	269	185	342	55	66	39	139	23	203	92
LASSO	281	231	132	124	238	217	252	232	59	59	86	85	187	177
Ridge	408	343	205	164	269	261	342	295	66	65	139	111	203	200

the feature weights are greater than $\overline{\mathbf{w}^{\mathcal{V}}}$. An example of that outcome is Relief and the PIA dataset where Relief determined 282 features to be significant, but also rated the random data ($\overline{\mathbf{w}^{\mathcal{V}}}$) at the same level.

The second scenario is the opposite, where all of the feature weights from the algorithmic selection process are greater than $\overline{\mathbf{w}^{\mathcal{V}}}$. An example of this outcome is CFS - Information Gain and the PIA dataset. The CFS process selected 14 features as significant and all of those were greater than $\overline{\mathbf{w}^{\mathcal{V}}}$.

The third outcome is a mix of these two extremes where some of the feature weights are greater than $\overline{\mathbf{w}^{\mathcal{V}}}$ and some are not. An example of that situation is RFE - Random Forest where the RFE process selected 382 features as significant, but only 146 of those had weights greater than $\overline{\mathbf{w}^{\mathcal{V}}}$.

Based on all the feature counts presented in Table 18, we can now calculate the algorithm effectiveness as defined earlier in equation (8). For the eleven feature selection methods and seven database combinations, the algorithm effectiveness is presented in Table 19. It is easy to notice that with two of the filter-multivariate methods, CFS –

Table 19
Feature Selection Algorithm Effectiveness.

FS Method	PIA	PI	PA	IA	P	I	A
InfoGain	93%	84%	96%	94%	98%	80%	100%
Chi2	93%	80%	97%	90%	92%	75%	100%
Relief	0%	0%	1%	0%	13%	9%	1%
CFS-InfoGain	100%	100%	100%	100%	100%	100%	100%
CFS-Chi2	100%	100%	100%	100%	100%	100%	100%
CFS-Relief	21%	18%	40%	30%	100%	100%	56%
RFE-NN	88%	87%	98%	94%	95%	86%	98%
RFE-RF	38%	23%	44%	36%	33%	14%	44%
RFE-SVM	52%	61%	69%	16%	59%	17%	45%
LASSO	82%	94%	91%	92%	100%	99%	95%
Ridge	84%	80%	97%	86%	98%	80%	99%

Information Gain and CFS – Chi-Square, are rated at the top performers at 100%. Also notice the poor performance of the filter-univariate method, Relief, where all or very nearly all features it selected were deemed not significant.

Table 20 removes the dataset segregation and shows the effectiveness for each feature selection algorithm for the total of all datasets, sorted best to worst. As expected from the previous table, CFS – Information Gain and CFS – Chi-Square are rated as the most effective algorithms and Relief was the poorest.

The same data aggregated by feature selection type is presented in Table 21, unsorted. Due to the extremely poor performance of Relief, the third column provides the data without inclusion of the Relief performance numbers. First notice that the worst performers are the wrapper methods. However, the best performers change in ranking based on the inclusion or exclusion of Relief. Given the few representative methods (three of each filter method and two embedded methods) one would have to consider this data inconclusive in terms of performance ranking by these group types.

Table 20
Overall Algorithm
Effectiveness.

FS Method	Effectiveness
CFS-InfoGain	100%
CFS-Chi2	100%
InfoGain	92%
RFE-NN	92%
LASSO	91%
Chi2	91%
Ridge	88%
CFS-Relief	52%
RFE-SVM	45%
RFE-RF	35%
Relief	2%

Table 21
Effectiveness by Feature Selection Type.

FS Method Type	Effectiveness	Effectiveness*
Filter - Univariate	68%	92%
Filter- Multivariate	81%	100%
Wrapper	57%	57%
Embedded	89%	65%

* Excluding Relief

A secondary measure that should be considered when comparing various algorithms is the time it takes to compute the results. Table 22 presents the computational time required for each feature selection algorithm for the total of all datasets, sorted fastest to slowest. The time is shown in minutes, but the third column is a normalized version which abstracts the time results from the performance of the specific computer system used in the experiments, which did not vary throughout the process. One could view this as three groupings: seven of the methods are 1% each, then there is

Table 22
Compute Time per Feature Selection Method.

FS Method	Time (min)	% of Max
Chi2	3.4	1%
Ridge	3.7	1%
RFE-NN	3.7	1%
RFE-RF	4.1	1%
LASSO	4.2	1%
CFS-Chi2	4.9	1%
InfoGain	5.1	1%
Relief	73.5	17%
CFS-Relief	83.6	20%
RFE-SVM	253.2	60%
CFS-InfoGain	424.2	100%

both versions of Relief accounting for roughly 20% each, and then the outliers of RFE – Support Vector Machine and CFS – Information Gain. Considering the effectiveness and speed together (Table 20 and Table 22 respectively) one would be tempted to label CFS - Chi-Square as the best performing feature selection method. However, another point to consider is the number of attributes selected by each method. In Table 18 it is obvious that the three CFS methods select considerably fewer features than most of the others. And while most if not all of the CFS algorithmic selections did pass the performance selection step, it does indicate that in situations where more features would improve classifier accuracy, CFS could be insufficient.

For completeness, Table 23 presents the compute time aggregated by feature selection type. It is a bit surprising that Wrapper methods were not the slowest, given they are well-known for being so given the nature of the designed, i.e., a selection algorithm wrapped around a machine learning classifier.

Ranking Features

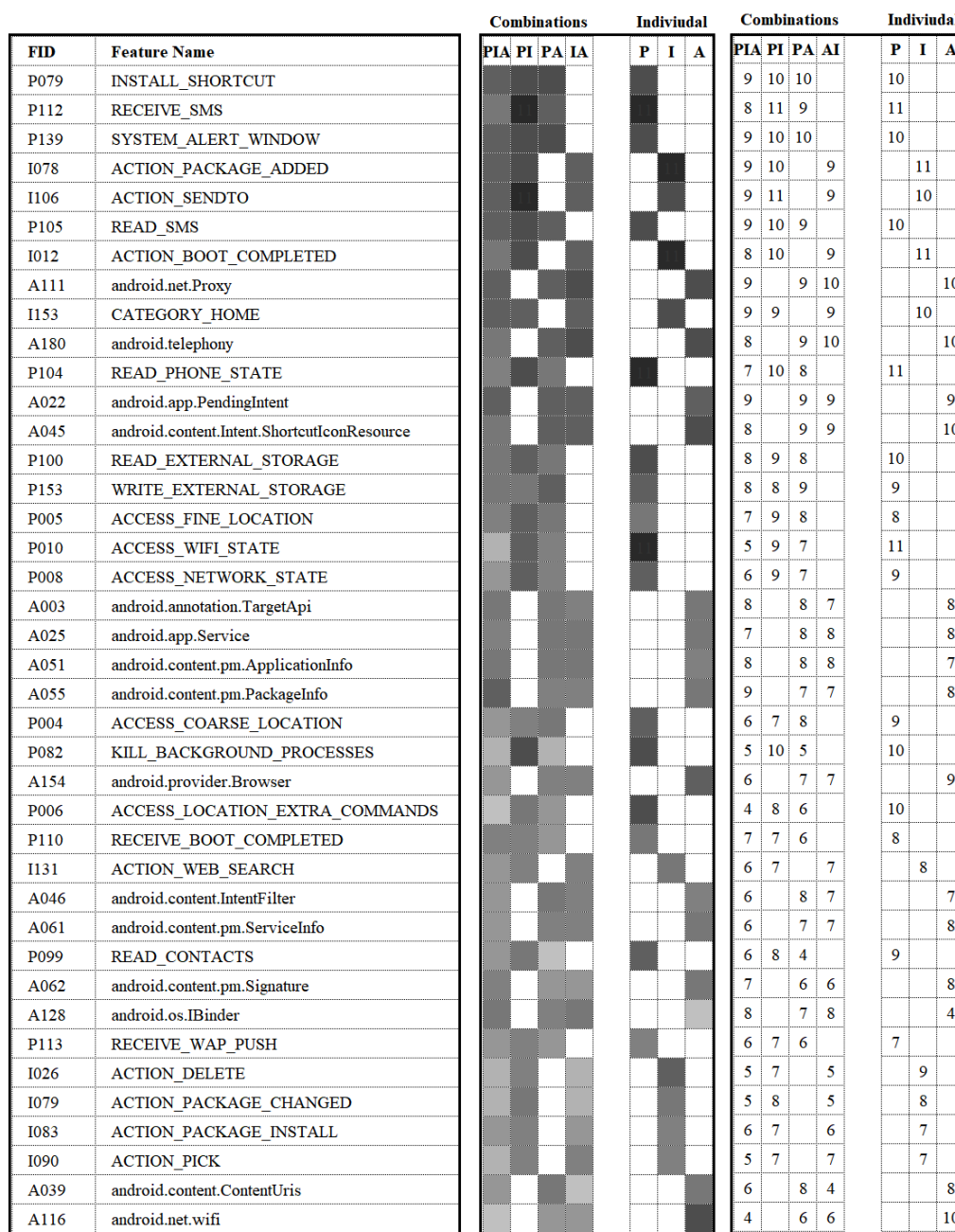
In terms of results from the experiments, a visual representation of ρ_{di} is shown in Figure 11 as a heatmap of the top feature rankings, segmented by feature groups, i.e., dataset combinations. The shading is based on the value of ρ_{di} for the specific feature.

Table 23
Compute Time per Feature Selection Type.

FS Method Type	Time (min)	% of Max
Filter - Univariate	82.0	16%
Filter- Multivariate	512.7	100%
Wrapper	261.0	51%
Embedded	7.8	2%

In the presentation that means a lighter shade indicates lower importance and darker is higher importance. The columns represent the seven different combinations of feature types: Permissions, Intents and API Calls.

Figure 11
Heatmap of Feature Ranking by Dataset

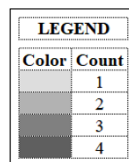
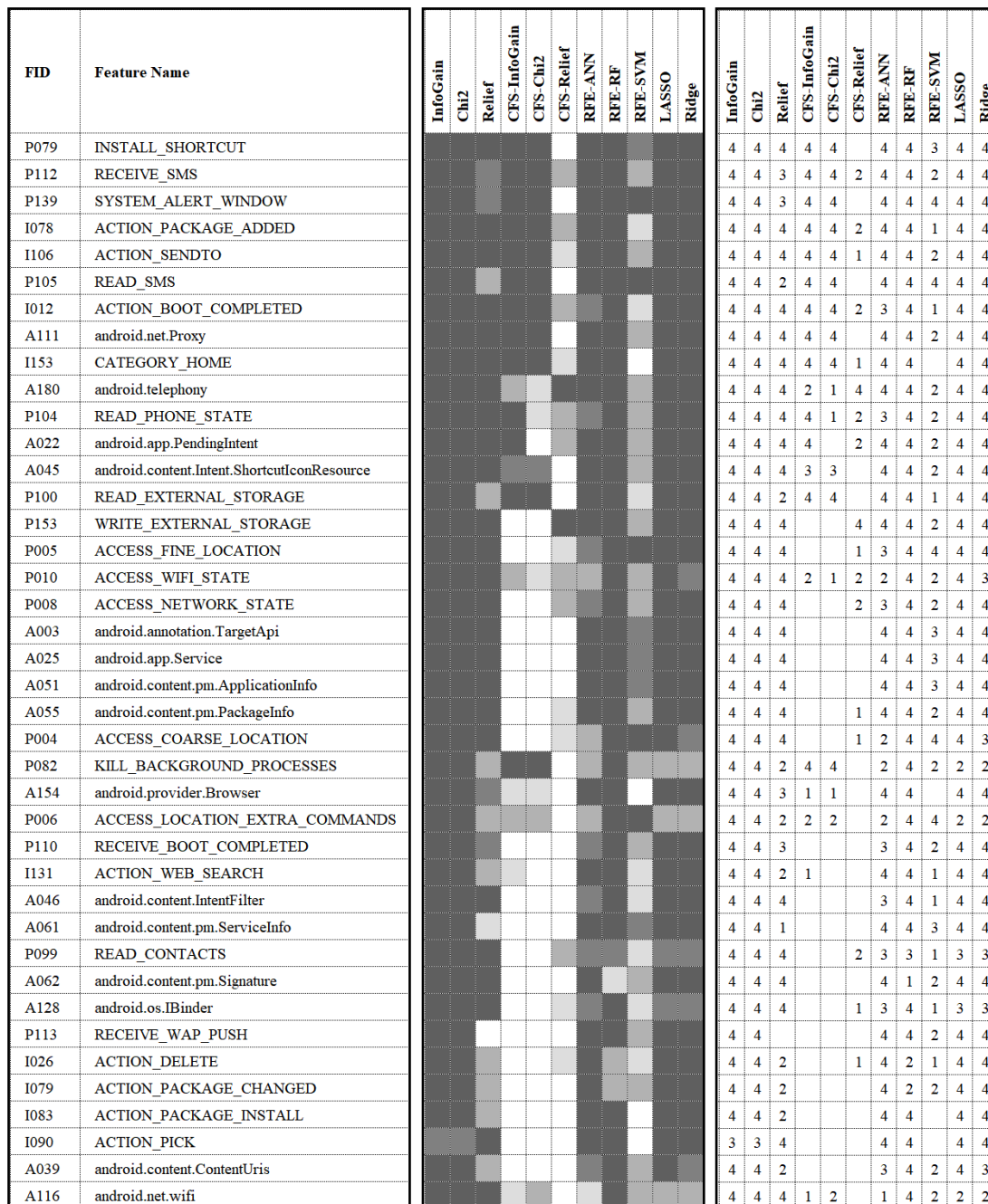


Color		Count	Abv	Dataset Feature Groups
		1	PIA	Permissions - Intents - APIs
		2	PI	Permissions - Intents
		3	PA	Permissions - APIs
		4	IA	Intents - APIs
		5	P	Permissions
		6	I	Intents
		7	A	APIs
		8		
		9		
		10		
		11		

The blank cells are datapoints where that feature was not available for ranking. For example, the top item, Permission P079, `INSTALL_SHORTCUT`, was not in the datasets of IA (Intents and API calls), I (Intents only) or A (API calls only) because there were no Permissions in those datasets by definition. Appendix C contains the numerical data for all features.

Figure 12 is a similar heatmap but of $\rho_{\tau i}$ which shows the top feature rankings segmented by feature selection algorithm. The columns represent the eleven different selection methods used. In this representation, the blank cells are datapoints where the feature selection algorithm eliminated that feature and therefore did not assign a weight. For example, the top item, Permission P079, `INSTALL_SHORTCUT`, was not in the features selected by the CFS-Relief method for any of the dataset combinations. Appendix D contains the numerical data for all features.

Figure 12
Heatmap of Feature Ranking by Feature Selection Algorithm



With feature ranking complete, the information is now available to address the first two research questions.

Research Question 1

How does feature ranking vary when Permissions, Intents and API Calls are selected separately versus combined?

To conduct this analysis, ranking of the features based on singular datasets (Permissions only, Intents only and API Calls only, datasets #5, #6 and #7 respectively in Table 12) is compared to rankings based on the combination of all datasets (dataset #1). By definition the combination set contains Permissions, Intents and API Calls, so in order to make the comparison, for each feature type, the other two features are removed from the combination set and it is contracted. As an example, P008 is the 18th most significant feature, as shown in Figure 11, but becomes the 10th most significant feature when Intents and API Calls are removed from the combination set.

If a null hypothesis were true, *there is no effect on ranking when comparing sequences*, then the two ranking vectors would be equal. Looking at simply the top ten for Permissions in each from the experimental results we can observe that:

$$\langle \pi_{11}, \pi_{12}, \dots, \pi_{1,10} \rangle = \langle P079, P112, P139, P105, P104, P100, P153, P005, P010, P008 \rangle$$

$$\langle \pi_{51}, \pi_{52}, \dots, \pi_{5,10} \rangle = \langle P105, P079, P139, P113, P153, P112, P104, P005, P004, P100 \rangle$$

so that clearly

$$\langle \pi_{11}, \pi_{12}, \dots, \pi_{1,10} \rangle \neq \langle \pi_{51}, \pi_{52}, \dots, \pi_{5,10} \rangle$$

and thus, more generally

$$\langle \pi_{11}, \pi_{12}, \dots, \pi_{1,k^F} \rangle \neq \langle \pi_{51}, \pi_{52}, \dots, \pi_{5,k^F} \rangle$$

and expanding to the other two feature types we can show

$$\langle \pi_{11}, \pi_{12}, \dots, \pi_{1,k^F} \rangle \neq \langle \pi_{61}, \pi_{62}, \dots, \pi_{6,k^F} \rangle$$

and

$$\langle \pi_{11}, \pi_{12}, \dots, \pi_{1,k^F} \rangle \neq \langle \pi_{71}, \pi_{72}, \dots, \pi_{7,k^F} \rangle$$

The null hypothesis is false.

To quantify the effect, a similarity scoring function is required that can show similarity as a sequence. One approach would be a binary scoring referred to as the Hamming distance in information theory. It involves comparing the i^{th} elements of each vector, but such a naïve comparison would lead to a conclusion that the two are mostly dissimilar, which is not the case.

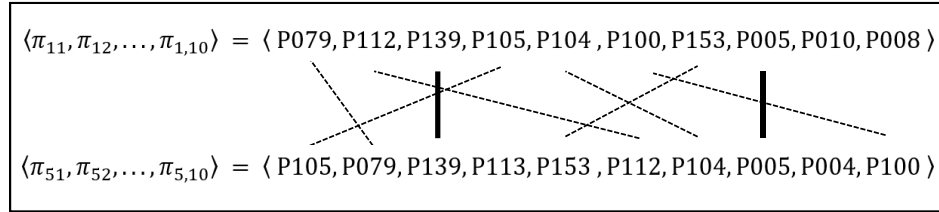
Consider again $\langle \pi_{11}, \pi_{12}, \dots, \pi_{1,10} \rangle$ and $\langle \pi_{51}, \pi_{52}, \dots, \pi_{5,10} \rangle$. As shown in Figure 13, a binary i^{th} -based comparison would indicate only two of the ten to be a match, P139 in position three and P005 in position eight. This would yield a similarity score of 20% considering the top 10 set. Yet when looking at the union of the set, 8 of the 10 elements of each are in common. Figure 14 shows the example with the original two matches plus the additional six. This would yield a similarity score of 80%. In a set of size 66, such as Permissions, one would conclude that if 8 of the top 10 are the same, then there is certainly some similarity.

Figure 13

Combined to Single Set Membership Example - Exact Match

$\langle \pi_{11}, \pi_{12}, \dots, \pi_{1,10} \rangle =$	\langle	P079,	P112,	P139,	P105,	P104,	P100,	P153,	P005,	P010,	P008	\rangle
$\langle \pi_{51}, \pi_{52}, \dots, \pi_{5,10} \rangle =$	\langle	P105,	P079,	P139,	P113,	P153,	P112,	P104,	P005,	P004,	P100	\rangle
binary scoring		0	0	1	0	0	0	0	1	0	0	

Figure 14
Combined to Single Set Membership Example - Union



However, there are limits to a strict set union analysis given that the two overall sets are actually equal, it is only the sequence that varies. What is needed is a string comparison technique that goes beyond evaluating similarity of elements contained in the sets, but that applies a penalty if the like elements are not in the same position. The Euclidean distance method provides just such a function, allowing the assumption that the sets are equal and calculating a similarity score based on how close the same elements are in position. For comparing $\pi(d_1i)$ to $\pi(d_5i)$, $\pi(d_6i)$ and $\pi(d_7i)$, the Euclidean distance, σ , is defined as:

$$\sigma_{1,5} = |\pi(d_1i) - \pi(d_5i)|,$$

$$\sigma_{1,6} = |\pi(d_1i) - \pi(d_6i)| \text{ and}$$

$$\sigma_{1,7} = |\pi(d_1i) - \pi(d_7i)|.$$

The reverse, for comparing to $\pi(d_5i)$, $\pi(d_6i)$ and $\pi(d_7i)$ to $\pi(d_1i)$ is defined as:

$$\sigma_{5,1} = |\pi(d_5i) - \pi(d_1i)|,$$

$$\sigma_{6,1} = |\pi(d_6i) - \pi(d_1i)| \text{ and}$$

$$\sigma_{7,1} = |\pi(d_7i) - \pi(d_1i)|,$$

In order to achieve a cross-comparison for understanding the overall difference between two result sets, we take an average of the two distances per comparison. The final Euclidean distance is then: :

dissimilar objects, usually accomplished by taking the inverse of the distance measures. In addition, for the overall similarity score, being normalized would abstract the number of features per type from the scoring vector and make comparison among feature types more meaningful. We thus define the similarity score, γ , describing the feature by element as:

$$\gamma(\pi_f) = 1 - \frac{\sigma(\pi_f)}{n_f}$$

where n_f is the number of attributes in each feature type. The similarity score for the features set overall is then:

$$\gamma_f = \overline{\gamma(\pi_f)}$$

For the above concrete example γ_p is undefined due to the distance measurements extending beyond the top 10, such as P010 which is in the top 10 of the combined sequence, but not in the top 10 of the individual sequence.

Moving beyond the first ten elements of the example and applying to the entire Permissions attribute set leads to a $\gamma_p = 92\%$ with an element-wise distribution presented in Figure 16. In the chart, the element-by-element score is the dotted line. Given the noise evident in the score line, a smoothing function (the dark line) is applied in order to provide a better visualization of the result. The smoothing function is simply the average of the surrounding cells, in this case the average encompasses 5% of the cells on each side of a point.

Figure 16

Similarity of Permissions: Combined Dataset vs Permissions Only Dataset

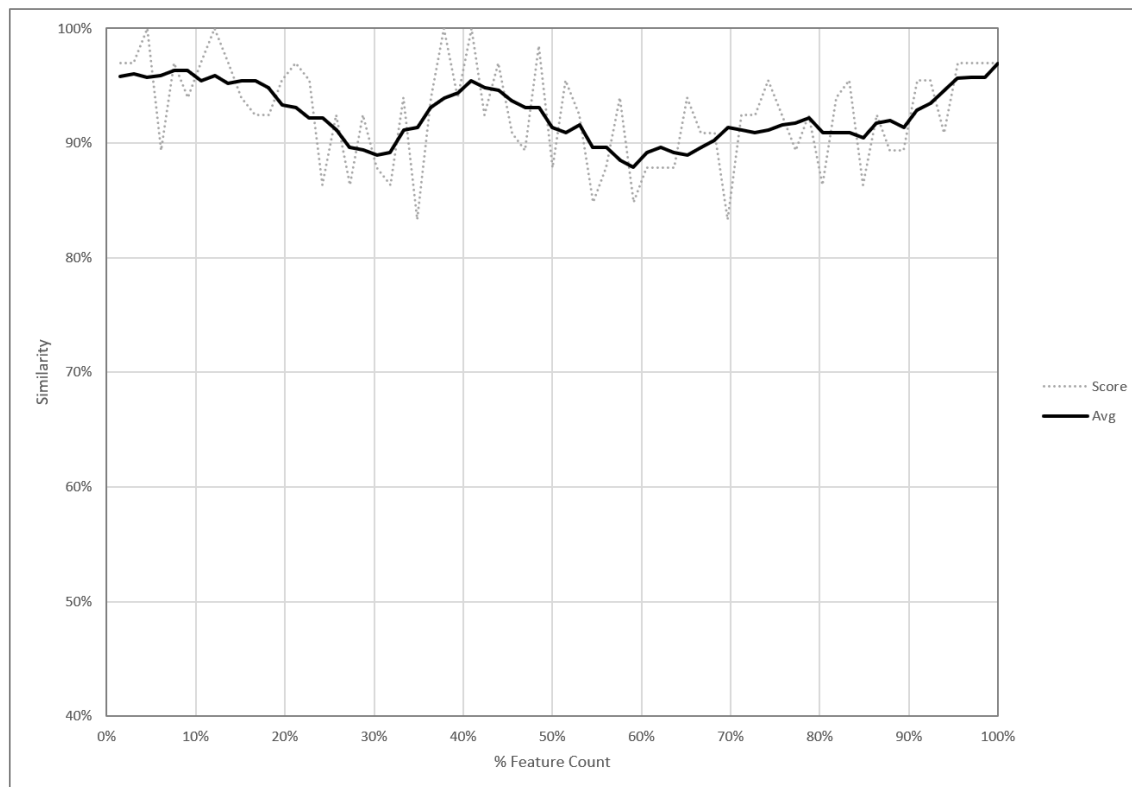
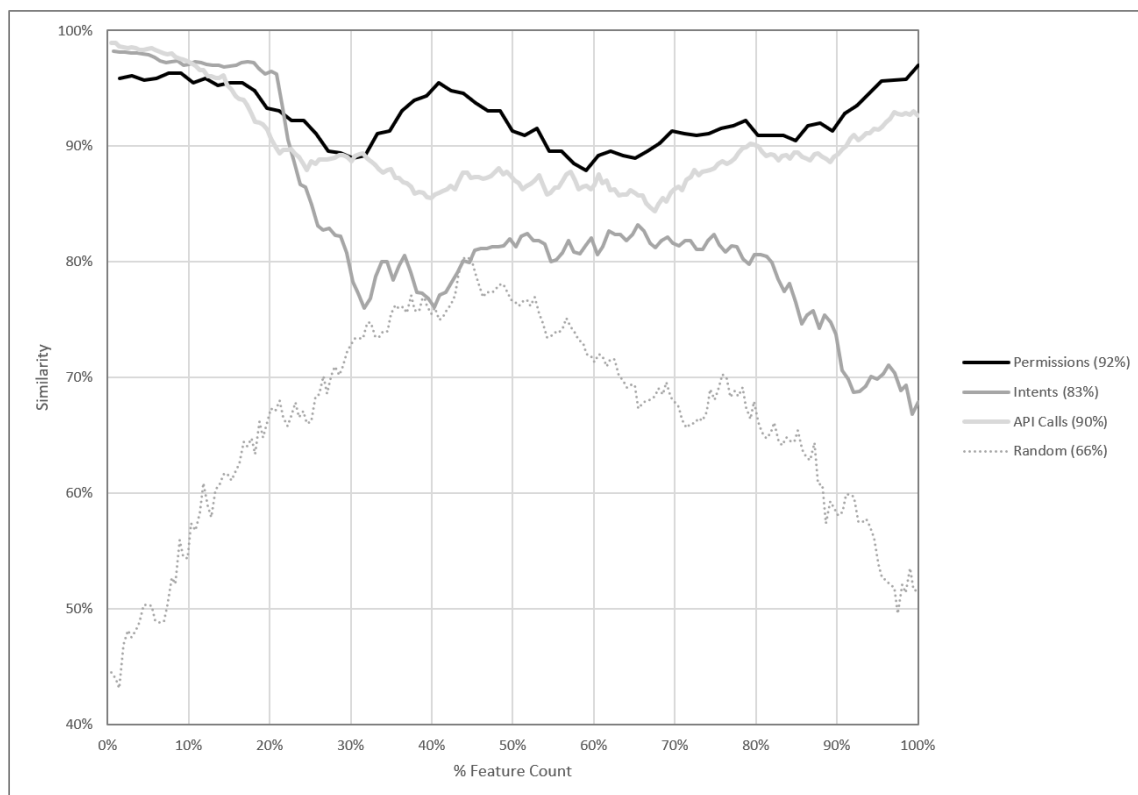


Figure 17 combines the same analysis with that of Intents and API Calls, plus a Random function to show as contrast. The chart indicates that there is minimal effect of getting feature ranking from the combined dataset versus the singular datasets when considering the top 10 – 15 percent for all three attribute types. This means that if one wants a feature set of Permissions, Intents and API Calls, and the desired feature count is small, then it does not matter if the feature selection process is performed with all three feature groups combined, or if selection is performed separately and then combined.

However, from around 15 – 25 percent there is a sharp divergence in the similarity. Intents go from mid-90s to around 80% and eventually below, operating in the same range as the random function. Both Permissions and API Calls oscillate in the

Figure 17

Similarity of Feature Types: Combined Dataset vs Singular Only Datasets



upper 80s to lower 90s range with no obvious pattern. The overall similarity of the three vectors, combined datasets to singular datasets, are shown in the legend of the figure, and are: Permissions = 92%, Intents = 83% and API Calls = 90%.

The implication of this data is that during the feature selection phase on the Android platform, researchers must be cognizant of the effect of layering multiple feature types in their dataset, with the detrimental effects getting worse the larger the feature set employed. As an example, suppose someone wants to have a dataset of Permissions and Intents with an attribute count of around 100. If feature selection is performed separately to get the most important Permissions to use and then to get the most important Intents,

the feature set could be significantly different than if the Permissions and Intents were combined and then feature selection performed.

Given the plethora of small sample sizes used in this research stream for convenience, a poor selection of the features could dramatically change the end results of classification.

Research Question 2

How does feature ranking vary across feature selection algorithms?

Similar to the prior question, this analysis compares the feature rankings from each of the 11 feature selection algorithms, to the ranking from the combination of all features. If there were no effect of feature selection method on feature ranking (the null hypothesis) then all permutations would be equal. Table 24 presents the top ten features for each of the eleven feature selection methods. Through simple observation one can see the null hypothesis is false. The closest in similarity are the first two methods, Information Gain and Chi-Square, where the sequences of the first five elements are the same, but other than those two segments, the table indicates a lot of diversity.

Table 24
Top10 Selected Features by Feature Selection Method

		Top 10 selected elements									
		1	2	3	4	5	6	7	8	9	10
Feature Selection Method	1	I078	P105	P079	I153	P112	A021	I012	P104	A022	A045
	2	I078	P105	P079	I153	P112	A045	I012	A111	I106	I083
	3	A180	P153	P104	A151	A010	A040	A067	I105	A066	A030
	4	A004	I106	A116	A180	P112	P100	P010	P082	I012	I153
	5	A004	P104	A021	P010	A116	I012	I078	A154	I106	P079
	6	A180	P104	I105	P153	P148	A125	A066	I078	I153	I101
	7	I078	P113	A180	A061	I011	P079	I131	I044	P139	A154
	8	I078	P105	P112	P079	I153	I012	P104	A045	I106	A111
	9	A069	P004	P005	P006	P067	P105	P053	A070	A139	P012
	10	A180	P113	I078	I131	P112	I011	I086	P079	P111	A061
	11	A180	I078	P113	I131	P079	P112	I086	A061	A154	P105

In order to evaluate how the methods vary, the ranking based on the combination of all methods, π_C , will be used as a standard for comparison. As with dataset evaluation, a similarity score based on Euclidean distance will be the base measurement.

The first comparison of methods is with the three univariate filter methods: Information Gain, Chi-Square and Relief. Figure 18 shows the similarity among the three. Notice the remarkable consistency between Information Gain and Chi-Square implying that these methods are essentially interchangeable for any researchers considering choosing one of the two. Also note that because they are strictly correlational, they provide rankings for the entire feature set whereas Relief only provides rankings for those features deemed significant by the algorithm, in this case, completing execution around 72% of Feature Count (x-axis).

Figure 18
Similarity of Univariate Filter Methods

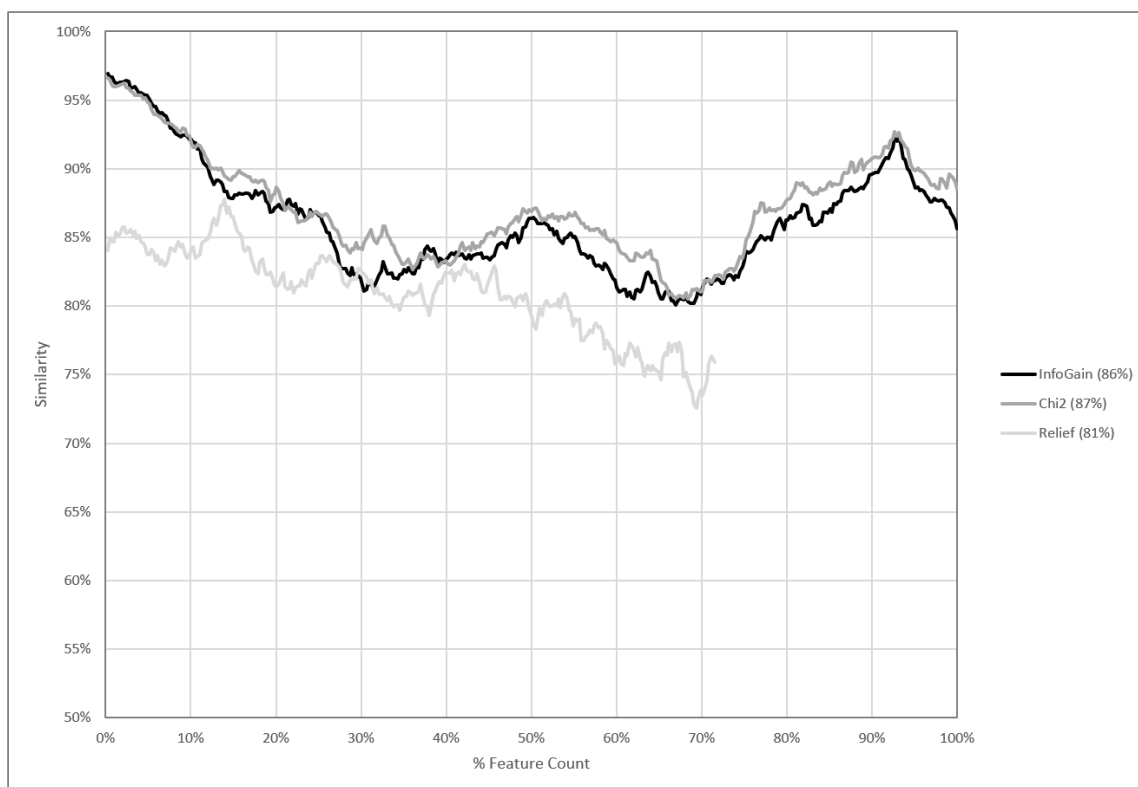
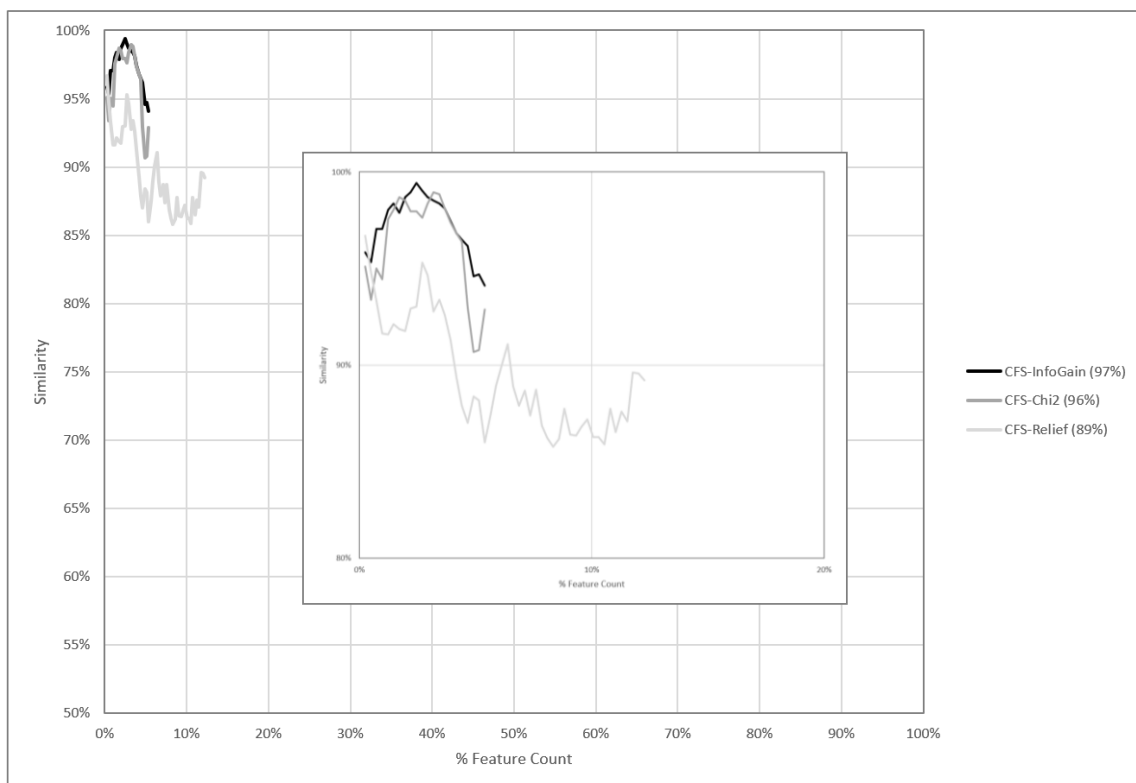


Figure 19 looks at the three multivariate filter methods: CFS – Information Gain, CFS – Chi-Square and CFS – Relief. The base chart presents the data at the same scale as all the other charts in this sequence, however, due to the limited coverage over the x-axis, an overlay is included in the figure that magnifies the area of interest.

Recall that the CFS algorithm starts with a correlation input and the three CFS methods employed here are using the same three univariate methods presented as stand-alone methods in the prior section. Therefore, one would expect some level of consistency between the stand-alone results and the CFS results.

As easily observed on the figure overlay, the consistency of CFS – Information Gain and CFS – Chi-Square is high. Also notice that both methods complete execution at

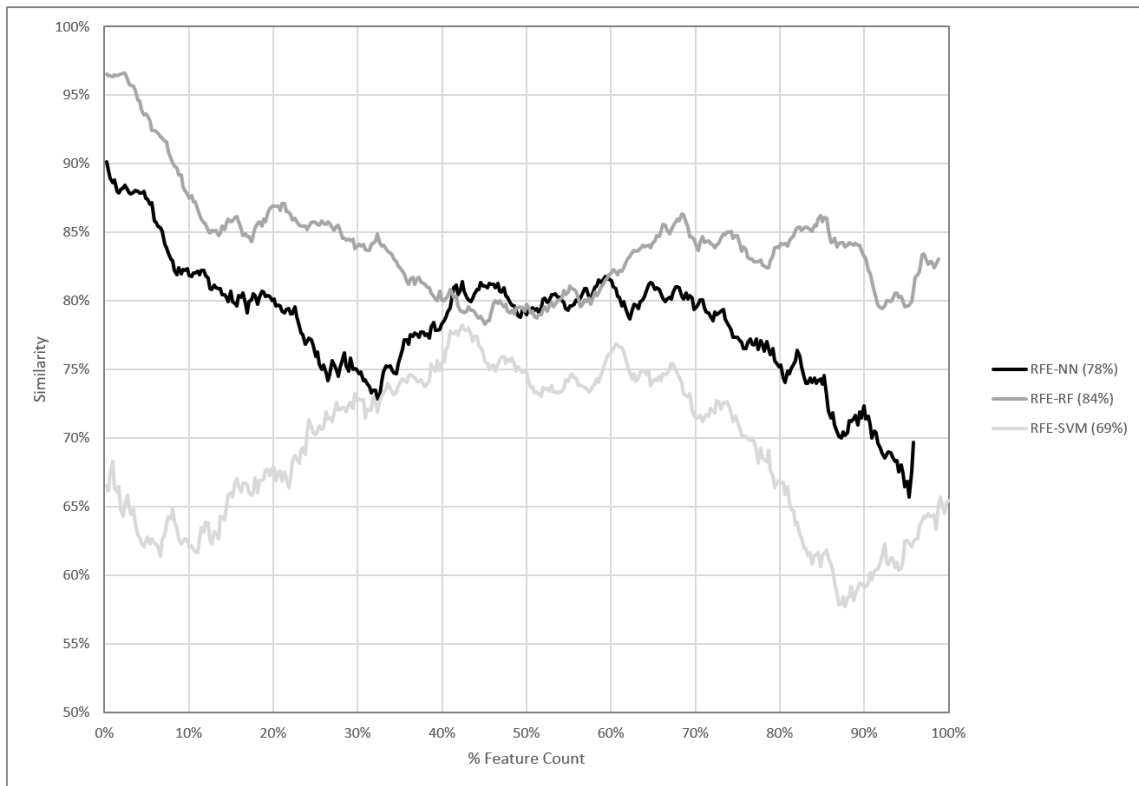
Figure 19
Similarity of Multivariate Filter Methods



just over 5% of feature count. In other words, out of 408 attributes, these two CFS methods returned 22 attributes as important. CFS – Relief diverges from the other two as with the univariate analysis, but returns 50 attributes as important, more than double the other two CFS methods.

Similarity of the three wrapper methods, RFE – Neural Net, RFE – Random Forest and RFE – Support Vector Machine, is presented in Figure 20. While there is some consistency between the first two in the first half of the feature set, overall, all three vary significantly. Because these wrapper methods use classifiers as their search engine, the classifiers may perform differently based on the domain. In a following section,

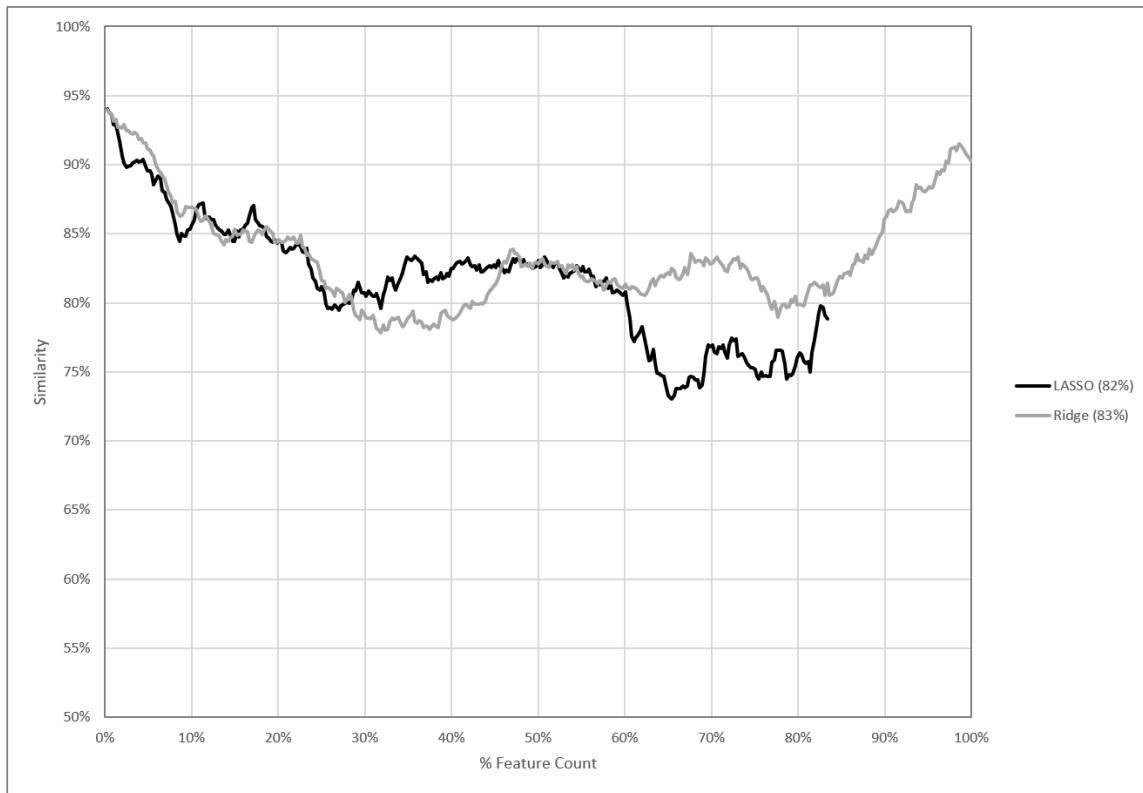
Figure 20
Similarity of Wrapper Methods



validating the feature sets with classifiers will shed light on this variation and its effect on accuracy.

Finally, the last grouping, embedded methods, consisting of LASSO Regression and Ridge Regression, are shown in Figure 21. We see that as expected, LASSO does select features, i.e., it does not provide a weighting for all features but only the ones the algorithm selects as significant. As shown in the chart, it completes execution at around 85% of feature count. Prior to that point, up to around 60% there is a lot of consistency between the two methods. Referring back to compute time in Table 22, the difference in that respect is minimal between the two, as well as with several other methods.

Figure 21
Similarity of Embedded Methods



Researchers trying to choose between L1 and L2 regression should see little impact between the two unless aiming for the maximum size attribute count.

The similarity of all eleven feature selection methods is presented in Figure 22. This chart is comprised of the same data presented in the previous four charts, all at the same scale. The volume of data makes it somewhat challenging to discern individual lines but notice the shading in the upper left quadrant. Ten out of eleven of the methods show a remarkable consistency given the variety of algorithms. This area shows that compared to the baseline measurement, they all start at 90% similar $\pm 5\%$ trending down to 80% similar $\pm 5\%$ approaching one third of the attribute set count. This shows that the larger the feature set size, the greater the impact based on the feature selection method.

Figure 22
Similarity of All Eleven Feature Selection Methods



Data Repeatability

When basing conclusions on experimental data, it is useful to understand the repeatability of the numbers, especially when there are outliers as RFE-SVM is in the previous figure. Such outliers always beg the question as to if the data is bad or if it is a valid phenomenon.

Repeatability was investigated for the eleven feature selection methods discussed in the previous section. In that it is the method to be proven, and not the effect of the data on the method, only one of the seven dataset combinations was chosen for the investigation, the Permissions only version. The assumption is that any variances in repeatability for one dataset would manifest itself in all seven datasets.

For each feature selection method, using the Permissions only dataset, the experiment was run ten times and the feature rankings compared. In a perfect scenario, the same method would provide the same feature ranking ten times in a row. For example, if Information Gain ranked P010 (ACCESS_WIFI_STATE) as the fifth most important Permission, the question is, would it rank it fifth every time, or could some variation in the process cause it to get ranked fourth or sixth, or some other ranking, during certain runs?

The answer is that it varies depending on the feature selection method. To analyze the effect, the standard deviation is calculated for each method for each significant feature, i.e., post-Performance Selection. Table 25 offers three examples of the calculations. The first column shows that Information Gain ranked feature P010 as fifth most important all ten iterations, thus the standard deviation is zero. The second column shows that the Relief method ranked P104 either first or second for all iterations resulting in a standard deviation of 0.52. The third column shows a much less desirable outcome with RFE – Neural Network ranking P043 in a range from 25 to 43 with a standard deviation of 6.72.

Table 25
Selected Repeatability Data

Iteration	InfoGain	Relief	RFE-NN
	P010	P104	P043
1	5	2	36
2	5	1	29
3	5	1	32
4	5	2	40
5	5	2	37
6	5	1	25
7	5	2	28
8	5	1	43
9	5	1	27
10	5	1	43
Std Dev	0	0.52	6.72

Figure 23 is a graph of all standard deviations for all eleven feature selection methods. Note that not all lines cover the full feature rank range. This is because the standard deviation is only relevant for features that were selected by the process. Refer back to Table 18 for the counts associated with each. As easily determined from the figure, RFE – Neural Network is the poorest performer in terms of data repeatability. Likewise, Relief and CFS – Relief shows relatively high standard deviations especially considering the few number of features selected by each.

Table 26 provides the average standard deviation for all methods. Note that the top four include Information Gain, Chi-Square and then the CFS version of each. Of those four, only Information Gain was not perfect repeatability, although it was perfect

Figure 23
Data Repeatability for Feature Selection Methods

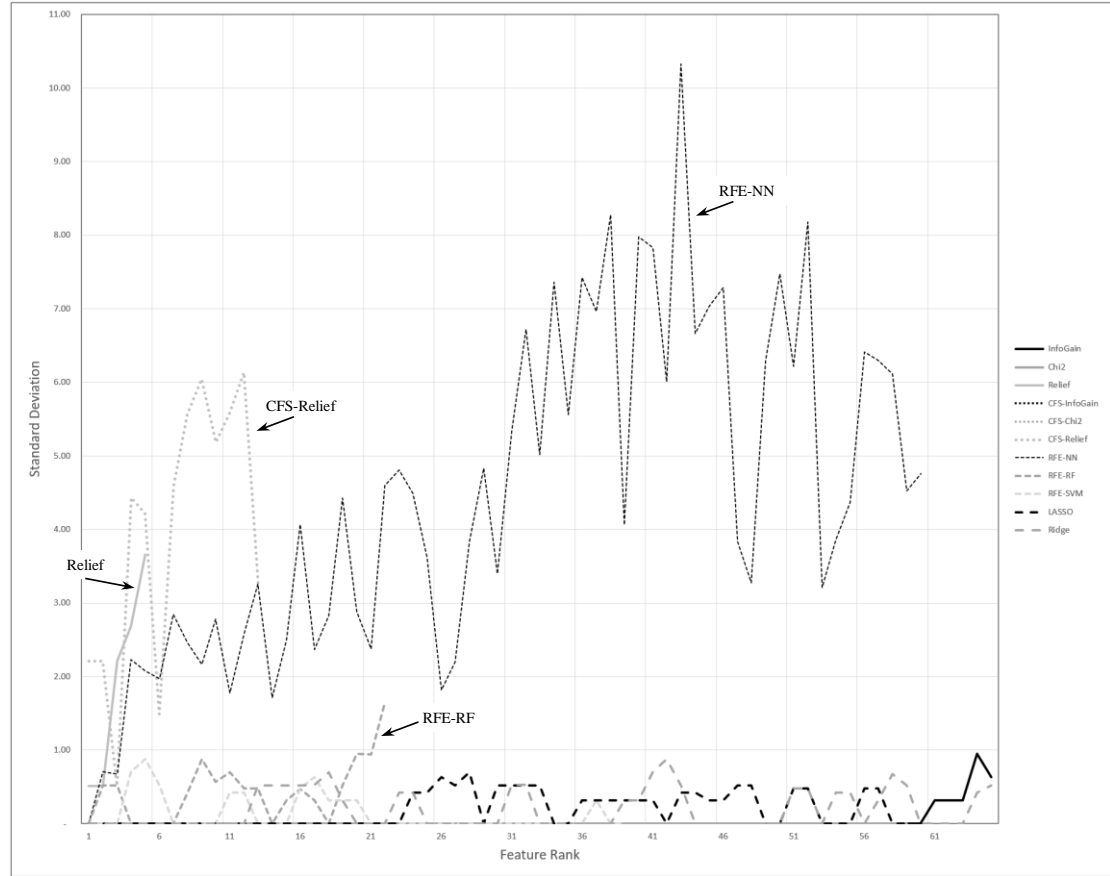


Table 26
Average Repeatability
(Standard Deviation)

FS Method	Std Dev
Chi2	0.00
CFS-InfoGain	0.00
CFS-Chi2	0.00
InfoGain	0.04
RFE-SVM	0.14
LASSO	0.19
Ridge	0.19
RFE-RF	0.44
Relief	1.92
CFS-Relief	3.96
RFE-NN	4.45

for rankings 1 – 60. Only the last four rankings showed any variation of between one to three ranks.

Validating Feature Subsets

After ranking all the features, the next step was to test the efficacy of the various feature subsets across the three machine learning classifiers, Random Forest, Support Vector Machine and Neural Network. In many feature selection algorithms, the user specifies the number of features they want returned, for example, top 10 or top 20. In these experiments, the number of features used with the classifiers was varied so as to determine the effect of feature set size on classifier accuracy. In cases where the number of features to test exceeded the number provided by the performance selection step, then that test sequence ended.

Table 27 presents the test matrix used, a total of 128 feature subsets. As described in [Performance Selection](#), the subset sizes were selected arbitrarily to provide robust coverage of the span of possible selected subsets. Each of these were then

Table 27*Classifier Test Matrix: Feature Subset Size by Feature Selection Method.*

FS Method	Feature count in subset																					
	5	10	15	20	30	40	50	60	75	100	125	150	175	200	225	250	275	300	325	350	375	
InfoGain	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Chi2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Relief	✓																					
CFS-InfoGain	✓	✓																				
CFS-Chi2	✓	✓																				
CFS-Relief	✓	✓	✓																			
RFE-NN	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
RFE-RF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓											
RFE-SVM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓								
LASSO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							
Ridge	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		

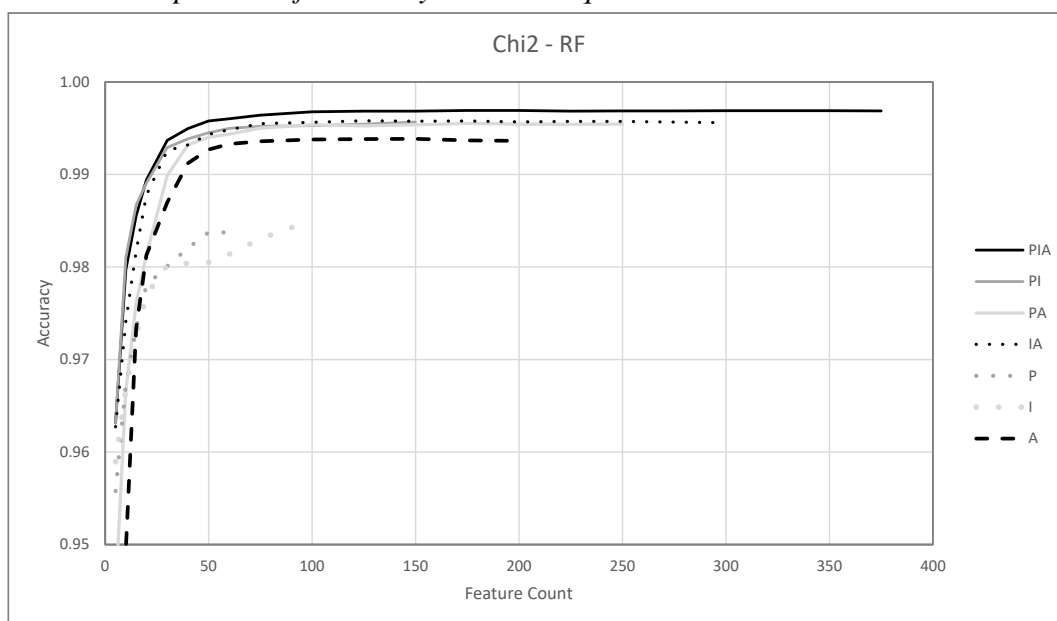
validated using each of the three machine learning methods, creating 384 unique result sets.

Before comparing the various feature selection methods, the data to be analyzed can be reduced by making the final determination on the best dataset to use. Recall from our earlier discussion that seven datasets, as listed in Table 16, have been used throughout the experiments. Each of these were used separately as the dataset for each feature selection method (Table 10) and each machine learning algorithm (Table 11).

Figure 24 shows a comparison in classifier accuracy of the seven datasets using Chi-Square as the feature selection method and Random Forest as the machine learning classifier. This is but one instance of the 33 variations (11 feature selection methods x 3 machine learning algorithms). It is clear from the chart that the dataset PIA (Permissions, Intents and API Calls) performs best, although not significantly. Notice that five of the seven variations achieve over 99% accuracy, with the lowest being API Calls alone. The two between 98% and 99% are Permissions alone and Intents alone.

Figure 24

Dataset Comparison of Accuracy with Chi-Square and Random Forest



The horizontal axis in this chart is feature set count. Recall that it was also a variable in the test matrix. The effects of feature count will be discussed later. The task at hand is to identify the dataset to carry forward into further analysis.

Considering all the data, with respect to the dataset analysis, Chi-Square is representative of the majority of the results. Figure 25 shows RFE – Neural Net with a Random Forest classifier and Figure 26 shows the same with Ridge Regression.

Notice that the dataset performance order is exactly the same in all three charts. There were a couple of instances where PIA was not the top performer, but those were cases with sparse data and no convergence on feature count.

To determine if there is an effect on ordering based on the classifier, Figure 27 shows the dataset comparison of accuracy using Chi-Square as the feature selection methods and Support Vector Machine (SVM) as the classifier and Figure 28 shows the same but with a Neural Network as the classifier.

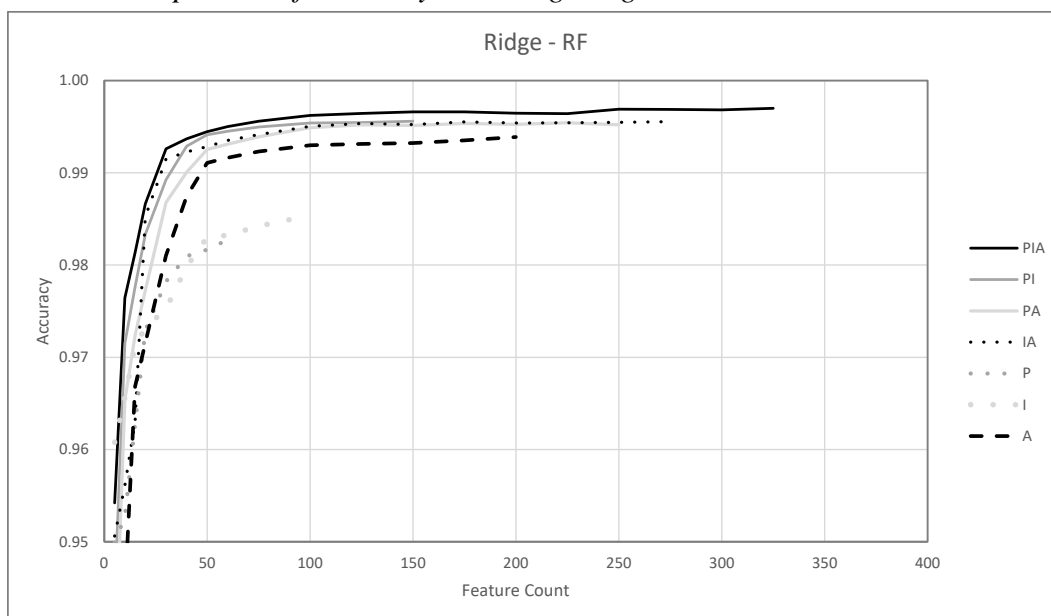
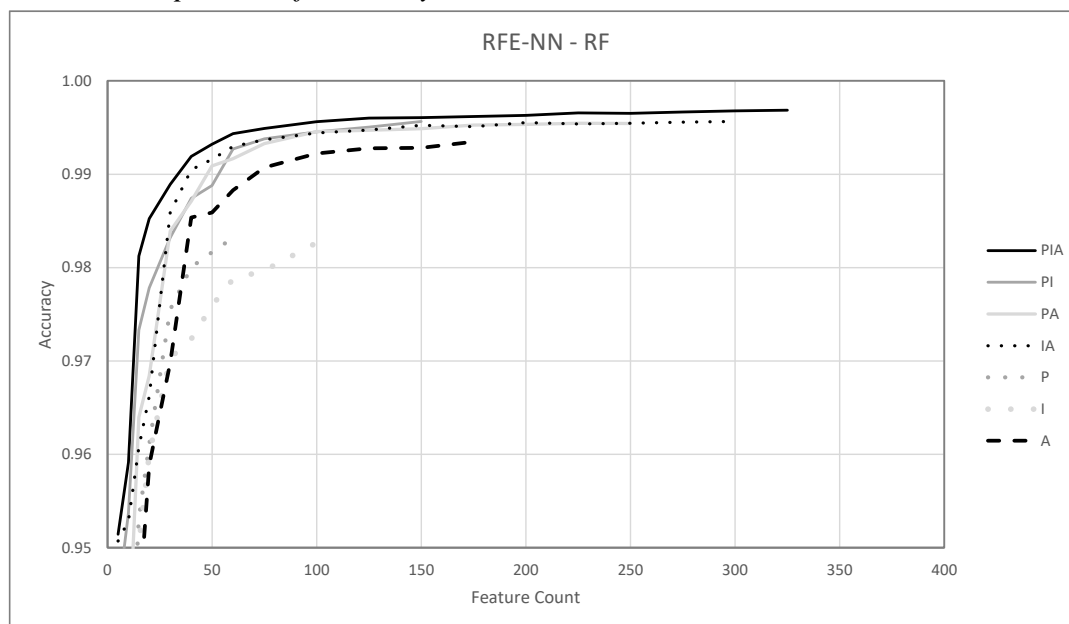
Figure 25*Dataset Comparison of Accuracy with Ridge Regression and Random Forest***Figure 26***Dataset Comparison of Accuracy with RFE-NN and Random Forest*

Figure 27
Dataset Comparison of Accuracy with Chi-Square and SVM

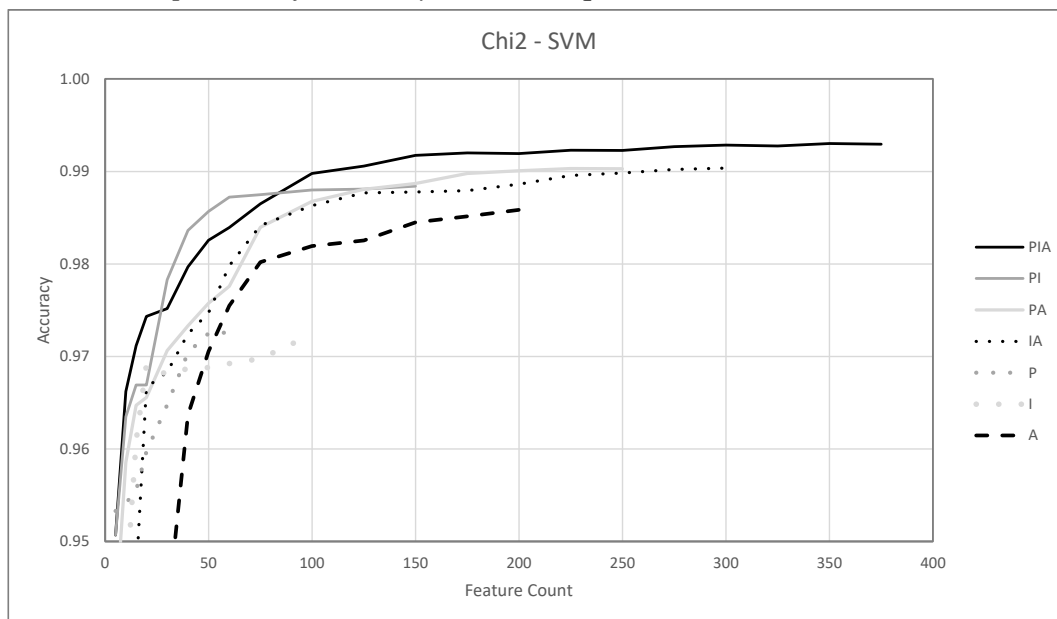
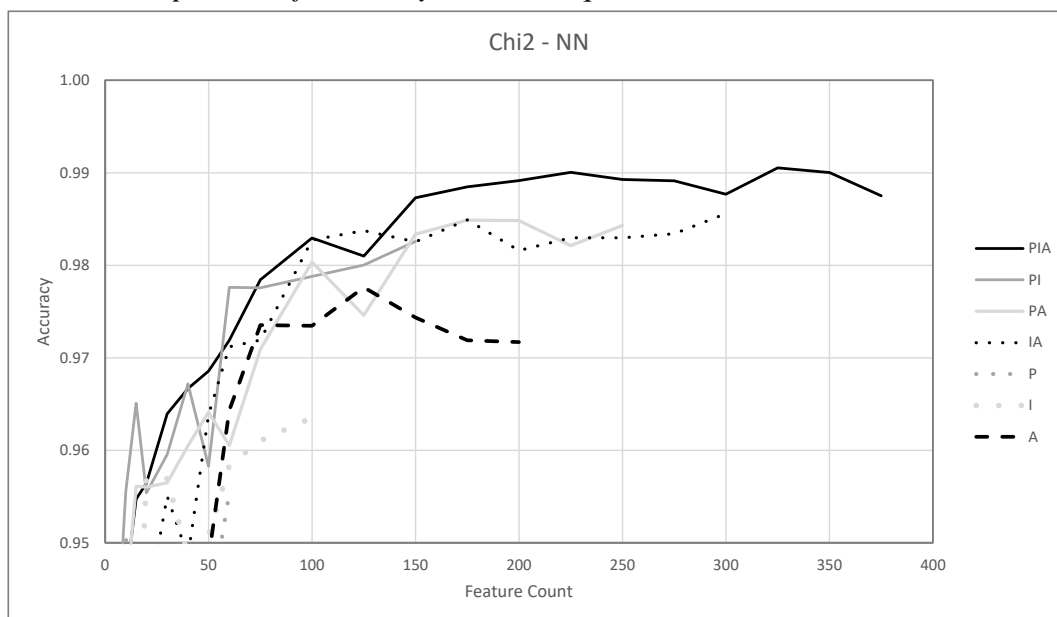


Figure 28
Dataset Comparison of Accuracy with Chi-Square and Neural Network



Note that there is a difference in accuracy among the three classifiers, to be discussed later, but with regard to the selection of the optimal dataset, the consensus for all three classifiers is that PIA performs best.

Charts for all dataset accuracy comparisons for the various feature selection methods and the three machine learning classifiers are presented in Appendix E. Certain variations, such as CFS – Chi-Square with Random Forest are not charted there due to a lack of data points that resulted from the three-phase selection process as shown in Table 27.

The final data stratification to look at is the metric. To this point the metric discussed has been *accuracy*. The actual raw data for the experiments are TP, TN, FN and FP (Table 12) which all go into the calculation of accuracy. All of the other metrics of interest, *true positive rate (TPR)*, *false positive rate (FPR)*, *precision* and *F-measure (FI)*, defined in the Analysis section, use the same raw data so it is no surprise that the trends are the same.

For completeness, these metrics for Chi-Square and Random Forest are presented next. First, Figure 29 shows true positive rate and Figure 30 shows false positive rate.

As one would expect, the two have an inverse relationship. The dataset PIA has the highest TPR and lowest FPR, respectively.

Finally, Figure 31 shows precision in this same context and Figure 32 shows F-measure.

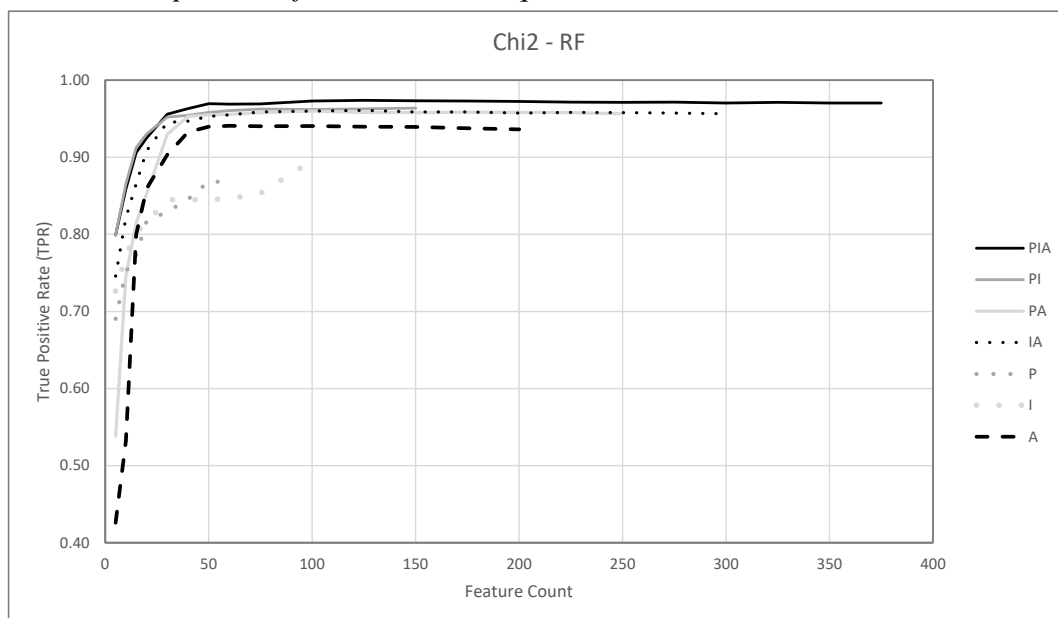
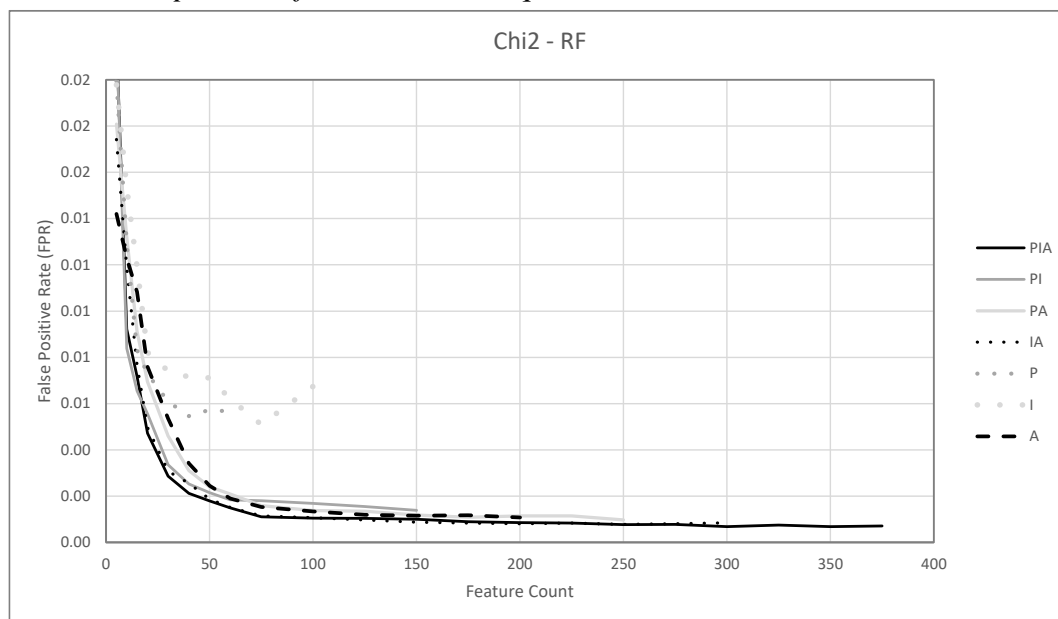
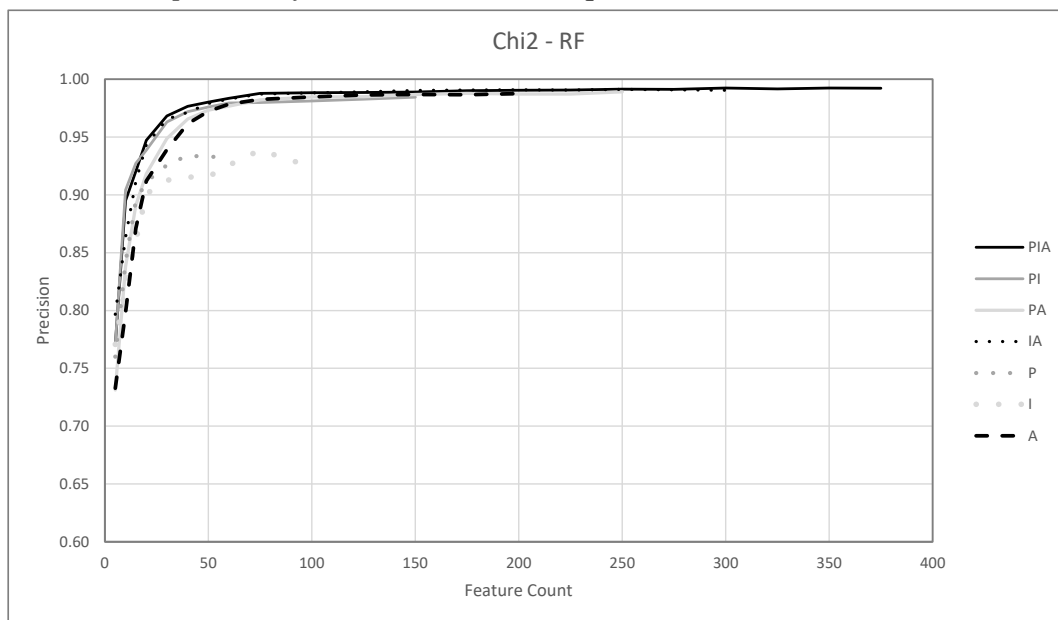
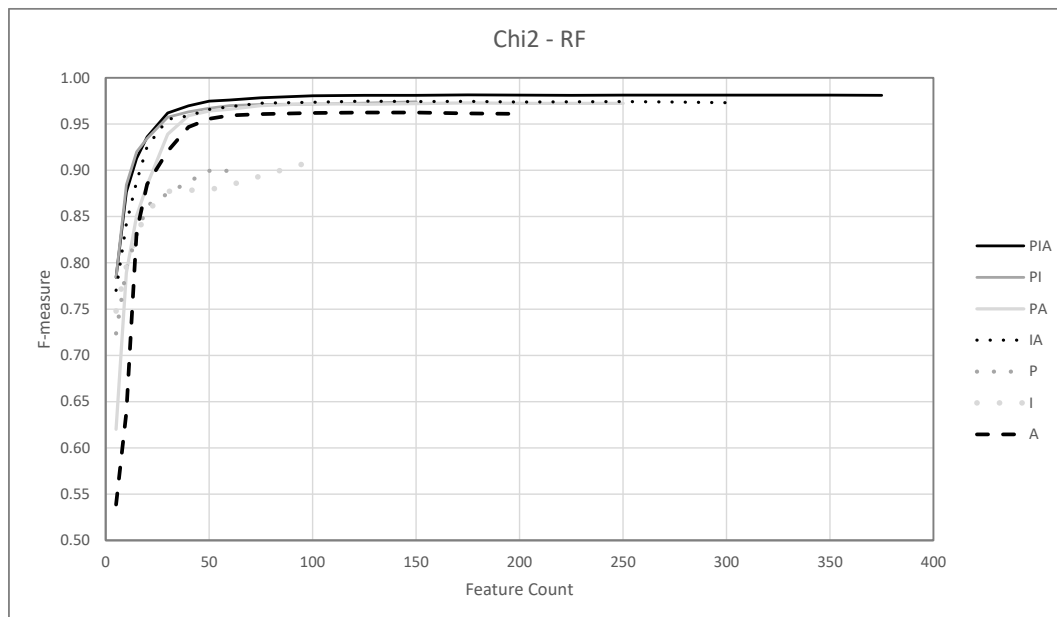
Figure 29*Dataset Comparison of TPR with Chi-Square and Random Forest***Figure 30***Dataset Comparison of FPR with Chi-Square and Random Forest*

Figure 31*Dataset Comparison of Precision with Chi-Square and Random Forest***Figure 32***Dataset Comparison of F-measure with Chi-Square and Random Forest*

With this comparison complete, it has been shown that when evaluating the three machine learning classifiers against the eleven feature selection methods, of the seven dataset combinations, the best performance is achieved using PIA (Permissions, Intents and API Calls) together as opposed to any other available combination and accuracy is a representative metric.

Research Question 3

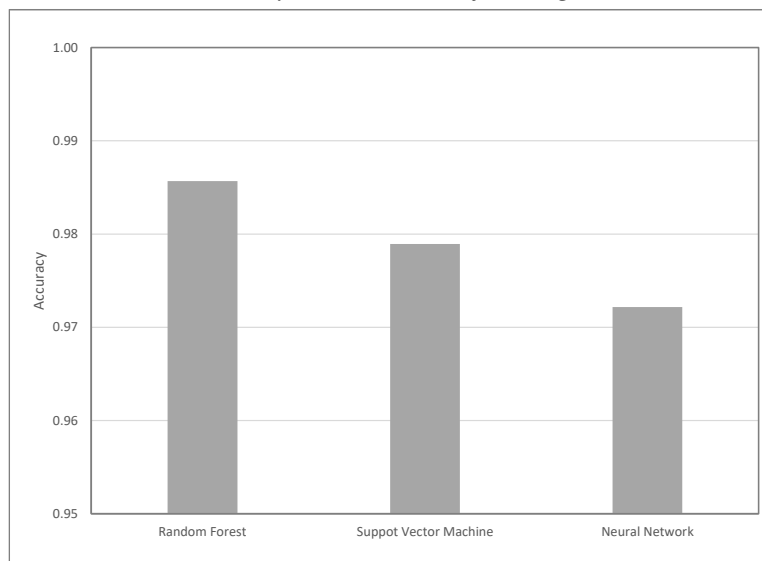
How does machine learning model accuracy vary across machine learning algorithms and feature selection algorithms?

The variation in model accuracy across machine learning algorithms for each feature selection method is presented in Table 28. These data are for the highest accuracy achieved across the attribute count spectrum.

The overall variation in accuracy across machine learning algorithms is shown in Figure 33. The vertical axis represents the average accuracy across all eleven feature selection methods. The figure shows that Random Forest has higher accuracy than SVM,

Table 28
Maximum Accuracy Across Test Matrix

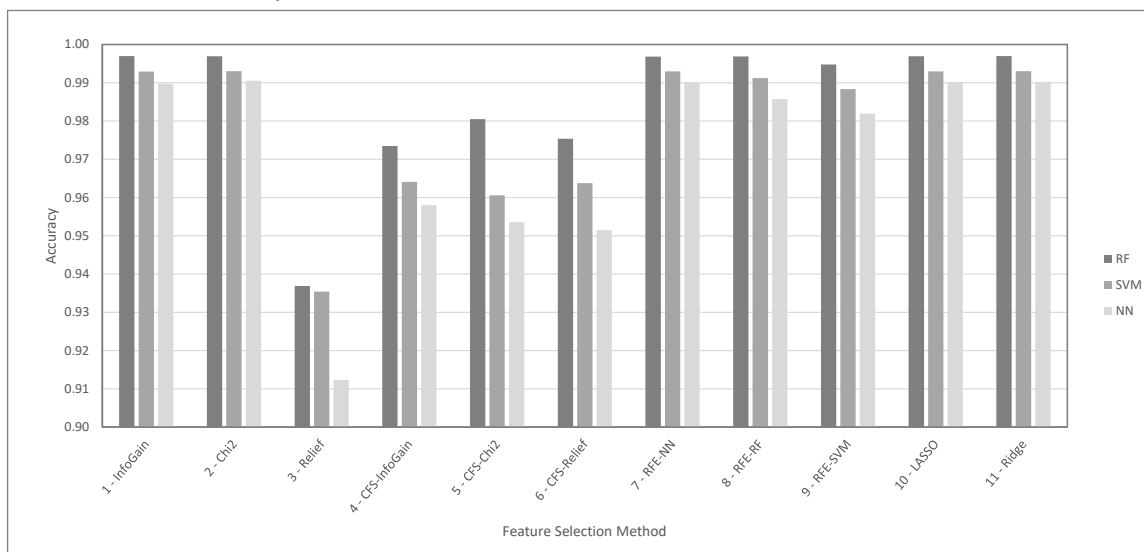
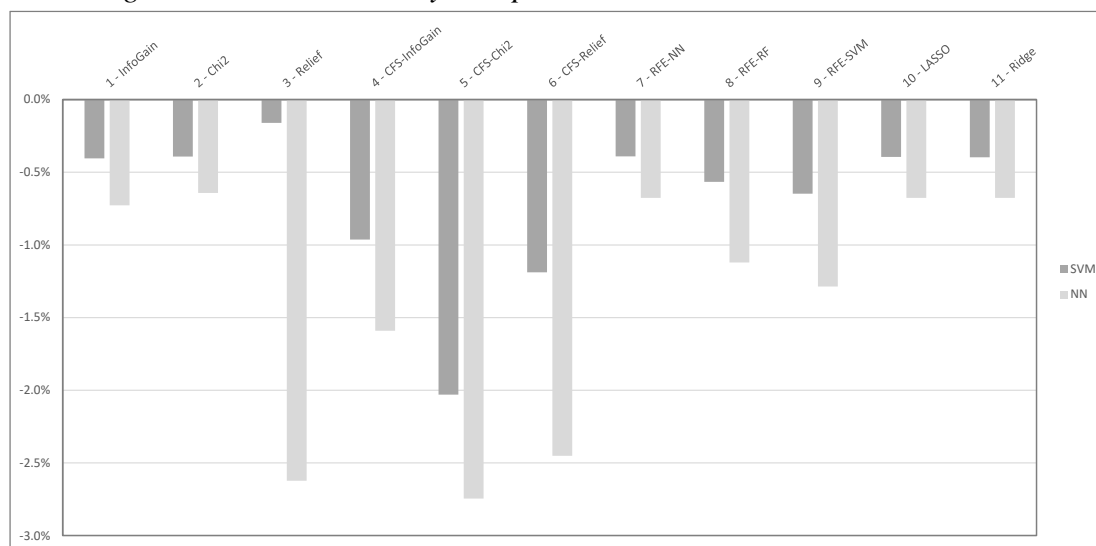
Feature Selection Method	Machine Learning Algorithm		
	Random Forest	Support Vector Machine	Neural Network
1 - InfoGain	0.9969	0.9929	0.9897
2 - Chi2	0.9969	0.9930	0.9875
3 - Relief *	0.9369	0.9354	0.9026
4 - CFS-InfoGain	0.9735	0.9641	0.9580
5 - CFS-Chi2	0.9803	0.9606	0.9526
6 - CFS-Relief *	0.9753	0.9638	0.9514
7 - RFE-NN	0.9969	0.9929	0.9901
8 - RFE-RF	0.9969	0.9912	0.9857
9 - RFE-SVM	0.9941	0.9853	0.9724
10 - LASSO	0.9969	0.9930	0.9902
11 - Ridge	0.9970	0.9930	0.9897

Figure 33*Variation in Accuracy Across Classifier Algorithm*

which in turn has higher accuracy than the Neural Networks. But one must also look at the scale. Compared to Random Forest, SVM is only 0.68% less, and Neural Net is 1.37% less.

Figure 34 shows the variation in accuracy for each classifier as well but broken out by feature selection method. For all eleven methods, the order of highest accuracy is the same, Random Forest, SVM and Neural Net, although divergence in the percent difference can be observed in some. If the null hypothesis were true, i.e., *there is no effect of feature selection algorithm on classifier accuracy*, then one would expect the ordering of highest accuracy to lowest accuracy to vary randomly across the 11 algorithms, but clearly as shown in Figure 34, the ordering is consistent with Random Forest always exhibiting the highest accuracy, followed by SVM and Neural Net. The null hypothesis is false.

To better demonstrate the difference in algorithms by feature selection method, Figure 35 shows the percentage decrease in accuracy for SVM and Neural Net compared

Figure 34*Variation in Accuracy Across Feature Selection Methods***Figure 35***Percentage Decrease in Accuracy Compared to Random Forest*

to Random Forest. Out of the eleven feature selection methods, four are noticeably lower in accuracy, Relief and the three multivariate filter methods based on CFS.

These are the same feature selection methods that stand out in Table 28 depicting the number of attributes selected for feature subsets. This phenomenon will be investigated in more detail next.

Research Question 4

How does feature set size affect model accuracy across feature selection methods?

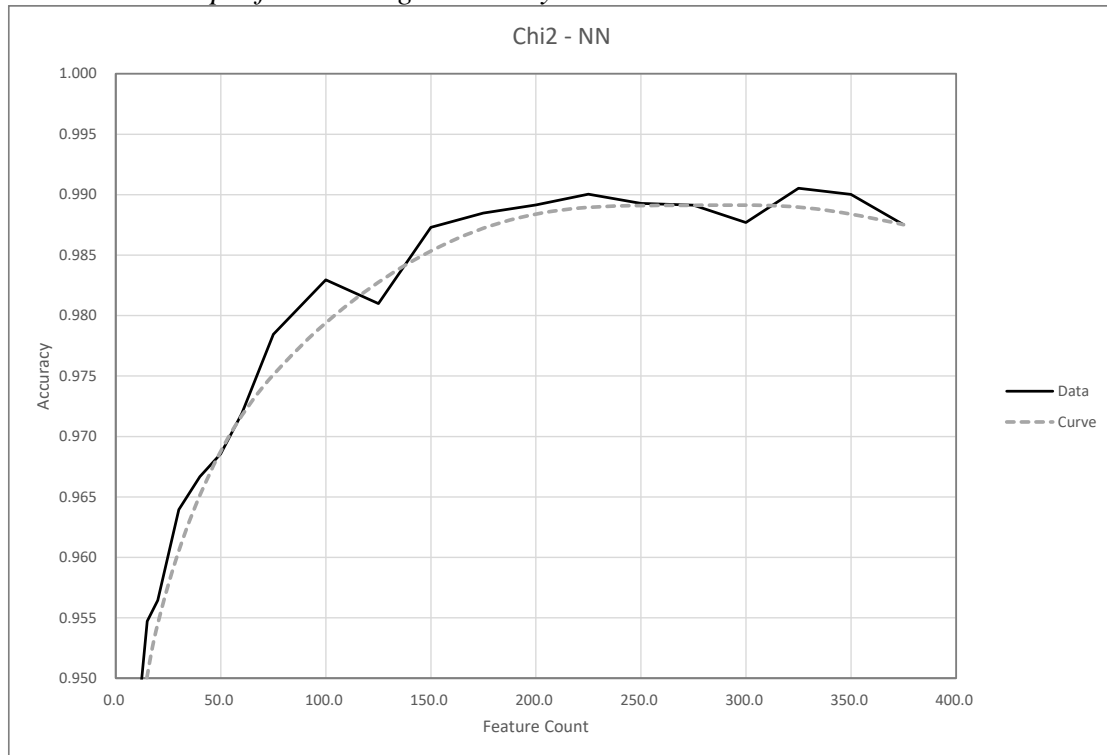
Given that feature set size was a variable in the experiments, there exists cases where a viable solution exists at a smaller set size than the largest tested. As an example, Figure 24 shows a graph of accuracy using Chi-Square as the feature selection method and Random Forest as the validating classifier. While feature set sizes were tested up through a set size of 375, it is clear that the solution converged on an acceptable accuracy level significantly before 375. In this context *convergence* is the point at which further improvements in accuracy are not significant and is defined by:

$$\Delta y \leq \tau \Rightarrow c = True$$

where Δy is the change in accuracy between feature set sizes, τ is the target value (in these experiments arbitrarily set to 0.005) and c is the boolean convergence value.

As shown in earlier data plots, there are some cases where accuracy oscillates and would never converge according to the above formulation. To remedy those situations in order to provide consistency for comparison, a smoothing function was used to produce a curve through the discrete data points. Figure 36 is one example using Chi-Square as the feature selection method and Neural Net as the validating classifier. The solid line

Figure 36
Curve Fit Example for Convergence Analysis



represents the actual data from the experiments and the dashed line represents points along the curve produced by the smoothing function and used for convergence analysis.

Recall there were also test cases where there were minimal valid feature set sizes, as shown in Table 28 with a case in point being Relief having only a test case of feature set size of five. In such instances, *convergence* has no meaning.

In terms of a null hypothesis, i.e., *there is no effect of feature set size on model accuracy*, one intuitively knows this to be false just considering a feature set size of one, to a feature set size of 10 or 100 or more; clearly there would be differences in results. But regardless of intuition, the null hypothesis is shown to be false by the data presented in Figure 36 with accuracy varying significantly as set size increases.

A convergence data summary for test cases using the full dataset (Permissions, Intents and API Calls – PIA) is presented in Figure 37. The dark squares indicate points where the solution was not converged. The lighter squares are with a converged solution and the lighter squares with a star indicate the first data point where the solution converged. Clear squares are where there was no valid feature set.

For cases that did not converge, such as CFS-InfoGain and RF, the accuracy of the last datapoint is shown along with the Δy at that point.

The data shows that classifier accuracy significantly varies based on the feature set size. For example, in the first item, Information Gain and Random Forest, using any feature set size less than 60 will result in a suboptimal accuracy performance. Also note that the optimal feature set size varies with the classifier. This is true in all cases with the exception of the embedded methods (LASSO Regression and Ridge Regression) where convergence occurs at approximately the same point.

Feature Summary

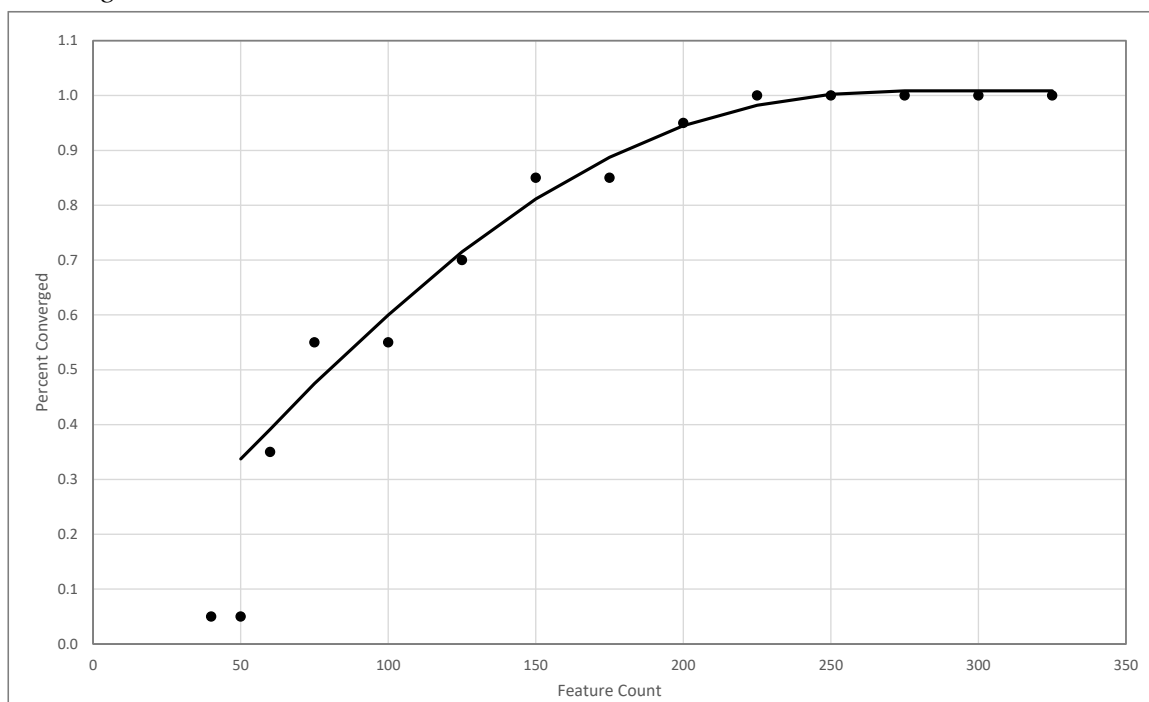
Research Question 5

Among Permissions, Intents and API Calls, what are the important features?

It has been shown above that 1) using Permissions, Intents and API Calls together are better than any of the three in other combinations, and 2) that optimal feature set size can vary significantly with feature selection method and is an important parameter to consider in order to build a model that will provide optimum accuracy.

Figure 38 plots the 20 convergence points from Figure 37 with a curve fit added to depict a continuous function. Clearly there is a trend, and what it indicates is that if one wants to ensure that their combination of feature selection method and classifier is optimized in terms of convergence, then using the top 200 or above Permissions, Intents and API Calls is the correct approach. For convenience, these top 200 attributes are listed in Appendix F.

Figure 38
Convergence Versus Feature Set Size



Chapter 5

Conclusions, Implications and Recommendations

Conclusions

The following provides the major conclusions of this study. The methods used are generic to the field, but these conclusions are specific to the domain of feature selection for malware detection on the Android platform.

Permissions vs Intents vs API Calls

Considering the use of Android Permissions, Intents and API Calls as features, using all three provides the best results. Additionally, using a combination of any two of the feature types provides better results than using any one of them alone.

Feature Selection Using Multiple Feature Types

When using a combination of the three feature types, performing feature selection on the combination provides better results than performing feature selection on each type and then combining the results to create a final feature set.

Feature Set Size

Feature set size is important. While one specific test case did produce optimal results with only the top 40 features, in general best results are achieved with a 200 or greater feature count. If a feature set is used that is larger than the minimum required, accuracy does not decrease, so it is safe to use more features than absolutely required when in doubt.

Feature Selection Methods to be Avoided

Relief proved to be a poor performer of feature selection. In general, the features it selects do not outperform randomly selected features. This could be attributable to the random row selection function in the algorithm. When used with a large sample size dataset and following conventional parameter settings of 10 iterations, there may be too much randomness to allow consistent selection of appropriate features. A study of varying dataset size and iteration count could shed light on the problem.

Given the performance of Relief, it is no surprise that CFS-Relief performed poorly as well. However, regardless of the base correlation method used, CFS should also be avoided as a feature selection method. While the features it selects are significant, the algorithm stops too soon thus providing an insufficient feature set size. The algorithm is deterministic in nature and thus repeatable based on the same correlation input, so overcoming its shortcomings will require revisiting the design.

RFE – Neural Net (Perceptron) as a feature selection method is unpredictable. While the accuracy results were acceptable in these experiments, the repeatability was poor thus implying that not all feature sets selected would be acceptable.

Acceptable Feature Selection Methods

Information Gain, Chi-Square, Ridge Regression and LASSO Regression are excellent feature selection methods with minimal variance across machine learning classifiers. Additionally, all four methods take little computational time.

The main characteristic these methods have in common are that they all provide weight factors for a large percentage of the feature input vectors. In fact, the first three provide weights for all features while LASSO does allow some feature weights to resolve

to zero. This implies that researchers seeking the highest fidelity should not rely on feature selection algorithms to specify the feature count, but rather should use some form of performance selection and convergence analysis as shown in this report.

The two wrapper methods RFE–Random Forest and RFE–Support Vector Machine provide acceptable results, but those results vary more with the machine learning classifier used compared to those listed in the previous paragraph. However, RFE-Support Vector Machine was significantly slower in terms of compute time compare to all other acceptable feature selection methods.

Accuracy

While it was not a goal of this research to find a best performing feature set or a best performing machine learning model, even without fine tuning of parameters, the six acceptable feature selection methods averaged 99.6% accuracy with a Random Forest classifier, 99.1% accuracy with a Support Vector Machine classifier and 98.6% accuracy with a Neural Net (Perceptron) classifier.

Implications

This work is the most exhaustive analysis of Android feature selection in this field of study to the best of our knowledge. It can be used as a guide for researchers performing feature selection in the domain or a reference for researchers who want to skip feature selection as a process and simply use a feature set based on the features listed in the document.

Previous studies in the field that made use of the five feature selection methods described as *to be avoided* or used feature set sizes too small, as defined herein, should have the results revisited and possibly revised using appropriate feature sets.

Many of the insights and conclusions presented here have the potential to be applied to other malware/anomaly detection tasks, or more broadly, other pattern recognitions problems, especially with regard to researchers needing to be cautious of blindly relying on feature selection algorithms.

As shown, Relief, CFS, RFE-NN in general are not suitable for feature selection in domains with large feature sets given their propensity to provide an insufficient feature count. On the other hand, Information Gain, Chi-Square, LASSO Regression and Ridge Regression are excellent feature selection choices assuming the full weight vectors are used appropriately with additional analysis as demonstrated.

Although evaluating performance of machine learning classifiers was not in the scope of this research, it was observed that Random Forest performed exceptionally well, which is consistent with other published work in the domain as discussed. Of the three techniques used, Random Forest was the only ensemble method, leading to the conjecture that other ensemble approaches might perform just as well or better.

Recommendations

Future Work

This work encompassed a large scope, but as with any research project, there are additional pathways that could be explored. One such path would be to expand the experiments by adding feature selection algorithms, such as using Pearson's Correlation or RFE with other embedded classifiers. One could also vary the parameter settings on the wrapper and embedded methods to see how such variations would affect selection. Likewise, feature set validation with additional classifiers would be interesting as well as varying parameter settings on the classifiers used in validation.

Another path would be to expand on the analysis such as varying the ensemble voting scheme using quartile membership or even using other similarity algorithms in addition to Euclidian distance. These would not change the underlying experimental data but might provide additional insights and interpretations.

A more extreme path would be to explore improving the Relief and CFS algorithms to determine if they can be improved for use in this domain.

Finally, even with the large size of the dataset, one could update it with more recent benign and malware samples.

Best Practices

It is clear from the results reported herein that researchers in the Android malware detection field need to pay significant attention to feature selection. One can use this work to explore and expand feature selection or simply pick from its recommendations. But using arbitrary methods such as using only Permissions categorized as *Dangerous* by Google (Permissions Overview, 2019) or having a feature selection method return its top 10 or top 20 features is not best practice.

Summary

Introduction

Android has been Google's mobile operating system from 2008 to the present. It currently holds approximately 85% of the world market with the only other relevant competitor being Apple's iOS. Android is open source with an open ecosystem which tends to make it an easy target for malware perpetrators.

Detection of malicious activity on computers has a significant research stream going as far back as 1980. This research is focused on anomaly detection which started

with Expert Systems and eventually evolved into using machine learning and data mining techniques. Malware detection on mobile devices typically takes a host-based approach, evaluating the apps on the system, as opposed to network-based which would be less effective given the on-again, off-again nature of mobile network connectivity.

When developing machine learning models, one initial, critical step is feature selection, the process of identifying a subset of relevant features for use in construction of the model. For Android, the features most often used are Permissions, Intents and API Calls, individually, together or in some combination. But there are a number of other features used less frequently such as hardware used, Android App Components, Control Flow Graphs, URLs and many others. The problem is there is no consensus in the research community as to the key Android feature types for machine learning models. Even just considering the top three of Permissions, Intents and API Call, those are just categories. There are hundreds of discrete attributes in each category.

Typical feature selection approaches in the community include using the subset of Permissions categorized as *Dangerous* by Google, hand selecting features based on domain knowledge, or even picking a method such as Information Gain and simply taking the top n -ranked features. Unfortunately, there is no definitive study on Android feature selection for researchers to be guided by or use as a reference.

Such is the goal of this research, to advance the state of the industry's knowledge on feature sets used for Android static analysis malware detection. The approach was to use a broad test matrix consisting of all combinations of Permissions, Intents and API Calls, each evaluated by several different feature selection algorithms and then each of

those validated with multiple machine learning classifiers. Then, using the experimental results, answer the following five questions:

- 1) How does feature ranking vary when Permissions, Intents and API Calls are selected separately versus combined?
- 2) How does feature ranking vary across feature selection algorithms?
- 3) How does machine learning model accuracy vary across machine learning algorithms and feature selection algorithms?
- 4) How does feature set size affect model accuracy across feature selection methods?
- 5) Among Permissions, Intents and API Calls, what are the important features?

Methodology

Based on common usage in the community as well as to use several types of feature selection algorithms, eleven different feature selection methods were chosen. The selected methods were:

Chi-Square

Information gain

Relief

Correlation-based Feature Selection (CFS) with Chi-Square

Correlation-based Feature Selection (CFS) with Information Gain

Correlation-based Feature Selection (CFS) with Relief

Recursive Feature Elimination (RFE) using Neural Network (Perceptron)

Recursive Feature Elimination (RFE) using Random Forest

Recursive Feature Elimination (RFE) using Support Vector Machine

Lasso regression (L1 regularization)

Ridge regression (L2 regularization)

which encompasses multiple examples of univariate methods, multivariate methods, wrapper methods and embedded methods.

To validate the results from the various feature selection methods, it was desired to use multiple machine learning classifiers so as to demonstrate independence. The algorithms selected were:

Random Forest

Support Vector Machine

Neural Network (Perceptron)

which also matched the classifiers used as the search engines in the RFE experiments.

The experiments used a large dataset of 119k Android applications, the Andro-AutoPsy set from the University of Korea Hacking and Countermeasure Research lab. Each app was decompressed and decompiled using an open-source tool called Jadx. Instances of Permissions, Intents and API Calls were extracted from all of the apps and transformed into a binary encoded dataset indicating the existence or lack of existence of each attribute in each app. In addition to the extracted features, each dataset contained three columns of random data as contrast variables against which we could compare feature selection results.

Each of the seven combinations of feature types (Permissions, Intents and API Calls) were used as input into the 11 different feature selection methods. The weight values returned by each algorithm was used to infer a ranking, most important to least important. Also, each attribute weight was compared to the weight the feature selection algorithm gave to the random data, and any attribute whose weight was equal to or less than the average weight of those random contrast variables was eliminated.

Finally, k -based feature subsets were created based on the rankings of the features still present in each subset, i.e., top 5, top 10, top 20, etc. These multiple sized subsets

from each feature selection method for each dataset combination, were then used as input for the three machine learning classifiers.

Results

For the feature vectors created from the experiments, Euclidean distance was employed to evaluate similarity. In terms of dataset combinations, it was shown that there was significant similarity in the first 10 to 15 percent of each vector when comparing the features selected from the combined datasets (Permissions, Intents and API Calls together) versus selecting from the feature types individually. But after 15 percent there is significant divergence.

Euclidean distance was also used to compare the similarity of the feature vectors representing the various feature selection methods. This demonstrated that Information Gain and Chi-Square produced very similar feature sets followed closely by the two embedded methods. Overall, ten of the feature selection methods exhibited reasonable similarity in the first 10 to 15 percent of the vectors with the one exception being RFE-SVM.

After using all the feature subsets with the three different machine learning classifiers, it was shown that using a combination of Permissions, Intents and API Calls produced higher accuracy than using any of those alone or in any other combination. It was also demonstrated that when using multiple feature types, feature selection should be performed on the types combined, not separately and then combined.

The k -based selection experiments showed that feature set size is important, and in general, researchers should be using around 200 features or more for optimal classification.

With regard to efficacy of the various feature selection methods, the data indicated that Relief, the three CFS based methods and RFE-Neural Network were not satisfactory and therefore should be avoided. Conversely the data showed that Information Gain, Chi-Square, LASSO Regression and Ridge Regression are very good feature selection methods, and RFE–Random Forest and RFE–Support Vector Machine are moderately good with the latter being least so due to the computational time it requires.

Data and Code Repositories

All the data and custom Python code used in these experiments is available to the public in the author's Github repositories:

https://github.com/fcguyton/android_feature_selection

https://github.com/fcguyton/ml_algorithms, and

https://github.com/fcguyton/python_utilities.

Appendix A

References for Table 4

First Author	Year	Reference
Aafer, Yousra	2013	(Aafer, Du, & Yin, 2013)
Abaid, Zainab	2017	(Abaid, Kaafar, & Jha, 2017)
Abro, Fauzia	2018	(Abro, 2018)
Adebayo, Olawale	2014	(Adebayo & AbdulAziz, 2014)
Alatwi, Huda	2016	(Alatwi, 2016)
Allix, Kevin	2014	(Allix K. , Bissyande, Jerome, Klein, & Le, 2016)
Allix, Kevin	2016	(Allix K. , Bissyande, Klein, & Le, 2014)
Alswaina, Fahad	2018	(Alswaina & Elleithy, 2018)
Altaher, Altyeb	2017	(Altaher, 2017)
Altaher, Altyeb	2017	(Altaher & BaRukab, 2017)
Ariyapala, Kanishka	2016	(Ariyapala, Do, Anh, Ng, & Conti, 2016)
Arp, Daniel	2014	(Arp, et al., 2014)
Aswini, AM	2014	(Aswini & Vinod, 2014)
Aung, Zarni	2013	(Aung & Zaw, 2013)
Chan, Patrick	2014	(Chan & Song, 2014)
Coronado-De-Alba, Lilian	2016	(Coronado-De-Alba, Rodriguez-Mota, & Escamilla-Ambrosio, 2016)
Duc, Nguyen	2018	(Duc & Giang, 2018)
Fan, Ming	2017	(Fan, et al., 2017)
Feizollah, Ali	2017	(Feizollah, Anuar, Salleh, Suarez-Tangil, & Furnell, 2017)
Firdaus, Ahmad	2018	(Firdaus, Anuar, Karim, & Ab, 2018)
Ghaffari, Fariba	2017	(Ghaffari, Abadi, & Tajoddin, 2017)
Ghorbanzadeh, Mo	2013	(Ghorbanzadeh, Chen, Ma, Clancy, & McGwier, 2013)
Glodek, William	2013	(Glodek & Harang, 2013)
Huang, Chun-Ying	2013	(Huang, Tsai, & Hsu, 2013)
Idrees, Fauzia	2014	(Idrees & Rajarajan, 2014)
Idrees, Fauzia	2017	(Idrees, Rajarajan, Conti, Chen, & Rahulamathavan, 2017)
Li, Jin	2018	(Li, et al., 2018)
Li, Wenjia	2015	(Li, Ge, & Dai, 2015)
Liu, Che-Hsun	2016	(Liu, Zhang, & Wang, 2016)

Liu, Ning	2017	(Liu, Yang, & Zhang, 2017)
Liu, Ning	2018	(Liu N. , et al., 2018)
Lu, Yung-Feng	2018	(Lu, Kuo, Chen, Chen, & Chou, 2018)
Mahindru, Arvind	2017	(Mahindru & Singh, 2017)
Mahmood, Riyadh	2014	(Mahmood, Mirzaei, & Malek, 2014)
Mariconti, Enrico	2016	(Mariconti, et al., 2016)
Martin, Alejandro	2017	(Martin, Fuentes-Hurtado, Naranjo, & Camacho, 2017)
Melis, Marco	2018	(Melis, Maiorca, Biggio, Giacinto, & Roli, 2018)
Moonsamy, Veelasha	2014	(Moonsamy, Rong, & Liu, 2014)
Morales-Ortega, Salvador	2016	(Morales-Ortega, Escamilla-Ambrosio, Rodriguez-Mota, & Coronado-De-Alba, 2016)
Nauman, Mohammad	2018	(Nauman, Tanveer, Khan, & Syed, 2018)
Naway, Abdelmonim	2018	(Naway & Li, 2018)
Nezhadkamali, Maryam	2017	(Nezhadkamali, Soltani, & Seno, 2017)
Nix, Robin	2017	(Nix & Zhang, 2017)
Papadopoulos, Harris	2018	(Papadopoulos, Georgiou, Eliades, & Konstantinidis, 2018)
Peiravian, Naser	2013	(Peiravian & Zhu, 2013)
Qiao, Mengyu	2016	(Qiao, Sung, & Liu, 2016)
Rashidi, Bahman	2017	(Rashidi, Fung, & Bertino, 2017)
Reyhani, Hamedani	2018	(Reyhani, Shin, Lee, Cho, & Hwang, 2018)
Rovelli, Paolo	2014	(Rovelli & Vigfusson, 2014)
Sahs, Justin	2012	(Sahs & Khan, 2012)
Shahriar, Hossain	2017	(Shahriar, Islam, & Clincy, 2017)
Shang, Fengjun	2017	(Shang, Li, Deng, & He, 2018)
Sharma, Akanksha	2014	(Sharma & Dash, 2014)
Shelke, Chetan	2017	(Shelke, 2017)
Smutz, Charles	2016	(Smutz & Stavrou, 2016)
Sun, Lichao	2016	(Sun, Li, Yan, Srisa-an, & Pan, 2016)
Verma, Sushma	2016	(Verma & Muttoo, 2016)
Wang, Wei	2018	(Wang, Li, Wang, Liu, & Zhang, 2018)
Wang, Wei	2018	(Wang, Zhao, & Wang, 2018)
Wang, Xiaoqing	2016	(Wang, Wang, & Zhu, 2016)
Wu, Dong-Jie	2012	(Wu, Mao, Wei, Lee, & Wu, 2012)
Xu, Ke	2018	(Xu, Li, Deng, & Chen, 2018)
Yang, Ming	2017	(Yang, Wang, Ling, Liu, & Ni, 2017)

Yerima, Suleiman	2013	(Yerima, Sezer, McWilliams, & Muttik, 2013)
Yerima, Suleiman	2015	(Yerima, Sezer, & Muttik, 2015)
Zhang, Yi	2018	(Zhang, Yang, & Wang, 2018)
Zhao, Min	2011	(Zhao, Ge, Zhang, & Yuan, 2011)
Zhao, Min	2012	(Zhao, Zhang, Ge, & Yuan, 2012)
Zhao, Xiaoyan	2014	(Zhao, Fang, & Wang, 2014)
Zhu, Hui-Juan w/ Jiang	2018	(Zhu, et al., 2018) w/ Jiang
Zhu, Hui-Juan w/ You	2018	(Zhu, et al., 2018) w/ You

Appendix B

Experiment Feature List

Permissions	
ID	Permission
P004	ACCESS_COARSE_LOCATION
P005	ACCESS_FINE_LOCATION
P006	ACCESS_LOCATION_EXTRA_COMMANDS
P008	ACCESS_NETWORK_STATE
P010	ACCESS_WIFI_STATE
P012	ACTIVITY_RECOGNITION
P013	ADD_VOICEMAIL
P015	BATTERY_STATS
P016	BIND_ACCESSIBILITY_SERVICE
P025	BIND_DEVICE_ADMIN
P028	BIND_INPUT_METHOD
P031	BIND_NOTIFICATION_LISTENER_SERVICE
P034	BIND_REMOTEVIEWS
P037	BIND_TEXT_SERVICE
P041	BIND_VPN_SERVICE
P043	BIND_WALLPAPER
P044	BLUETOOTH
P045	BLUETOOTH_ADMIN
P050	BROADCAST_STICKY
P053	CALL_PHONE
P055	CAMERA
P058	CHANGE_CONFIGURATION
P059	CHANGE_NETWORK_STATE
P060	CHANGE_WIFI_MULTICAST_STATE
P061	CHANGE_WIFI_STATE
P062	CLEAR_APP_CACHE
P064	DELETE_CACHE_FILES
P067	DISABLE_KEYGUARD
P069	EXPAND_STATUS_BAR
P072	GET_ACCOUNTS
P074	GET_PACKAGE_SIZE
P076	GLOBAL_SEARCH
P079	INSTALL_SHORTCUT
P081	INTERNET
P082	KILL_BACKGROUND_PROCESSES
P088	MODIFY_AUDIO_SETTINGS
P092	NFC
P094	PACKAGE_USAGE_STATS
P096	PROCESS_OUTGOING_CALLS
P097	READ_CALENDAR
P098	READ_CALL_LOG
P099	READ_CONTACTS
P100	READ_EXTERNAL_STORAGE
P104	READ_PHONE_STATE
P105	READ_SMS
P106	READ_SYNC_SETTINGS
P107	READ_SYNC_STATS
P110	RECEIVE_BOOT_COMPLETED
P111	RECEIVE_MMS
P112	RECEIVE_SMS
P113	RECEIVE_WAP_PUSH
P114	RECORD_AUDIO
P115	REORDER_TASKS
P125	SET_ALARM
P133	SET_WALLPAPER
P134	SET_WALLPAPER_HINTS
P139	SYSTEM_ALERT_WINDOW

P146	USE_SIP
P147	VIBRATE
P148	WAKE_LOCK
P150	WRITE_CALENDAR
P151	WRITE_CALL_LOG
P152	WRITE_CONTACTS
P153	WRITE_EXTERNAL_STORAGE
P156	WRITE_SETTINGS
P157	WRITE_SYNC_SETTINGS
Intents	
ID	Intent
I001	ACTION_AIRPLANE_MODE_CHANGED
I002	ACTION_ALL_APPS
I003	ACTION_ANSWER
I006	ACTION_APP_ERROR
I007	ACTION_ASSIST
I008	ACTION_ATTACH_DATA
I009	ACTION_BATTERY_CHANGED
I010	ACTION_BATTERY_LOW
I011	ACTION_BATTERY_OKAY
I012	ACTION_BOOT_COMPLETED
I013	ACTION_BUG_REPORT
I014	ACTION_CALL
I015	ACTION_CALL_BUTTON
I016	ACTION_CAMERA_BUTTON
I018	ACTION_CHOOSER
I019	ACTION_CLOSE_SYSTEM_DIALOGS
I020	ACTION_CONFIGURATION_CHANGED
I022	ACTION_CREATE_SHORTCUT
I023	ACTION_DATE_CHANGED
I024	ACTION_DEFAULT
I026	ACTION_DELETE
I027	ACTION_DEVICE_STORAGE_LOW
I028	ACTION_DEVICE_STORAGE_OK
I029	ACTION_DIAL
I030	ACTION_DOCK_EVENT
I031	ACTION_DREAMING_STARTED
I032	ACTION_DREAMING_STOPPED
I033	ACTION_EDIT
I034	ACTION_EXTERNAL_APPLICATIONS_AVAILABLE
I035	ACTION_EXTERNAL_APPLICATIONS_UNAVAILABLE
I036	ACTION_FACTORY_TEST
I037	ACTION_GET_CONTENT
I041	ACTION_HEADSET_PLUG
I042	ACTION_INPUT_METHOD_CHANGED
I043	ACTION_INSERT
I044	ACTION_INSERT_OR_EDIT
I046	ACTION_INSTALL_PACKAGE
I047	ACTION_LOCALE_CHANGED
I049	ACTION_MAIN
I055	ACTION_MANAGE_NETWORK_USAGE
I056	ACTION_MANAGE_PACKAGE_STORAGE
I057	ACTION_MEDIA_BAD_REMOVAL
I058	ACTION_MEDIA_BUTTON
I059	ACTION_MEDIA_CHECKING
I060	ACTION_MEDIA_EJECT
I061	ACTION_MEDIA_MOUNTED
I062	ACTION_MEDIA_NOFS
I063	ACTION_MEDIA_REMOVED
I064	ACTION_MEDIA_SCANNER_FINISHED
I065	ACTION_MEDIA_SCANNER_SCAN_FILE
I066	ACTION_MEDIA_SCANNER_STARTED
I067	ACTION_MEDIA_SHARED
I068	ACTION_MEDIA_UNMOUNTABLE
I069	ACTION_MEDIA_UNMOUNTED
I070	ACTION_MY_PACKAGE_REPLACED

I073	ACTION_NEW_OUTGOING_CALL
I074	ACTION_OPEN_DOCUMENT
I078	ACTION_PACKAGE_ADDED
I079	ACTION_PACKAGE_CHANGED
I080	ACTION_PACKAGE_DATA_CLEARED
I081	ACTION_PACKAGE_FIRST_LAUNCH
I082	ACTION_PACKAGE_FULLY_REMOVED
I083	ACTION_PACKAGE_INSTALL
I084	ACTION_PACKAGE_NEEDS_VERIFICATION
I085	ACTION_PACKAGE_REMOVED
I086	ACTION_PACKAGE_REPLACED
I087	ACTION_PACKAGE_RESTARTED
I089	ACTION_PASTE
I090	ACTION_PICK
I091	ACTION_PICK_ACTIVITY
I092	ACTION_POWER_CONNECTED
I093	ACTION_POWER_DISCONNECTED
I094	ACTION_POWER_USAGE_SUMMARY
I096	ACTION_PROVIDER_CHANGED
I097	ACTION_QUICK_CLOCK
I099	ACTION_REBOOT
I100	ACTION_RUN
I101	ACTION_SCREEN_OFF
I102	ACTION_SCREEN_ON
I103	ACTION_SEARCH
I104	ACTION_SEARCH_LONG_PRESS
I105	ACTION_SEND
I106	ACTION_SENDTO
I107	ACTION_SEND_MULTIPLE
I108	ACTION_SET_WALLPAPER
I110	ACTION_SHUTDOWN
I111	ACTION_SYNC
I112	ACTION_SYSTEM_TUTORIAL
I113	ACTION_TIMEZONE_CHANGED
I114	ACTION_TIME_CHANGED
I115	ACTION_TIME_TICK
I116	ACTION_TRANSLATE
I117	ACTION_UID_REMOVED
I118	ACTION_UMS_CONNECTED
I119	ACTION_UMS_DISCONNECTED
I120	ACTION_UNINSTALL_PACKAGE
I121	ACTION_USER_BACKGROUND
I122	ACTION_USER_FOREGROUND
I123	ACTION_USER_INITIALIZE
I124	ACTION_USER_PRESENT
I126	ACTION_VIEW
I129	ACTION_VOICE_COMMAND
I130	ACTION_WALLPAPER_CHANGED
I131	ACTION_WEB_SEARCH
I132	CATEGORY_ALTERNATIVE
I133	CATEGORY_APP_BROWSER
I134	CATEGORY_APP_CALCULATOR
I135	CATEGORY_APP_CALENDAR
I136	CATEGORY_APP_CONTACTS
I137	CATEGORY_APP_EMAIL
I139	CATEGORY_APP_GALLERY
I140	CATEGORY_APP_MAPS
I141	CATEGORY_APP_MARKET
I142	CATEGORY_APP_MESSAGING
I143	CATEGORY_APP_MUSIC
I144	CATEGORY_BROWSABLE
I145	CATEGORY_CAR_DOCK
I146	CATEGORY_CAR_MODE
I147	CATEGORY_DEFAULT
I148	CATEGORY_DESK_DOCK
I149	CATEGORY_DEVELOPMENT_PREFERENCE

I150	CATEGORY_EMBED
I151	CATEGORY_FRAMEWORK_INSTRUMENTATION_TEST
I152	CATEGORY_HE_DESK_DOCK
I153	CATEGORY_HOME
I154	CATEGORY_INFO
I155	CATEGORY_LAUNCHER
I157	CATEGORY_LE_DESK_DOCK
I158	CATEGORY_MONKEY
I159	CATEGORY_OPENABLE
I160	CATEGORY_PREFERENCE
I161	CATEGORY_SAMPLE_CODE
I163	CATEGORY_SELECTED_ALTERNATIVE
I164	CATEGORY_TAB
I165	CATEGORY_TEST
I167	CATEGORY_UNIT_TEST
I259	FLAG_ACTIVITY_CLEAR_TOP
I273	FLAG_ACTIVITY_REORDER_TO_FRONT
I276	FLAG_ACTIVITY_SINGLE_TOP
API Calls	
ID	API Call
A001	android.accounts
A002	android.annotation.SuppressLint
A003	android.annotation.TargetApi
A004	android.app.ActivityManager
A005	android.app.admin.DeviceAdminReceiver
A006	android.app.admin.DevicePolicyManager
A007	android.app.AlarmManager
A008	android.app.AlertDialog
A009	android.app.AlertDialog.Builder
A010	android.app.Application
A011	android.app.backup
A012	android.app.DownloadManager
A013	android.app.Instrumentation
A014	android.app.IntentService
A015	android.app.KeyguardManager
A016	android.app.LauncherActivity
A017	android.app.ListFragment
A018	android.app.LoaderManager
A019	android.app.LocalActivityManager
A020	android.app.NativeActivity
A021	android.app.Notification
A022	android.app.PendingIntent
A023	android.app.SearchableInfo
A024	android.app.SearchManager
A025	android.app.Service
A026	android.app.TabActivity
A027	android.app.TaskStackBuilder
A028	android.bluetooth
A029	android.content.AbstractThreadedSyncAdapter
A030	android.content.ActivityNotFoundException
A031	android.content.AsyncQueryHandler
A032	android.content.BroadcastReceiver
A033	android.content.ClipboardManager
A034	android.content.ClipData
A035	android.content.ComponentCallbacks
A036	android.content.ComponentName
A037	android.content.ContentProvider
A038	android.content.ContentResolver
A039	android.content.ContentUris
A040	android.content.ContentValues
A041	android.content.ContextWrapper
A042	android.content.CursorLoader
A043	android.content.DialogInterface
A044	android.content.Entity
A045	android.content.Intent.ShortcutIconResource
A046	android.content.IntentFilter

A047	android.content.IntentSender
A048	android.content.Loader
A049	android.content.OperationApplicationException
A050	android.content.pm.ActivityInfo
A051	android.content.pm.ApplicationInfo
A052	android.content.pm.FeatureInfo
A053	android.content.pm.IPackageStatsObserver
A054	android.content.pm.LabeledIntent
A055	android.content.pm.PackageInfo
A056	android.content.pm.PackageManager
A057	android.content.pm.PackageStats
A058	android.content.pm.PermissionInfo
A059	android.content.pm.ProviderInfo
A060	android.content.pm.ResolveInfo
A061	android.content.pm.ServiceInfo
A062	android.content.pm.Signature
A063	android.content.res.AssetFileDescriptor
A064	android.content.res.AssetManager
A065	android.content.res.ColorStateList
A066	android.content.res.Configuration
A067	android.content.res.Resources
A068	android.content.res.XmlResourceParser
A069	android.content.SearchRecentSuggestionsProvider
A070	android.content.ServiceConnection
A071	android.content.SharedPreferences
A072	android.content.SyncResult
A073	android.content.UriMatcher
A074	android.database
A075	android.hardware.Camera
A076	android.hardware.display.DisplayManager
A077	android.hardware.GeomagneticField
A078	android.hardware.Sensor
A079	android.location.Address
A080	android.location.Criteria
A081	android.location.Geocoder
A082	android.location.Gps
A083	android.location.Location
A084	android.media.AsyncPlayer
A085	android.media.AudioManager
A086	android.media.AudioRecord
A087	android.media.AudioTrack
A088	android.media.CamcorderProfile
A089	android.media.ExifInterface
A090	android.media.FaceDetector
A091	android.media.JetPlayer
A092	android.media.MediaMetadataRetriever
A093	android.media.MediaPlayer
A094	android.media.MediaRecorder
A095	android.media.MediaScannerConnection
A096	android.media.RemoteControlClient
A097	android.media.Ringtone
A098	android.media.SoundPool
A099	android.media.ThumbnailUtils
A100	android.media.ToneGenerator
A101	android.net.ConnectivityManager
A102	android.net.DhcpInfo
A103	android.net.http.AndroidHttpClient
A104	android.net.http.HttpResponseCache
A105	android.net.http.SslCertificate
A106	android.net.LocalServerSocket
A107	android.net.LocalSocket
A108	android.net.LocalSocketAddress
A109	android.net.MailTo
A110	android.net.NetworkInfo
A111	android.net.Proxy
A112	android.net.SSLCertificateSocketFactory

A113	android.net.TrafficStats
A114	android.net.Uri
A115	android.net.UrlQuerySanitizer
A116	android.net.wifi
A117	android.nfc
A118	android.os.Binder
A119	android.os.Build
A120	android.os.CancellationSignal
A121	android.os.ConditionVariable
A122	android.os.CountDownTimer
A123	android.os.DeadObjectException
A124	android.os.Debug
A125	android.os.Environment
A126	android.os.FileObserver
A127	android.os.Handler
A128	android.os.IBinder
A129	android.os.IInterface
A130	android.os.Looper
A131	android.os.MemoryFile
A132	android.os.Message
A133	android.os.Messenger
A134	android.os.Parcel
A135	android.os.Parcelable
A136	android.os.ParcelFileDescriptor
A137	android.os.PowerManager
A138	android.os.Process
A139	android.os.RemoteCallbackList
A140	android.os.RemoteException
A141	android.os.ResultReceiver
A142	android.os.ServiceManager
A143	android.os.StatFs
A144	android.os.StrictMode
A145	android.os.SystemClock
A146	android.os.Vibrator
A147	android.preference.CheckBoxPreference
A148	android.preference.DialogPreference
A149	android.preference.EditTextPreference
A150	android.preference.ListPreference
A151	android.preference.Preference
A152	android.preference.RingtonePreference
A153	android.provider.BaseColumns
A154	android.provider.Browser
A155	android.provider.CalendarContract
A156	android.provider.CallLog.Calls
A157	android.provider.Contacts
A158	android.provider.MediaStore
A159	android.provider.SearchRecentSuggestions
A160	android.provider.Settings
A161	android.sax
A162	android.service.dreams.DreamService
A163	android.service.wallpaper.WallpaperService
A164	android.speech.RecognitionListener
A165	android.speech.SpeechRecognizer
A166	android.speech.tts.TextToSpeech
A167	android.support.v4.app
A168	android.support.v4.content
A169	android.support.v4.media.TransportMediator
A170	android.support.v4.os.EnvironmentCompat
A171	android.support.v4.os.ParcelableCompat
A172	android.support.v4.os.ParcelableCompatCreatorCallbacks
A173	android.support.v4.util.DebugUtils
A174	android.support.v4.util.LogWriter
A175	android.support.v4.util.LruCache
A176	android.support.v4.util.SimpleArrayMap
A177	android.support.v4.util.SparseArrayCompat
A178	android.support.v4.util.TimeUtils

A179	android.support.v7.app
A180	android.telephony
A181	android.util.AndroidException
A182	android.util.AndroidRuntimeException
A183	android.util.AttributeSet
A184	android.util.Base64
A185	android.util.Config
A186	android.util.DisplayMetrics
A187	android.util.EventLog
A188	android.util.FloatMath
A189	android.util.JsonReader
A190	android.util.JsonWriter
A191	android.util.Log
A192	android.util.LruCache
A193	android.util.MonthDisplayHelper
A194	android.util.Pair
A195	android.util.Patterns
A196	android.util.SparseArray
A197	android.util.SparseBooleanArray
A198	android.util.SparseIntArray
A199	android.util.StateSet
A200	android.util.TypedValue
A201	android.util.Xml
A202	android.util.Xml.Encoding
A203	android.webkit

Appendix C

Heatmap Data of Feature Ranking by Dataset.

ID	Feature Name	PIA	PI	PA	IA	P	I	A
P079	INSTALL_SHORTCUT	9	10	10		10		
P112	RECEIVE_SMS	8	11	9		11		
P139	SYSTEM_ALERT_WINDOW	9	10	10		10		
I078	ACTION_PACKAGE_ADDED	9	10		9		11	
I106	ACTION_SENDTO	9	11		9		10	
P105	READ_SMS	9	10	9		10		
I012	ACTION_BOOT_COMPLETED	8	10		9		11	
A111	android.net.Proxy	9		9	10			10
I153	CATEGORY_HOME	9	9		9		10	
A180	android.telephony	8		9	10			10
P104	READ_PHONE_STATE	7	10	8		11		
A022	android.app.PendingIntent	9		9	9			9
A045	android.content.Intent.ShortcutIconResource	8		9	9			10
P100	READ_EXTERNAL_STORAGE	8	9	8		10		
P153	WRITE_EXTERNAL_STORAGE	8	8	9		9		
P005	ACCESS_FINE_LOCATION	7	9	8		8		
P010	ACCESS_WIFI_STATE	5	9	7		11		
P008	ACCESS_NETWORK_STATE	6	9	7		9		
A003	android.annotation.TargetApi	8		8	7			8
A025	android.app.Service	7		8	8			8
A051	android.content.pm.ApplicationInfo	8		8	8			7
A055	android.content.pm.PackageInfo	9		7	7			8
P004	ACCESS_COARSE_LOCATION	6	7	8		9		
P082	KILL_BACKGROUND_PROCESSES	5	10	5		10		
A154	android.provider.Browser	6		7	7			9
P006	ACCESS_LOCATION_EXTRA_COMMANDS	4	8	6		10		
P110	RECEIVE_BOOT_COMPLETED	7	7	6		8		
I131	ACTION_WEB_SEARCH	6	7		7		8	
A046	android.content.IntentFilter	6		8	7			7
A061	android.content.pm.ServiceInfo	6		7	7			8
P099	READ_CONTACTS	6	8	4		9		
A062	android.content.pm.Signature	7		6	6			8
A128	android.os.IBinder	8		7	8			4
P113	RECEIVE_WAP_PUSH	6	7	6		7		
I026	ACTION_DELETE	5	7		5		9	
I079	ACTION_PACKAGE_CHANGED	5	8		5		8	
I083	ACTION_PACKAGE_INSTALL	6	7		6		7	
I090	ACTION_PICK	5	7		7		7	
A039	android.content.ContentUris	6		8	4			8
A116	android.net.wifi	4		6	6			10
A134	android.os.Parcel	4		8	7			7
A191	android.util.Log	6		5	7			8
P045	BLUETOOTH_ADMIN	4	7	6		8		
P053	CALL_PHONE	4	8	5		8		
P061	CHANGE_WIFI_STATE	6	8	3		8		
I085	ACTION_PACKAGE_REMOVED	6	8		4		7	
I086	ACTION_PACKAGE_REPLACED	5	8		5		7	
A004	android.app.ActivityManager	5		5	6			9
A032	android.content.BroadcastReceiver	5		6	6			8
A130	android.os.Looper	5		8	4			8
P111	RECEIVE_MMS	5	7	5		7		
P147	VIBRATE	4	7	4		9		
I027	ACTION_DEVICE_STORAGE_LOW	4	7		5		8	
I029	ACTION_DIAL	4	7		4		9	
I105	ACTION_SEND	3	8		4		9	
I159	CATEGORY_OPENABLE	3	7		5		9	
A194	android.util.Pair	6		5	7			6
P069	EXPAND_STATUS_BAR	3	7	6		7		

I044	ACTION_INSERT_OR_EDIT	3	8		4		8	
I124	ACTION_USER_PRESENT	5	6		5		7	
I126	ACTION_VIEW	4	5		6		8	
I144	CATEGORY_BROWSABLE	3	7		5		8	
A024	android.app.SearchManager	6		5	5			7
A073	android.content.UriMatcher	4		8	4			7
A075	android.hardware.Camera	3		4	8			8
A085	android.media.AudioManager	5		4	6			8
A101	android.net.ConnectivityManager	7		7	5			4
P134	SET_WALLPAPER_HINTS	4	7	4		7		
P148	WAKE_LOCK	4	4	5		9		
I101	ACTION_SCREEN_OFF	2	7		4		9	
A021	android.app.Notification	4		6	5			7
A030	android.content.ActivityNotFoundException	4		4	7			7
A047	android.content.IntentSender	4		6	5			7
A064	android.content.res.AssetManager	5		7	4			6
A120	android.os.CancellationSignal	5		5	6			6
A135	android.os.Parcelable	4		3	7			8
A151	android.preference.Preference	4		6	5			7
A176	android.support.v4.util.SimpleArrayMap	5		5	6			6
P044	BLUETOOTH	4	5	4		8		
P058	CHANGE_CONFIGURATION	2	8	3		8		
P062	CLEAR_APP_CACHE	2	7	4		8		
P081	INTERNET	2	7	3		9		
P088	MODIFY_AUDIO_SETTINGS	3	8	2		8		
I091	ACTION_PICK_ACTIVITY	2	7		5		7	
I130	ACTION_WALLPAPER_CHANGED	3	5		5		8	
I154	CATEGORY_INFO	3	7		3		8	
I158	CATEGORY_MONKEY	2	7		4		8	
A040	android.content.ContentValues	5		5	5			6
A068	android.content.res.XmlResourceParser	3		4	7			7
A081	android.location.Geocoder	3		6	4			8
A197	android.util.SparseBooleanArray	4		6	5			6
P059	CHANGE_NETWORK_STATE	2	8	2		8		
P114	RECORD_AUDIO	3	5	4		8		
I022	ACTION_CREATE_SHORTCUT	2	5		6		7	
I102	ACTION_SCREEN_ON	2	7		3		8	
I108	ACTION_SET_WALLPAPER	2	6		5		7	
A056	android.content.pm.PackageManager	5		5	5			5
A093	android.media.MediaPlayer	2		5	6			7
A127	android.os.Handler	4		5	5			6
A143	android.os.StatFs	4		4	8			4
P050	BROADCAST_STICKY	2	6	3		8		
P074	GET_PACKAGE_SIZE	1	7	4		7		
P156	WRITE_SETTINGS	3	5	3		8		
I010	ACTION_BATTERY_LOW	3	5		3		8	
I011	ACTION_BATTERY_OKAY	4	4		3		8	
I014	ACTION_CALL	3	6		3		7	
I019	ACTION_CLOSE_SYSTEM_DIALOGS		8		3		8	
I023	ACTION_DATE_CHANGED	1	6		4		8	
I147	CATEGORY_DEFAULT	3	5		2		9	
A007	android.app.AlarmManager	5		4	4			6
A008	android.app.AlertDialog	4		4	6			5
A037	android.content.ContentProvider	3		6	5			5
A110	android.net.NetworkInfo	5		4	5			5
A125	android.os.Environment	4		4	4			7
A186	android.util.DisplayMetrics	5		4	5			5
A202	android.util.Xml.Encoding	4		4	6			5
P133	SET_WALLPAPER	2	5	3		8		
I058	ACTION_MEDIA_BUTTON	1	7		3		7	
I061	ACTION_MEDIA_MOUNTED	1	6		3		8	
A060	android.content.pm.ResolveInfo	1		6	4			7
A066	android.content.res.Configuration	4		4	5			5
A079	android.location.Address	4		3	5			6
A132	android.os.Message	4		4	5			5
A137	android.os.PowerManager	5		4	4			5

A148	android.preference.DialogPreference	4		3	5			6
P067	DISABLE_KEYGUARD	1	7	2		7		
P092	NFC	3	4	3		7		
P115	REORDER_TASKS	3	4	3		7		
P125	SET_ALARM	2	5	3		7		
P157	WRITE_SYNC_SETTINGS	3	3	3		8		
I033	ACTION_EDIT	1	5		2		9	
I037	ACTION_GET_CONTENT	1	4		4		8	
I103	ACTION_SEARCH	1	6		2		8	
I104	ACTION_SEARCH_LONG_PRESS		6		3		8	
I150	CATEGORY_EMBED		6		3		8	
I155	CATEGORY_LAUNCHER	1	6		2		8	
A014	android.app.IntentService	4		4	4			5
A080	android.location.Criteria	5		4	4			4
A083	android.location.Location	5		2	6			4
A094	android.media.MediaRecorder	2		4	4			7
A123	android.os.DeadObjectException	4		2	5			6
A138	android.os.Process	4		4	5			4
A146	android.os.Vibrator	2		3	7			5
A157	android.provider.Contacts	3		5	3			6
A158	android.provider.MediaStore	2		3	4			8
A161	android.sax	3		4	5			5
A201	android.util.Xml	4		4	4			5
P055	CAMERA	1	3	3		9		
P106	READ_SYNC_SETTINGS	1	4	3		8		
I049	ACTION_MAIN	2	5		2		7	
I065	ACTION_MEDIA_SCANNER_SCAN_FILE	1	8				7	
A010	android.app.Application	3		4	4			5
A031	android.content.AsyncQueryHandler	3		6	2			5
A038	android.content.ContentResolver	2		5	3			6
A043	android.content.DialogInterface	6		6	2			2
A071	android.content.SharedPreferences	3		4	5			4
A118	android.os.Binder	2		4	3			7
A160	android.provider.Settings	4		4	4			4
P028	BIND_INPUT_METHOD	1	4	3		7		
P060	CHANGE_WIFI_MULTICAST_STATE	2	3	3		7		
I018	ACTION_CHOOSER		4		3		8	
I043	ACTION_INSERT		6		1		8	
I094	ACTION_POWER_USAGE_SUMMARY	1	6				8	
I096	ACTION_PROVIDER_CHANGED		7		1		7	
I111	ACTION_SYNC		6		1		8	
A019	android.app.LocalActivityManager	4		4	3			4
A053	android.content.pm.IPackageStatsObserver	4		4	4			3
A063	android.content.res.AssetFileDescriptor	4		4	3			4
A078	android.hardware.Sensor	2		4	4			5
P015	BATTERY_STATS	1	4	2		7		
P072	GET_ACCOUNTS		4	1		9		
I060	ACTION_MEDIA_EJECT		5		3		6	
I066	ACTION_MEDIA_SCANNER_STARTED		6				8	
I080	ACTION_PACKAGE_DATA_CLEARED		7				7	
I107	ACTION_SEND_MULTIPLE	2	4		1		7	
I113	ACTION_TIMEZONE_CHANGED		6				8	
I132	CATEGORY_ALTERNATIVE	1	5		1		7	
A067	android.content.res.Resources	2		4	2			6
A136	android.os.ParcelFileDescriptor	3		3	4			4
A141	android.os.ResultReceiver	1		4	2			7
A177	android.support.v4.util.SparseArrayCompat	3		4	3			4
A196	android.util.SparseArray	2		4	3			5
A200	android.util.TypedValue	4		3	3			4
P043	BIND_WALLPAPER		5			8		
P096	PROCESS_OUTGOING_CALLS		6			7		
P097	READ_CALENDAR		4	1		8		
P150	WRITE_CALENDAR		2	3		8		
P152	WRITE_CONTACTS		3	1		9		
I129	ACTION_VOICE_COMMAND		6				7	
I160	CATEGORY_PREFERENCE	1	3		1		8	

A016	android.app.LauncherActivity	3		4	3			3
A044	android.content.Entity	4		3	3			3
A097	android.media.Ringtone			2	3			8
A109	android.net.MailTo	3		2	4			4
A112	android.net.SSLCertificateSocketFactory	2		4	3			4
A114	android.net.Uri	2		3	4			4
A115	android.net.UrlQuerySanitizer	3		3	3			4
A129	android.os.IInterface	3		2	3			5
A139	android.os.RemoteCallbackList	4		4	3			2
A140	android.os.RemoteException	4		3	3			3
A170	android.support.v4.os.EnvironmentCompat	2		4	3			4
P025	BIND_DEVICE_ADMIN		4				8	
P107	READ_SYNC_STATS	1	4	1			6	
P151	WRITE_CALL_LOG	1	3	1			7	
I008	ACTION_ATTACH_DATA		4				8	
I015	ACTION_CALL_BUTTON		4		1		7	
I064	ACTION_MEDIA_SCANNER_FINISHED		4				8	
I087	ACTION_PACKAGE_RESTARTED	1	2		1		8	
I100	ACTION_RUN		5				7	
A005	android.app.admin.DeviceAdminReceiver	4		2	3			3
A036	android.content.ComponentName	2		2	2			6
A059	android.content.pm.ProviderInfo	3		3	4			2
A074	android.database	2		4	2			4
A082	android.location.Gps	1		3	3			5
A119	android.os.Build	2		2	3			5
A145	android.os.SystemClock	2		3	3			4
A147	android.preference.CheckBoxPreference	3		2	4			3
A203	android.webkit	2		3	3			4
P016	BIND_ACCESSIBILITY_SERVICE		4				7	
P034	BIND_REMOTEVIEWS		3	1			7	
P064	DELETE_CACHE_FILES	1	3				7	
I002	ACTION_ALL_APPS		4		1		6	
I016	ACTION_CAMERA_BUTTON		3		1		7	
I020	ACTION_CONFIGURATION_CHANGED		5		1		5	
I024	ACTION_DEFAULT		2		1		8	
I035	ACTION_EXTERNAL_APPLICATIONS_UNAVAILABLE		3				8	
I042	ACTION_INPUT_METHOD_CHANGED		3				8	
I062	ACTION_MEDIA_NOFS	1	2		1		7	
I069	ACTION_MEDIA_UNMOUNTED		4				7	
I099	ACTION_REBOOT		3				8	
I115	ACTION_TIME_TICK	1	2				8	
I161	CATEGORY_SAMPLE_CODE		3				8	
A054	android.content.pm.LabeledIntent	2		2	3			4
A070	android.content.ServiceConnection	3		2	2			4
A117	android.nfc	1		2	4			4
A124	android.os.Debug	1		4	1			5
A153	android.provider.BaseColumns	2		3	2			4
A163	android.service.wallpaper.WallpaperService	1		3	2			5
P076	GLOBAL_SEARCH		2	1			7	
I003	ACTION_ANSWER		2		1		7	
I034	ACTION_EXTERNAL_APPLICATIONS_AVAILABLE		2				8	
I041	ACTION_HEADSET_PLUG		5				5	
I047	ACTION_LOCALE_CHANGED		3				7	
I057	ACTION_MEDIA_BAD_REMOVAL		4				6	
I063	ACTION_MEDIA_REMOVED		3				7	
I117	ACTION_UID_REMOVED		2		1		7	
I163	CATEGORY_SELECTED_ALTERNATIVE		2		1		7	
I164	CATEGORY_TAB	1	2		1		6	
A002	android.annotation.SuppressLint	2		2	3			3
A009	android.app.AlertDialog.Builder	2		3	2			3
A026	android.app.TabActivity	2		2	1			5
A057	android.content.pm.PackageStats			2	2			6
A105	android.net.http.SslCertificate			4	2			4
A126	android.os.FileObserver	2		3	2			3
A169	android.support.v4.media.TransportMediator	2		3	2			3
A188	android.util.FloatMath	1		4	1			4

A199	android.util.StateSet			3	2			5
I009	ACTION_BATTERY_CHANGED		2		1		6	
I046	ACTION_INSTALL_PACKAGE	1	3				5	
I056	ACTION_MANAGE_PACKAGE_STORAGE		2				7	
I068	ACTION_MEDIA_UNMOUNTABLE		3				6	
I070	ACTION_MY_PACKAGE_REPLACED				2		7	
I112	ACTION_SYSTEM_TUTORIAL		4		1		4	
I119	ACTION_UMS_DISCONNECTED		3				6	
I149	CATEGORY_DEVELOPMENT_PREFERENCE		2		1		6	
I167	CATEGORY_UNIT_TEST		3		1		5	
A001	android.accounts	3		1	2			3
A012	android.app.DownloadManager			2	2			5
A069	android.content.SearchRecentSuggestionsProvider	1		2	1			5
A185	android.util.Config			3	1			5
P012	ACTIVITY_RECOGNITION		1	1			6	
P013	ADD_VOICEMAIL		1	1			6	
P041	BIND_VPN_SERVICE		1	1			6	
P098	READ_CALL_LOG		1				7	
P146	USE_SIP			1			7	
I013	ACTION_BUG_REPORT		2				6	
I059	ACTION_MEDIA_CHECKING		1				7	
I067	ACTION_MEDIA_SHARED		1				7	
I073	ACTION_NEW_OUTGOING_CALL		2				6	
I118	ACTION_UMS_CONNECTED		1				7	
I148	CATEGORY_DESK_DOCK	1					7	
I151	CATEGORY_FRAMEWORK_INSTRUMENTATION_TEST		3				5	
I165	CATEGORY_TEST		2				6	
A035	android.content.ComponentCallbacks	1		1	1			5
A049	android.content.OperationApplicationException	1		2	2			3
A076	android.hardware.display.DisplayManager	1		2	3			2
A096	android.media.RemoteControlClient	1		2	3			2
A098	android.media.SoundPool	1		1	2			4
A142	android.os.ServiceManager	1		1	1			5
A166	android.speech.tts.TextToSpeech			1	2			5
A181	android.util.AndroidException			2	1			5
A183	android.util.AttributeSet	2		2	2			2
A184	android.util.Base64			2	1			5
A198	android.util.SparseIntArray			3	3			2
P031	BIND_NOTIFICATION_LISTENER_SERVICE		1				6	
P037	BIND_TEXT_SERVICE	1		1			5	
P094	PACKAGE_USAGE_STATS						7	
I030	ACTION_DOCK_EVENT						7	
A013	android.app.Instrumentation	1			3			3
A018	android.app.LoaderManager	1		1	1			4
A050	android.content.pm.ActivityInfo			3	1			3
A058	android.content.pm.PermissionInfo	3		1	2			1
A065	android.content.res.ColorStateList	1		2				4
A099	android.media.ThumbnailUtils			2	1			4
A102	android.net.DhcpInfo			1				6
A167	android.support.v4.app	2		1	2			2
I028	ACTION_DEVICE_STORAGE_OK						6	
I145	CATEGORY_CAR_DOCK						6	
A103	android.net.http.AndroidHttpClient	2			3			1
A122	android.os.CountDownTimer	3			3			
A131	android.os.MemoryFile			1				5
A149	android.preference.EditTextPreference	1		2	1			2
A174	android.support.v4.util.LogWriter	1		1	1			3
A192	android.util.LruCache	1		1	1			3
I001	ACTION_AIRPLANE_MODE_CHANGED		1		1		3	
I055	ACTION_MANAGE_NETWORK_USAGE		1				4	
I093	ACTION_POWER_DISCONNECTED		1				4	
I141	CATEGORY_APP_MARKET		1				4	
A006	android.app.admin.DevicePolicyManager				3			2
A015	android.app.KeyguardManager	2						3
A023	android.app.SearchableInfo	1		1	1			2
A041	android.content.ContextWrapper	1		3				1

A052	android.content.pm.FeatureInfo				2				3
A087	android.media.AudioTrack				1	2			2
A113	android.net.TrafficStats				1	1			3
A121	android.os.ConditionVariable	1			1	1			2
A133	android.os.Messenger					1			4
A150	android.preference.ListPreference	1			1	2			1
A159	android.provider.SearchRecentSuggestions				1				4
A164	android.speech.RecognitionListener				1				4
A165	android.speech.SpeechRecognizer				2				3
A168	android.support.v4.content	1			2	1			1
A173	android.support.v4.util.DebugUtils				1				4
A182	android.util.AndroidRuntimeException	1			2	1			1
A193	android.util.MonthDisplayHelper	1			1				3
A195	android.util.Patterns	2			1	1			1
I081	ACTION_PACKAGE_FIRST_LAUNCH							4	
I143	CATEGORY_APP_MUSIC					1		3	
A011	android.app.backup	1				2			1
A028	android.bluetooth	1			2				1
A033	android.content.ClipboardManager	1			2				1
A084	android.media.AsyncPlayer	1			1	1			1
A089	android.media.ExifInterface				1	1			2
A107	android.net.LocalSocket	1			1				2
A155	android.provider.CalendarContract	1			1				2
I006	ACTION_APP_ERROR			1					2
I074	ACTION_OPEN_DOCUMENT			1		1		1	
I082	ACTION_PACKAGE_FULLY_REMOVED			1		1		1	
I092	ACTION_POWER_CONNECTED			1				2	
I097	ACTION_QUICK_CLOCK			1		1		1	
I114	ACTION_TIME_CHANGED							3	
I146	CATEGORY_CAR_MODE					1		2	
A020	android.app.NativeActivity	1			1				1
A027	android.app.TaskStackBuilder	1			1	1			
A034	android.content.ClipData	1			1				1
A095	android.media.MediaScannerConnection					1			2
A100	android.media.ToneGenerator	1				1			1
A108	android.net.LocalSocketAddress	1				1			1
A144	android.os.StrictMode	1			1				1
A156	android.provider.CallLog.Calls				3				
A179	android.support.v7.app				1	1			1
I007	ACTION_ASSIST			1				1	
I031	ACTION_DREAMING_STARTED					1		1	
I036	ACTION_FACTORY_TEST	1						1	
I110	ACTION_SHUTDOWN					1		1	
I116	ACTION_TRANSLATE	1						1	
I120	ACTION_UNINSTALL_PACKAGE							2	
I133	CATEGORY_APP_BROWSER					1		1	
I136	CATEGORY_APP_CONTACTS					1		1	
I140	CATEGORY_APP_MAPS	1						1	
I157	CATEGORY_LE_DESK_DOCK			1				1	
I259	FLAG_ACTIVITY_CLEAR_TOP	1	1						
I276	FLAG_ACTIVITY_SINGLE_TOP					1		1	
A017	android.app.ListFragment	1							1
A029	android.content.AbstractThreadedSyncAdapter	1							1
A048	android.content.Loader	1							1
A077	android.hardware.GeomagneticField	1							1
A086	android.media.AudioRecord					1			1
A104	android.net.http.HttpResponseCache	1							1
A171	android.support.v4.os.ParcelableCompat				1				1
A172	android.support.v4.os.ParcelableCompatCreatorCallbacks				1	1			
A187	android.util.EventLog				1				1
I032	ACTION_DREAMING_STOPPED							1	
I084	ACTION_PACKAGE_NEEDS_VERIFICATION							1	
I089	ACTION_PASTE							1	
I121	ACTION_USER_BACKGROUND					1			
I123	ACTION_USER_INITIALIZE							1	
I135	CATEGORY_APP_CALENDAR							1	

Appendix D

Heatmap Data of Feature Ranking by Algorithm.

FID	Feature Name	InfoGain	Chi2	Relief	CFS-InfoGain	CFS-Chi2	CFS-Relief	RFE-ANN	RFE-RF	RFE-SVM	LASSO	Ridge
P079	INSTALL_SHORTCUT	4	4	4	4	4		4	4	3	4	4
P112	RECEIVE_SMS	4	4	3	4	4	2	4	4	2	4	4
P139	SYSTEM_ALERT_WINDOW	4	4	3	4	4		4	4	4	4	4
I078	ACTION_PACKAGE_ADDED	4	4	4	4	4	2	4	4	1	4	4
I106	ACTION_SENDTO	4	4	4	4	4	1	4	4	2	4	4
P105	READ_SMS	4	4	2	4	4		4	4	4	4	4
I012	ACTION_BOOT_COMPLETED	4	4	4	4	4	2	3	4	1	4	4
A111	android.net.Proxy	4	4	4	4	4		4	4	2	4	4
I153	CATEGORY_HOME	4	4	4	4	4	1	4	4		4	4
A180	android.telephony	4	4	4	2	1	4	4	4	2	4	4
P104	READ_PHONE_STATE	4	4	4	4	1	2	3	4	2	4	4
A022	android.app.PendingIntent	4	4	4	4		2	4	4	2	4	4
A045	android.content.Intent.ShortcutIconResource	4	4	4	3	3		4	4	2	4	4
P100	READ_EXTERNAL_STORAGE	4	4	2	4	4		4	4	1	4	4
P153	WRITE_EXTERNAL_STORAGE	4	4	4			4	4	4	2	4	4
P005	ACCESS_FINE_LOCATION	4	4	4			1	3	4	4	4	4
P010	ACCESS_WIFI_STATE	4	4	4	2	1	2	2	4	2	4	3
P008	ACCESS_NETWORK_STATE	4	4	4			2	3	4	2	4	4
A003	android.annotation.TargetApi	4	4	4				4	4	3	4	4
A025	android.app.Service	4	4	4				4	4	3	4	4
A051	android.content.pm.ApplicationInfo	4	4	4				4	4	3	4	4
A055	android.content.pm.PackageInfo	4	4	4			1	4	4	2	4	4
P004	ACCESS_COARSE_LOCATION	4	4	4			1	2	4	4	4	3
P082	KILL_BACKGROUND_PROCESSES	4	4	2	4	4		2	4	2	2	2
A154	android.provider.Browser	4	4	3	1	1		4	4		4	4
P006	ACCESS_LOCATION_EXTRA_COMMANDS	4	4	2	2	2		2	4	4	2	2
P110	RECEIVE_BOOT_COMPLETED	4	4	3				3	4	2	4	4
I131	ACTION_WEB_SEARCH	4	4	2	1			4	4	1	4	4
A046	android.content.IntentFilter	4	4	4				3	4	1	4	4
A061	android.content.pm.ServiceInfo	4	4	1				4	4	3	4	4
P099	READ_CONTACTS	4	4	4			2	3	3	1	3	3
A062	android.content.pm.Signature	4	4	4				4	1	2	4	4
A128	android.os.IBinder	4	4	4			1	3	4	1	3	3
P113	RECEIVE_WAP_PUSH	4	4					4	4	2	4	4
I026	ACTION_DELETE	4	4	2			1	4	2	1	4	4
I079	ACTION_PACKAGE_CHANGED	4	4	2				4	2	2	4	4
I083	ACTION_PACKAGE_INSTALL	4	4	2				4	4		4	4
I090	ACTION_PICK	3	3	4				4	4		4	4
A039	android.content.ContentUris	4	4	2				3	4	2	4	3
A116	android.net.wifi	4	4	4	1	2		1	4	2	2	2
A134	android.os.Parcel	3	3	4				4	3	1	4	4
A191	android.util.Log	4	1	4			3	4	2		4	4
P045	BLUETOOTH_ADMIN	3	3	2				4	2	3	4	4
P053	CALL_PHONE	4	4	3			1	1	4	4	2	2
P061	CHANGE_WIFI_STATE	4	4	2				3	4	2	3	3
I085	ACTION_PACKAGE_REMOVED	4	4	4			1	2	4		3	3
I086	ACTION_PACKAGE_REPLACED	4	4	2				4	2	1	4	4
A004	android.app.ActivityManager	4	4	4	1	1		1	4	2	2	2
A032	android.content.BroadcastReceiver	4	4	4				1	4	2	3	3
A130	android.os.Looper	4	4	4				2	4	3	2	2
P111	RECEIVE_MMS	4	4					4	2	2	4	4
P147	VIBRATE	4	4	4			2	1	4	1	2	2
I027	ACTION_DEVICE_STORAGE_LOW	3	2	1				4	2	4	4	4

I029	ACTION_DIAL	4	4	4			1	1	4	2	2	2
I105	ACTION_SEND	3	4	4			2	1	4	2	2	2
I159	CATEGORY_OPENABLE	3	3	2			1	4	2	1	4	4
A194	android.util.Pair	4	4	4				1	4		3	4
P069	EXPAND_STATUS_BAR	4	4					3	2	3	3	4
I044	ACTION_INSERT_OR_EDIT	3	2	2				4	2	2	4	4
I124	ACTION_USER_PRESENT	1	2	4				4	4		4	4
I126	ACTION_VIEW	4	4	4			1	1	3	1	3	2
I144	CATEGORY_BROWSABLE	2	3	2				3	4	2	4	3
A024	android.app.SearchManager	4	4	1				4		2	4	4
A073	android.content.UriMatcher	4	4	2				2	4	3	2	2
A075	android.hardware.Camera	3	3	4				2	2	3	3	3
A085	android.media.AudioManager	1	2	4				3	4	1	4	4
A101	android.net.ConnectivityManager	4	4	2				2	4	3	2	2
P134	SET_WALLPAPER_HINTS	4	4					3	2	2	4	3
P148	WAKE_LOCK	4	4	4				1	1	4	2	1
I101	ACTION_SCREEN_OFF	1	1	4			2	3	4	2	3	2
A021	android.app.Notification	4	4	3	3	2			4	1	1	
A030	android.content.ActivityNotFoundException	1	3	4			3	1	4	2	2	2
A047	android.content.IntentSender	4	3					4	1	2	4	4
A064	android.content.res.AssetManager	4	4	4				1	4		3	2
A120	android.os.CancellationSignal	4	4					4	1	1	4	4
A135	android.os.Parcelable	1	2	2				4	3	2	4	4
A151	android.preference.Preference		1	4			4	2	4		4	3
A176	android.support.v4.util.SimpleArrayMap	4	4					4	1	1	4	4
P044	BLUETOOTH	1	1	2				4	2	3	4	4
P058	CHANGE_CONFIGURATION	3	4	2				2	2	4	2	2
P062	CLEAR_APP_CACHE	4	4	1				3	2	2	2	3
P081	INTERNET	4	3	4			1	1	2	2	2	2
P088	MODIFY_AUDIO_SETTINGS	3	3	2				3	2	2	3	3
I091	ACTION_PICK_ACTIVITY	3	3	2				3	1	3	2	4
I130	ACTION_WALLPAPER_CHANGED	4	4	2				2	2	4	1	2
I154	CATEGORY_INFO	2	2	2				4	2	1	4	4
I158	CATEGORY_MONKEY	3	3	1				4	2	2	2	4
A040	android.content.ContentValues	4	4	4			4		4	1		
A068	android.content.res.XmlResourceParser	4	4	3					4	2	2	2
A081	android.location.Geocoder	4	4	2		1		2	4		2	2
A197	android.util.SparseBooleanArray	4	3					4		2	4	4
P059	CHANGE_NETWORK_STATE	4	4	2				2	2	2	2	2
P114	RECORD_AUDIO	4	4	2				1	4	3	1	1
I022	ACTION_CREATE_SHORTCUT	4	4	2				1	3	1	3	2
I102	ACTION_SCREEN_ON	4	4	2				2	3	1	2	2
I108	ACTION_SET_WALLPAPER	4	4	2				3	1	1	2	3
A056	android.content.pm.PackageManager			4				4	4		4	4
A093	android.media.MediaPlayer	3	3	4					4	2	3	1
A127	android.os.Handler			4				3	4	1	4	4
A143	android.os.StatFs	4	4	4				1	3	1	2	1
P050	BROADCAST_STICKY	1	1	2				4	2	2	3	4
P074	GET_PACKAGE_SIZE	3	4					3	2	2	2	3
P156	WRITE_SETTINGS	4	4	3				1	2	2	2	1
I010	ACTION_BATTERY_LOW	1	1	2				4	2	1	4	4
I011	ACTION_BATTERY_OKAY	1	1	1				4	1	3	4	4
I014	ACTION_CALL	4	4	4			1	1	2		2	1
I019	ACTION_CLOSE_SYSTEM_DIALOGS	3	3	2				2	2	2	3	2
I023	ACTION_DATE_CHANGED	2	2	2				4	2	2	2	3
I147	CATEGORY_DEFAULT	1	1	4			1	2	4	1	3	2
A007	android.app.AlarmManager	4	4	4		1			4	2		
A008	android.app.AlertDialog			4			1	2	4	3	2	3
A037	android.content.ContentProvider			3				4	3	1	4	4
A110	android.net.NetworkInfo	4	4	4					4	3		
A125	android.os.Environment	4	4	4			1		4		1	1
A186	android.util.DisplayMetrics	4	4	4			1		4	2		
A202	android.util.Xml.Encoding	4	2					4		1	4	4
P133	SET_WALLPAPER	4	4	2				1	2	3	1	1
I058	ACTION_MEDIA_BUTTON	2	2	2				3	2	1	3	3
I061	ACTION_MEDIA_MOUNTED	1	1	2				3	2	2	4	3

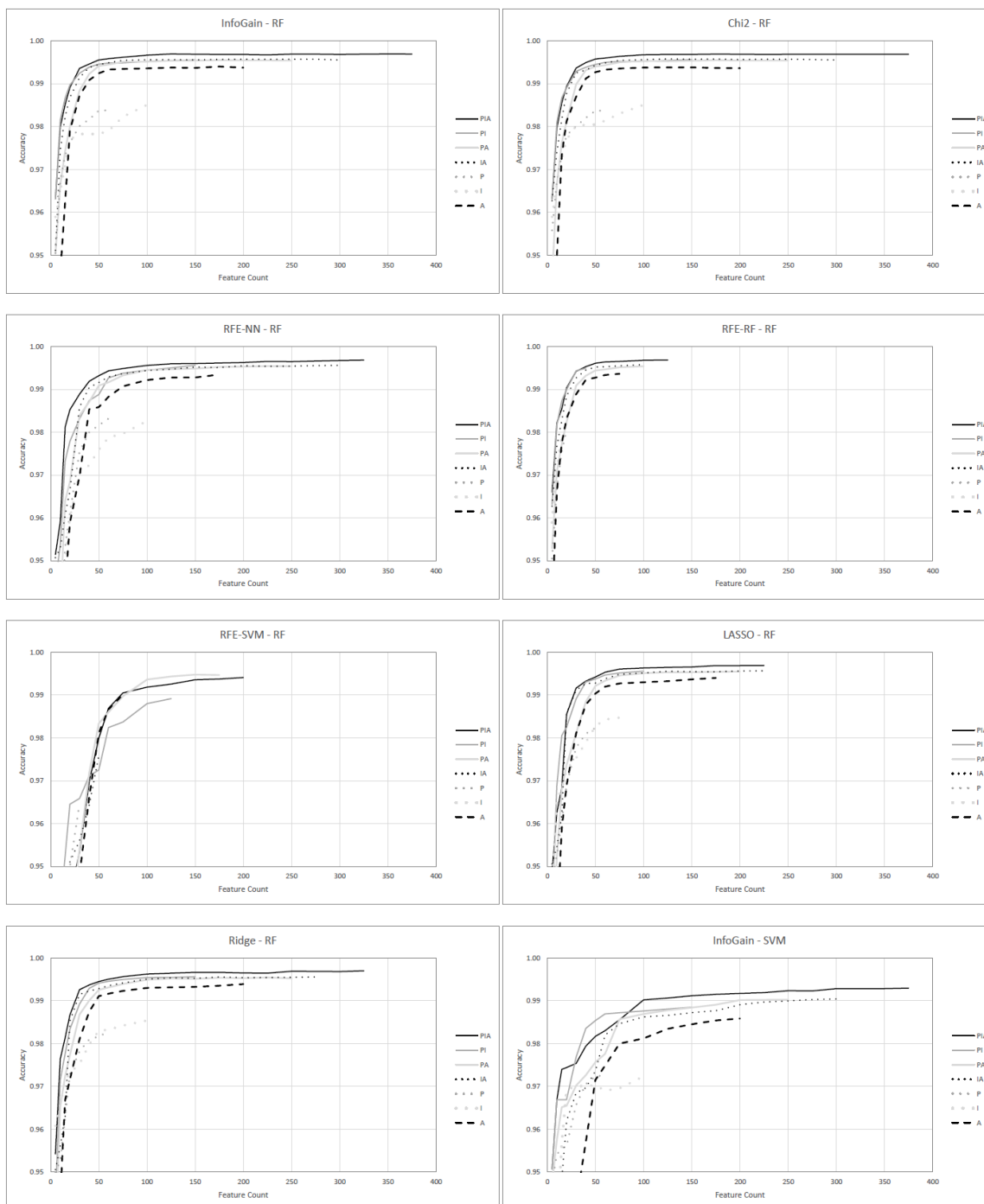
A060	android.content.pm.ResolveInfo	2	3	4				2	3		2	2
A066	android.content.res.Configuration	2	3	4			4		4	1		
A079	android.location.Address	4	4	2				1	4	3		
A132	android.os.Message	4	4	4			1		4	1		
A137	android.os.PowerManager	4	4	4					4	2		
A148	android.preference.DialogPreference	2	3	1				3		2	4	3
P067	DISABLE_KEYGUARD	2	2					2	2	4	3	2
P092	NFC	1	1					4	1	2	4	4
P115	REORDER_TASKS	1	1					4	1	2	4	4
P125	SET_ALARM	4	4					2	2	2	1	2
P157	WRITE_SYNC_SETTINGS	1	1	1				4	1	1	4	4
I033	ACTION_EDIT	1	1	2			1	1	4	3	2	2
I037	ACTION_GET_CONTENT	1	1	4			2		3	1	3	2
I103	ACTION_SEARCH	1	1	4			2	1	3	1	2	2
I104	ACTION_SEARCH_LONG_PRESS	2	2	2				3	1	1	3	3
I150	CATEGORY_EMBED	1	2	1				3	2	2	3	3
I155	CATEGORY_LAUNCHER	3	1	4				2	2	1	2	2
A014	android.app.IntentService	4	4	4					4	1		
A080	android.location.Criteria	4	4	4					4	1		
A083	android.location.Location			4				2	4	1	3	3
A094	android.media.MediaRecorder	4	4	1				1	3	1	2	1
A123	android.os.DeathObjectException	4	4					3		2	2	2
A138	android.os.Process	4	4	4					3	1	1	
A146	android.os.Vibrator	1	1	4			4	1	3	1	1	1
A157	android.provider.Contacts	1	1	4				1	4	2	3	1
A158	android.provider.MediaStore	3	4	2				1	4	1	1	1
A161	android.sax	3	2					4			4	4
A201	android.util.Xml	4	4	4					1	4		
P055	CAMERA	1	1	4			1	1	3	3	1	1
P106	READ_SYNC_SETTINGS	1	1	1				4	1	2	3	3
I049	ACTION_MAIN	3	1	4				2	1		2	3
I065	ACTION_MEDIA_SCANNER_SCAN_FILE	2	2	2				1	2	3	2	2
A010	android.app.Application	1	3	4			4		4			
A031	android.content.AsyncQueryHandler	4	4					2		2	2	2
A038	android.content.ContentResolver	1	3	4					4	2	2	
A043	android.content.DialogInterface			4				2	4	2	2	2
A071	android.content.SharedPreferences	4	4	4					2	2		
A118	android.os.Binder		1	2				1	4	1	4	3
A160	android.provider.Settings	4	4	4					4			
P028	BIND_INPUT_METHOD	1	1					4	1	2	3	3
P060	CHANGE_WIFI_MULTICAST_STATE	1	1					4	1	1	3	4
I018	ACTION_CHOOSER	3	2	1				3	1	2	1	2
I043	ACTION_INSERT	1	2	2				3	2	1	2	2
I094	ACTION_POWER_USAGE_SUMMARY	2	2	2				2	1	3	2	1
I096	ACTION_PROVIDER_CHANGED	2	2	2				2	1	1	3	2
I111	ACTION_SYNC	3	2	2				2	1	1	2	2
A019	android.app.LocalActivityManager							4		3	4	4
A053	android.content.pm.IPackageStatsObserver							4		3	4	4
A063	android.content.res.AssetFileDescriptor	1	1	4				2	1	1	3	2
A078	android.hardware.Sensor	3	4	4					3	1		
P015	BATTERY_STATS	1	1					3	2	1	2	4
P072	GET_ACCOUNTS	1	2	2			1	1	3	2	1	1
I060	ACTION_MEDIA_EJECT			2				3	2	3	1	3
I066	ACTION_MEDIA_SCANNER_STARTED	2	2	2				2	2	2	1	1
I080	ACTION_PACKAGE_DATA_CLEARED	2	2	2				2	2		2	2
I107	ACTION_SEND_MULTIPLE	1	1	2				2	4	1	2	1
I113	ACTION_TIMEZONE_CHANGED	1	2	2				2	2	1	2	2
I132	CATEGORY_ALTERNATIVE	1	1	2				3	1	2	2	2
A067	android.content.res.Resources			4			1	1	4	1	2	1
A136	android.os.ParcelFileDescriptor							4		2	4	4
A141	android.os.ResultReceiver	4	3					2	1	1	1	2
A177	android.support.v4.util.SparseArrayCompat			1				4		1	4	4
A196	android.util.SparseArray			2			1	1	3	2	2	3
A200	android.util.TypedValue		1	4					4	3	2	
P043	BIND_WALLPAPER	2	2	2				1	2	1	2	1
P096	PROCESS_OUTGOING_CALLS	2	2					2	2	1	2	2

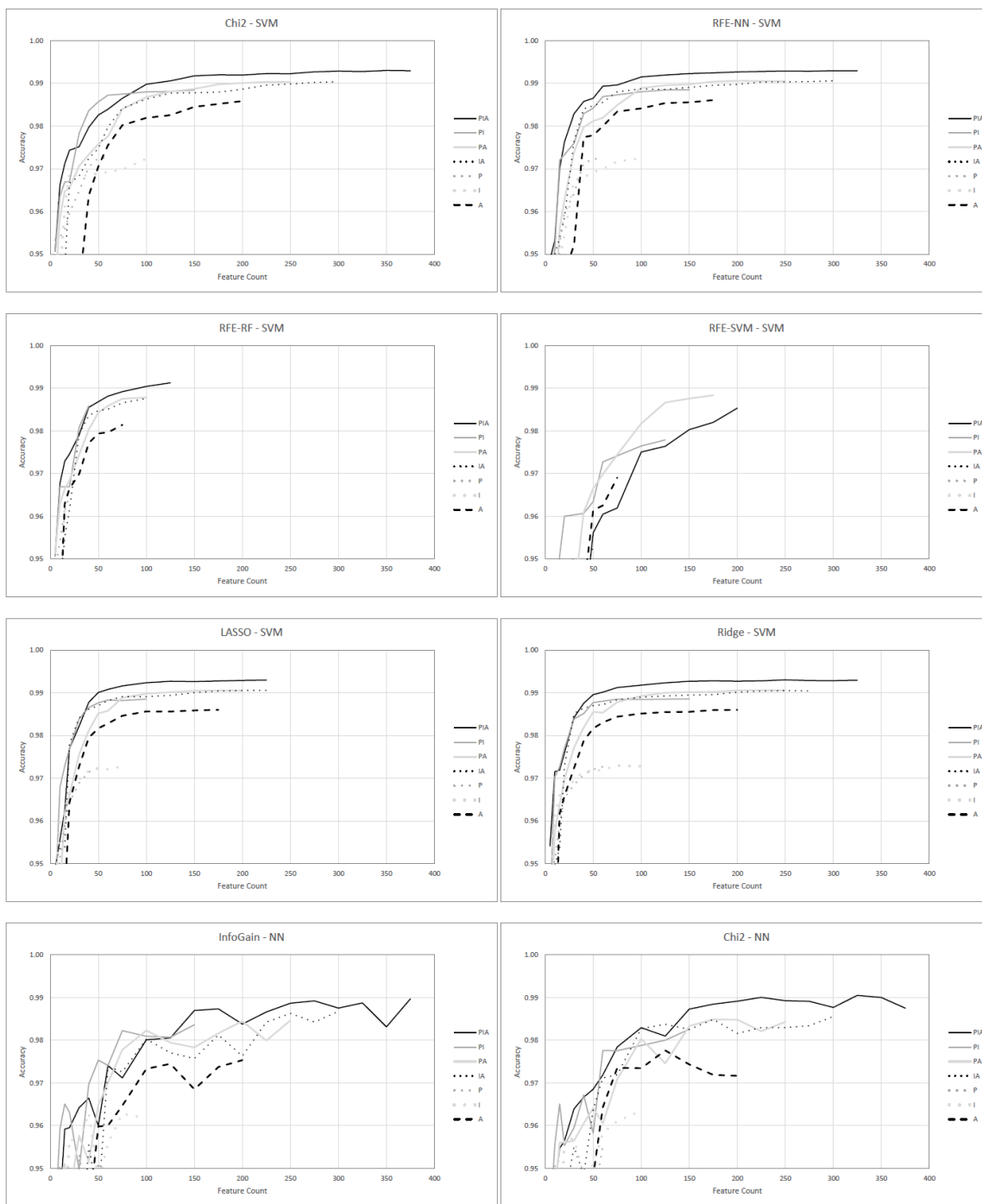
P097	READ_CALENDAR	2	2	1				1	2	3	1	1
P150	WRITE_CALENDAR	1	1	2				1	2	2	2	2
P152	WRITE_CONTACTS	1	1	2			1	1	2	3	1	1
I129	ACTION_VOICE_COMMAND	2	2	2				2	1	2		2
I160	CATEGORY_PREFERENCE	2	2	1				2	1	3	1	1
A016	android.app.LauncherActivity							4		1	4	4
A044	android.content.Entity							4		1	4	4
A097	android.media.Ringtone	3	2	1				2	1	2	1	1
A109	android.net.MailTo							4		2	3	4
A112	android.net.SSLCertificateSocketFactory							4		3	3	3
A114	android.net.Uri	4	3	3					1	2		
A115	android.net.UrlQuerySanitizer							4		3	2	4
A129	android.os.IInterface	1	2	2					4	1	2	1
A139	android.os.RemoteCallbackList							4		4	2	3
A140	android.os.RemoteException			1	4			2	4	2		
A170	android.support.v4.os.EnvironmentCompat	4	4					2		1		2
P025	BIND_DEVICE_ADMIN	1	1	1				2	1	2	2	2
P107	READ_SYNC_STATS	1	1					3	1	3	1	2
P151	WRITE_CALL_LOG	1	1					2	1	3	2	2
I008	ACTION_ATTACH_DATA	2	2	1				1	1	2	1	2
I015	ACTION_CALL_BUTTON	2	1	1				3	1	1	1	2
I064	ACTION_MEDIA_SCANNER_FINISHED	2	2	2				1	2	1	1	1
I087	ACTION_PACKAGE_RESTARTED	1	1	2				2	1	1	1	3
I100	ACTION_RUN	2	2	2				2	1		1	2
A005	android.app.admin.DeviceAdminReceiver							4		1	4	3
A036	android.content.ComponentName	1	1	4					4	1	1	
A059	android.content.pm.ProviderInfo	1	2					3			3	3
A074	android.database	1	1	4					4		1	1
A082	android.location.Gps	1	1					3			3	4
A119	android.os.Build	1	1	4					4	2		
A145	android.os.SystemClock	1	1	4					4	2		
A147	android.preference.CheckBoxPreference			3				2		1	3	3
A203	android.webkit			4			1		4	1	1	1
P016	BIND_ACCESSIBILITY_SERVICE	1	1					2	1	2	2	2
P034	BIND_REMOTEVIEWS	1	1					2	1	2	2	2
P064	DELETE_CACHE_FILES	1	1					3	1	1	2	2
I002	ACTION_ALL_APPS	2	2	2				1	1	3		
I016	ACTION_CAMERA_BUTTON	2	1	1				2	1	1	1	2
I020	ACTION_CONFIGURATION_CHANGED	1	2	1				1	2	1	2	1
I024	ACTION_DEFAULT	1	1	2				1	1	3	1	1
I035	ACTION_EXTERNAL_APPLICATIONS_UNAVAIL LABLE	1	1	2				2	2	1	1	1
I042	ACTION_INPUT_METHOD_CHANGED	2	2	1				2	1	1	1	1
I062	ACTION_MEDIA_NOFS	1	1					3	1	3	1	1
I069	ACTION_MEDIA_UNMOUNTED	1	1	2				1	2	1	2	1
I099	ACTION_REBOOT	2	1	2				1	1	2	1	1
I115	ACTION_TIME_TICK	1	1	1				1	2	3	1	1
I161	CATEGORY_SAMPLE_CODE	2	2	1				2	1	1	1	1
A054	android.content.pm.LabeledIntent	4	4					2		1		
A070	android.content.ServiceConnection			1	3				4	3		
A117	android.nfc							3		2	2	4
A124	android.os.Debug				4			1		2	2	2
A153	android.provider.BaseColumns	4	4						1	2		
A163	android.service.wallpaper.WallpaperService			3				3	1	1	2	1
P076	GLOBAL_SEARCH	1	1					3	1	2	1	1
I003	ACTION_ANSWER	2	2	1				1	1	2		1
I034	ACTION_EXTERNAL_APPLICATIONS_AVAILA BLE	1	1	2				1	2	1	1	1
I041	ACTION_HEADSET_PLUG			2				1	2	2	2	1
I047	ACTION_LOCALE_CHANGED	1	1					2	1	1	2	2
I057	ACTION_MEDIA_BAD_REMOVAL	1	1					2	1	1	2	2
I063	ACTION_MEDIA_REMOVED	1	1					2	2	2	1	1
I117	ACTION_UID_REMOVED	2	2	1				1	1	2		1
I163	CATEGORY_SELECTED_ALTERNATIVE	2	2	1				1	1	1	1	1
I164	CATEGORY_TAB	3	2	1				1	1	1		1
A002	android.annotation.SuppressLint			4					3	3		

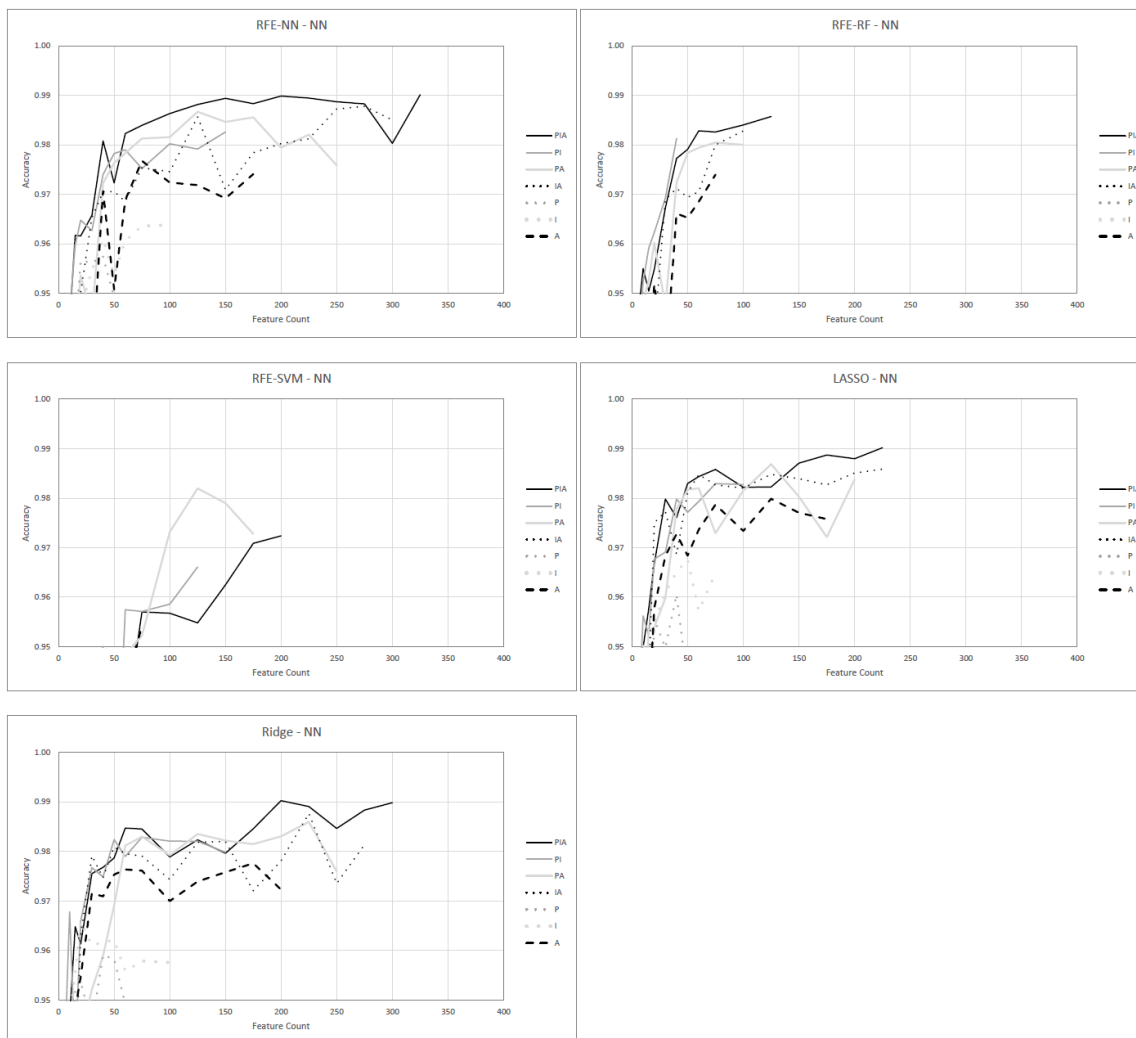
A009	android.app.AlertDialog.Builder			4				4	2		
A026	android.app.TabActivity			4			1	1	1	2	1
A057	android.content.pm.PackageStats	2	3				1		2	1	1
A105	android.net.http.SslCertificate						3		3	2	2
A126	android.os.FileObserver						4			2	4
A169	android.support.v4.media.TransportMediator						4		3		3
A188	android.util.FloatMath			4				2		2	2
A199	android.util.StateSet	1	1				2			3	3
I009	ACTION_BATTERY_CHANGED			3		1		2	1	1	1
I046	ACTION_INSTALL_PACKAGE	1	1				2		3	1	1
I056	ACTION_MANAGE_PACKAGE_STORAGE	2	2	1			1	1	1		1
I068	ACTION_MEDIA_UNMOUNTABLE	2	2	1			1	1	1		1
I070	ACTION_MY_PACKAGE_REPLACED	1	1				2	1	1	1	2
I112	ACTION_SYSTEM_TUTORIAL	2	2	2				1	2		
I119	ACTION_UMS_DISCONNECTED	2	2	2			1	1			1
I149	CATEGORY_DEVELOPMENT_PREFERENCE	2	2	1			1	1	1		1
I167	CATEGORY_UNIT_TEST	2	2	1				1	3		
A001	android.accounts			1			2	4	1		1
A012	android.app.DownloadManager	1	2				3			1	2
A069	android.content.SearchRecentSuggestionsProvider	1					1		4	1	2
A185	android.util.Config	3	1				2			1	2
P012	ACTIVITY_RECOGNITION	1	1				1	1	3		1
P013	ADD_VOICEMAIL	1	1				1	1	3		1
P041	BIND_VPN_SERVICE	1	1				1	1	3		1
P098	READ_CALL_LOG	1	1				1	1	1	1	2
P146	USE_SIP	1	1	1			1	1	2		1
I013	ACTION_BUG_REPORT	2	2	1			1	1	1		
I059	ACTION_MEDIA_CHECKING	1	1				1	1	2	1	1
I067	ACTION_MEDIA_SHARED	1	1				1	1	2	1	1
I073	ACTION_NEW_OUTGOING_CALL			2			1	2	1	1	1
I118	ACTION_UMS_CONNECTED	2	1	1			1	1		1	1
I148	CATEGORY_DESK_DOCK	1	1				1	1	2	1	1
I151	CATEGORY_FRAMEWORK_INSTRUMENTATION_TEST	2	2	1				1	2		
I165	CATEGORY_TEST	2	2	1			1	1			1
A035	android.content.ComponentCallbacks						1	3	2	1	1
A049	android.content.OperationApplicationException	3	2	1					1	1	
A076	android.hardware.display.DisplayManager	3	4							1	
A096	android.media.RemoteControlClient	3	4						1		
A098	android.media.SoundPool	1	1	4				1	1		
A142	android.os.ServiceManager	1	3				1		1	1	1
A166	android.speech.tts.TextToSpeech	3	1				1		1	1	1
A181	android.util.AndroidException	1	1				2		1	1	2
A183	android.util.AttributeSet			4				4			
A184	android.util.Base64	1	2	2				2	1		
A198	android.util.SparseIntArray	1					2		1	2	2
P031	BIND_NOTIFICATION_LISTENER_SERVICE	1	1				1	1	2		1
P037	BIND_TEXT_SERVICE	1	1					1	3		1
P094	PACKAGE_USAGE_STATS	1	1				1	1	1	1	1
I030	ACTION_DOCK_EVENT	1	1				1	1	1	1	1
A013	android.app.Instrumentation	1	1				2		1	1	1
A018	android.app.LoaderManager	1							4	1	1
A050	android.content.pm.ActivityInfo			2				3	2		
A058	android.content.pm.PermissionInfo						1		3	2	1
A065	android.content.res.ColorStateList						1	1	1	2	2
A099	android.media.ThumbnailUtils						2		3	1	1
A102	android.net.DhcpInfo	1	1				2		1	1	1
A167	android.support.v4.app						1	4	2		
I028	ACTION_DEVICE_STORAGE_OK	1	1				1	1	1		1
I145	CATEGORY_CAR_DOCK	1	1				1		1	1	1
A103	android.net.http.AndroidHttpClient						2		1	1	2
A122	android.os.CountDownTimer						2			2	2
A131	android.os.MemoryFile	1	1				1		2		1
A149	android.preference.EditTextPreference			3					3		
A174	android.support.v4.util.LogWriter						2		1	2	1
A192	android.util.LruCache						2		3	1	

Appendix E

Dataset Evaluation by Feature Selection Method and Classifier







Appendix F

Top 200 Permissions, Intents and API Calls

#	Feature Type	Feature Name
1	Permission	INSTALL_SHORTCUT
2	Permission	RECEIVE_SMS
3	Permission	SYSTEM_ALERT_WINDOW
4	Intent	ACTION_PACKAGE_ADDED
5	Intent	ACTION_SENDTO
6	Permission	READ_SMS
7	Intent	ACTION_BOOT_COMPLETED
8	API Call	android.net.Proxy
9	Intent	CATEGORY_HOME
10	API Call	android.telephony
11	Permission	READ_PHONE_STATE
12	API Call	android.app.PendingIntent
13	API Call	android.content.Intent.ShortcutIconResource
14	Permission	READ_EXTERNAL_STORAGE
15	Permission	WRITE_EXTERNAL_STORAGE
16	Permission	ACCESS_FINE_LOCATION
17	Permission	ACCESS_WIFI_STATE
18	Permission	ACCESS_NETWORK_STATE
19	API Call	android.annotation.TargetApi
20	API Call	android.app.Service
21	API Call	android.content.pm.ApplicationInfo
22	API Call	android.content.pm.PackageInfo
23	Permission	ACCESS_COARSE_LOCATION
24	Permission	KILL_BACKGROUND_PROCESSES
25	API Call	android.provider.Browser
26	Permission	ACCESS_LOCATION_EXTRA_COMMANDS
27	Permission	RECEIVE_BOOT_COMPLETED
28	Intent	ACTION_WEB_SEARCH
29	API Call	android.content.IntentFilter
30	API Call	android.content.pm.ServiceInfo
31	Permission	READ_CONTACTS
32	API Call	android.content.pm.Signature
33	API Call	android.os.IBinder
34	Permission	RECEIVE_WAP_PUSH
35	Intent	ACTION_DELETE
36	Intent	ACTION_PACKAGE_CHANGED
37	Intent	ACTION_PACKAGE_INSTALL
38	Intent	ACTION_PICK
39	API Call	android.content.ContentUris

40	API Call	android.net.wifi
41	API Call	android.os.Parcel
42	API Call	android.util.Log
43	Permission	BLUETOOTH_ADMIN
44	Permission	CALL_PHONE
45	Permission	CHANGE_WIFI_STATE
46	Intent	ACTION_PACKAGE_REMOVED
47	Intent	ACTION_PACKAGE_REPLACED
48	API Call	android.app.ActivityManager
49	API Call	android.content.BroadcastReceiver
50	API Call	android.os.Looper
51	Permission	RECEIVE_MMS
52	Permission	VIBRATE
53	Intent	ACTION_DEVICE_STORAGE_LOW
54	Intent	ACTION_DIAL
55	Intent	ACTION_SEND
56	Intent	CATEGORY_OPENABLE
57	API Call	android.util.Pair
58	Permission	EXPAND_STATUS_BAR
59	Intent	ACTION_INSERT_OR_EDIT
60	Intent	ACTION_USER_PRESENT
61	Intent	ACTION_VIEW
62	Intent	CATEGORY_BROWSABLE
63	API Call	android.app.SearchManager
64	API Call	android.content.UriMatcher
65	API Call	android.hardware.Camera
66	API Call	android.media.AudioManager
67	API Call	android.net.ConnectivityManager
68	Permission	SET_WALLPAPER_HINTS
69	Permission	WAKE_LOCK
70	Intent	ACTION_SCREEN_OFF
71	API Call	android.app.Notification
72	API Call	android.content.ActivityNotFoundException
73	API Call	android.content.IntentSender
74	API Call	android.content.res.AssetManager
75	API Call	android.os.CancellationSignal
76	API Call	android.os.Parcelable
77	API Call	android.preference.Preference
78	API Call	android.support.v4.util.SimpleArrayMap
79	Permission	BLUETOOTH
80	Permission	CHANGE_CONFIGURATION
81	Permission	CLEAR_APP_CACHE
82	Permission	INTERNET
83	Permission	MODIFY_AUDIO_SETTINGS

84	Intent	ACTION_PICK_ACTIVITY
85	Intent	ACTION_WALLPAPER_CHANGED
86	Intent	CATEGORY_INFO
87	Intent	CATEGORY_MONKEY
88	API Call	android.content.ContentValues
89	API Call	android.content.res.XmlResourceParser
90	API Call	android.location.Geocoder
91	API Call	android.util.SparseBooleanArray
92	Permission	CHANGE_NETWORK_STATE
93	Permission	RECORD_AUDIO
94	Intent	ACTION_CREATE_SHORTCUT
95	Intent	ACTION_SCREEN_ON
96	Intent	ACTION_SET_WALLPAPER
97	API Call	android.content.pm.PackageManager
98	API Call	android.media.MediaPlayer
99	API Call	android.os.Handler
100	API Call	android.os.StatFs
101	Permission	BROADCAST_STICKY
102	Permission	GET_PACKAGE_SIZE
103	Permission	WRITE_SETTINGS
104	Intent	ACTION_BATTERY_LOW
105	Intent	ACTION_BATTERY_OKAY
106	Intent	ACTION_CALL
107	Intent	ACTION_CLOSE_SYSTEM_DIALOGS
108	Intent	ACTION_DATE_CHANGED
109	Intent	CATEGORY_DEFAULT
110	API Call	android.app.AlarmManager
111	API Call	android.app.AlertDialog
112	API Call	android.content.ContentProvider
113	API Call	android.net.NetworkInfo
114	API Call	android.os.Environment
115	API Call	android.util.DisplayMetrics
116	API Call	android.util.Xml.Encoding
117	Permission	SET_WALLPAPER
118	Intent	ACTION_MEDIA_BUTTON
119	Intent	ACTION_MEDIA_MOUNTED
120	API Call	android.content.pm.ResolveInfo
121	API Call	android.content.res.Configuration
122	API Call	android.location.Address
123	API Call	android.os.Message
124	API Call	android.os.PowerManager
125	API Call	android.preference.DialogPreference
126	Permission	DISABLE_KEYGUARD
127	Permission	NFC

128	Permission	REORDER_TASKS
129	Permission	SET_ALARM
130	Permission	WRITE_SYNC_SETTINGS
131	Intent	ACTION_EDIT
132	Intent	ACTION_GET_CONTENT
133	Intent	ACTION_SEARCH
134	Intent	ACTION_SEARCH_LONG_PRESS
135	Intent	CATEGORY_EMBED
136	Intent	CATEGORY_LAUNCHER
137	API Call	android.app.IntentService
138	API Call	android.location.Criteria
139	API Call	android.location.Location
140	API Call	android.media.MediaRecorder
141	API Call	android.os.DeadObjectException
142	API Call	android.os.Process
143	API Call	android.os.Vibrator
144	API Call	android.provider.Contacts
145	API Call	android.provider.MediaStore
146	API Call	android.sax
147	API Call	android.util.Xml
148	Permission	CAMERA
149	Permission	READ_SYNC_SETTINGS
150	Intent	ACTION_MAIN
151	Intent	ACTION_MEDIA_SCANNER_SCAN_FILE
152	API Call	android.app.Application
153	API Call	android.content.AsyncQueryHandler
154	API Call	android.content.ContentResolver
155	API Call	android.content.DialogInterface
156	API Call	android.content.SharedPreferences
157	API Call	android.os.Binder
158	API Call	android.provider.Settings
159	Permission	BIND_INPUT_METHOD
160	Permission	CHANGE_WIFI_MULTICAST_STATE
161	Intent	ACTION_CHOOSER
162	Intent	ACTION_INSERT
163	Intent	ACTION_POWER_USAGE_SUMMARY
164	Intent	ACTION_PROVIDER_CHANGED
165	Intent	ACTION_SYNC
166	API Call	android.app.LocalActivityManager
167	API Call	android.content.pm.IPackageStatsObserver
168	API Call	android.content.res.AssetFileDescriptor
169	API Call	android.hardware.Sensor
170	Permission	BATTERY_STATS
171	Permission	GET_ACCOUNTS

172	Intent	ACTION_MEDIA_EJECT
173	Intent	ACTION_MEDIA_SCANNER_STARTED
174	Intent	ACTION_PACKAGE_DATA_CLEARED
175	Intent	ACTION_SEND_MULTIPLE
176	Intent	ACTION_TIMEZONE_CHANGED
177	Intent	CATEGORY_ALTERNATIVE
178	API Call	android.content.res.Resources
179	API Call	android.os.ParcelFileDescriptor
180	API Call	android.os.ResultReceiver
181	API Call	android.support.v4.util.SparseArrayCompat
182	API Call	android.util.SparseArray
183	API Call	android.util.TypedValue
184	Permission	BIND_WALLPAPER
185	Permission	PROCESS_OUTGOING_CALLS
186	Permission	READ_CALENDAR
187	Permission	WRITE_CALENDAR
188	Permission	WRITE_CONTACTS
189	Intent	ACTION_VOICE_COMMAND
190	Intent	CATEGORY_PREFERENCE
191	API Call	android.app.LauncherActivity
192	API Call	android.content.Entity
193	API Call	android.media.Ringtone
194	API Call	android.net.MailTo
195	API Call	android.net.SSLCertificateSocketFactory
196	API Call	android.net.Uri
197	API Call	android.net.UrlQuerySanitizer
198	API Call	android.os.IInterface
199	API Call	android.os.RemoteCallbackList
200	API Call	android.os.RemoteException

References

- Aafer, Y., Du, W., & Yin, H. (2013). Droidapiminer: Mining api-level features for robust malware detection in android. *International Conference on Security and Privacy in Communication Systems* (pp. 86-103). Springer.
- Abaid, Z., Kaafar, M. A., & Jha, S. (2017). Quantifying the impact of adversarial evasion attacks on machine learning based android malware classifiers. *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)* (pp. 1-10). IEEE.
- Abro, F. I. (2018). *Investigating Android permissions and intents for malware detection*. London, UK: City, University of London. Retrieved from <http://openaccess.city.ac.uk/19741/>
- Acar, Y., Backes, M., Bugiel, S., Fahl, S., McDaniel, P., & Smith, M. (2016). Sok: Lessons learned from android security research for appified software platforms. *2016 IEEE Symposium on Security and Privacy (SP)* (pp. 433-451). IEEE.
- Adebayo, O. S., & AbdulAziz, N. (2014). Android malware classification using static code analysis and apriori algorithm improved with particle swarm optimization. *2014 4th World Congress on Information and Communication Technologies (WICT 2)* (pp. 123-128). IEEE.
- Alatwi, H. A. (2016). Android malware detection using category-based machine learning classifiers. Rochester Institute of Technology.
- Allix, K., Bissyande, T. F., Jerome, Q., Klein, J., & Le, T. Y. (2016). Empirical assessment of machine learning-based malware detectors for Android. *Empirical Software Engineering*, 21(1), 183-211.
- Allix, K., Bissyande, T. F., Klein, J., & Le, T. Y. (2014). *Machine Learning-Based Malware Detection for Android Applications: History Matters!* University of Luxembourg, SnT.
- Alswaina, F., & Elleithy, K. (2018). Android Malware Permission-Based Multi-Class Classification Using Extremely Randomized Trees. *IEEE Access*, 6, 76217-76227.
- Altaher, A. (2017). An improved Android malware detection scheme based on an evolving hybrid neuro-fuzzy classifier (EHNFC) and permission-based features. *Neural Computing and Applications*, 28(12), 4147-4157.

- Altaher, A., & BaRukab, O. (2017). Android malware classification based on ANFIS with fuzzy c-means clustering using significant application permissions. *Turkish Journal of Electrical Engineering & Computer Sciences*, 25(3), 2232-2242.
- Amadeo, R. (2017). *Android 8.0 Oreo, thoroughly reviewed*. Retrieved 12 1, 2018, from arstechnica.com: <https://arstechnica.com/gadgets/2017/09/android-8-0-oreo-thoroughly-reviewed/5/#h1>
- Anderson, H. S., Kharkar, A., Filar, B., Evans, D., & Roth, P. (2018). Learning to evade static PE machine learning malware models via reinforcement learning. *08917 ArXiv Preprint ArXiv*.
- Application Fundamentals*. (2020). Retrieved 5 1, 2020, from android.com: <https://developer.android.com/guide/components/fundamentals.html>
- Ariyapala, K., Do, H. G., Anh, H. N., Ng, W. K., & Conti, M. (2016). A host and network based intrusion detection for android smartphones. *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)* (pp. 849-854). IEEE.
- Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. (2014). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. *NDSS. 14*, pp. 23-26. Internet Society.
- Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., . . . McDaniel, P. (2014). Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*. 49, pp. 259-269. ACM.
- Aswini, A., & Vinod, P. (2014). Android malware analysis using ensemble features. *International Conference on Security, Privacy, and Applied Cryptography Engineering* (pp. 303-318). Springer.
- Aung, Z., & Zaw, W. (2013). Permission-based android malware detection. *International Journal of Scientific & Technology Research*, 2(3), 228-234.
- AV-Test Malware Statistics*. (2019). Retrieved 5 30, 2019, from av-test.org: <https://www.av-test.org/en/statistics/malware/>
- Bhattacharya, A., Goswami, R. T., & Mukherjee, K. (2019). A feature selection technique based on rough set and improvised PSO algorithm (PSORS-FS) for permission based detection of Android malwares. *International Journal of Machine Learning and Cybernetics*, 10(7), 1893-1907.

- Bolon-Canedo, V., Sanchez-Marono, N., & Alonso-Betanzos, A. (2013). A review of feature selection methods on synthetic data. *Knowledge and Information Systems*, 34(3), 483-519.
- Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2), 1153-1176.
- Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. (2011). Crowdroid: behavior-based malware detection system for android. *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices* (pp. 15-26). ACM.
- Caruana, R., & Niculescu-Mizil, A. (2006). An Empirical Comparison of Supervised Learning Algorithms. *Proceedings of the 23rd International Conference on Machine Learning*.
- Castillo, C. A. (2011). Android malware past, present, and future.
- Chan, P. P., & Song, W.-K. (2014). Static detection of Android malware by using permissions and API calls. *2014 International Conference on Machine Learning and Cybernetics. 1*, pp. 82-87. IEEE.
- Chen, S., Fan, L., Chen, C., Su, T., Li, W., Liu, Y., & Xu, L. (2019). Storydroid: Automated generation of storyboard for Android apps. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)* (pp. 596-607). IEEE.
- Contagio Malware Dump*. (2020). Retrieved 12 1, 2019, from contagiodump.blogspot.com: <http://contagiodump.blogspot.com/>
- Coronado-De-Alba, L. D., Rodriguez-Mota, A., & Escamilla-Ambrosio, P. J. (2016). Feature selection and ensemble of classifiers for Android malware detection. *2016 8th IEEE Latin-American Conference on Communications (LATINCOM)* (pp. 1-6). IEEE.
- Dex to Java decompiler*. (2019). Retrieved 12 1, 2019, from github.com/skylot: <https://github.com/skylot/jadx>
- Duc, N. V., & Giang, P. T. (2018). NADM: Neural Network for Android Detection Malware. *Proceedings of the Ninth International Symposium on Information and Communication Technology* (pp. 449-455). ACM.
- Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., . . . Sheth, A. N. (2010). TaintDroid: an information-flow tracking system for realtime privacy

- monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2), 5.
- Fan, M., Liu, J., Wang, W., Li, H., Tian, Z., & Liu, T. (2017). Dapasa: detecting android piggybacked apps through sensitive subgraph analysis. *IEEE Transactions on Information Forensics and Security*, 12(8), 1772-1785.
- Fang, Z., Wang, J., Li, B., Wu, S., Zhou, Y., & Huang, H. (2019). Evading Anti-Malware Engines With Deep Reinforcement Learning. *IEEE Access*, 7, 48867-48879.
- Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., & Rajarajan, M. (2015). Android security: a survey of issues, malware penetration, and defenses. *IEEE Communications Surveys & Tutorials*, 17(2), 998-1022.
- Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G., & Furnell, S. (2017). Androdialysis: Analysis of android intent effectiveness in malware detection. *Computers & Security*, 65, 121-134.
- Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). Android permissions demystified. *Proceedings of the 18th ACM Conference on Computer and Communications Security* (pp. 627-638). ACM.
- Firdaus, A., Anuar, N. B., Karim, A., & Ab, R. M. (2018). Discovering optimal features using static analysis and a genetic search based method for Android malware detection. *Frontiers of Information Technology & Electronic Engineering*, 19(6), 712-736.
- Gandotra, E., Bansal, D., & Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security*, 2014.
- Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., & Vazquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1), 18-28.
- Ghaffari, F., Abadi, M., & Tajoddin, A. (2017). AMD-EC: Anomaly-based Android malware detection using ensemble classifiers. *2017 Iranian Conference on Electrical Engineering (ICEE)* (pp. 2247-2252). IEEE.
- Ghorbanzadeh, M., Chen, Y., Ma, Z., Clancy, T. C., & McGwier, R. (2013). A neural network approach to category validation of android applications. *2013 International Conference on Computing, Networking and Communications (ICNC)* (pp. 740-744). IEEE.

- Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017.* (2018). Retrieved 3 24, 2018, from statista.com:
<https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>
- Glodek, W., & Harang, R. (2013). Rapid permissions-based detection and analysis of mobile malware using random decision forests. *MILCOM 2013-2013 IEEE Military Communications Conference* (pp. 980-985). IEEE.
- Grajeda, C., Breitingner, F., & Baggili, I. (2017). Availability of datasets for digital forensics--And what is missing. *Digital Investigation, 22*, S94-S105.
- Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research, 3*(Mar), 1157-1182.
- Guyon, I., Aliferis, C., Cooper, G., Elisseeff, A., Pellet, J.-P., Spirtes, P., & Statnikov, A. (2008). Design and analysis of the causation and prediction challenge. *Causation and Prediction Challenge*, (pp. 1-33).
- Guyon, I., Janzing, D., & Scholkopf, B. (2010). Causality: Objectives and assessment. *Causality: Objectives and Assessment*, (pp. 1-42).
- Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning, 46*(1-3), 389-422.
- Hall, M. A. (1999). Correlation-based feature selection for machine learning. University of Waikato Hamilton.
- Hou, O. (2012). *A Look at Google Bouncer*. Retrieved 4 6, 2018, from trendmicro.com:
<https://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/>
- Huang, C.-Y., Tsai, Y.-T., & Hsu, C.-H. (2013). Performance evaluation on permission-based detection for android malware. *Advances in Intelligent Systems and Applications, 2*, 111-120.
- Idrees, F., & Rajarajan, M. (2014). Investigating the android intents and permissions for malware detection. *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)* (pp. 354-358). IEEE.

- Idrees, F., Rajarajan, M., Conti, M., Chen, T. M., & Rahulamathavan, Y. (2017). PIndroid: A novel Android malware detection system using ensemble learning methods. *Computers & Security*, *68*, 36-46.
- Intents and Intent Filters*. (2019). Retrieved 4 1, 2018, from android.com: <https://developer.android.com/guide/components/intents-filters.html>
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An Introduction to Statistical Learning*. Springer.
- Jang, J.-w., Kang, H., Woo, J., Mohaisen, A., & Kim, H. K. (2015). Andro-autopsy: Anti-malware system based on similarity matching of malware and malware creator-centric information. *Digital Investigation*, *14*, 17-35.
- Kang, H., Jang, J.-w., Mohaisen, A., & Kim, H. K. (2015). Detecting and classifying android malware using static analysis along with creator information. *International Journal of Distributed Sensor Networks*, *11*(6), 479174.
- Kononenko, I. (1994). Estimating attributes: analysis and extensions of RELIEF. *European conference on machine learning* (pp. 171-182). Springer.
- Kotsiantis, S., Zaharakis, I., & Pintelas, P. (2007). Supervised Machine Learning: A Review of Classification Techniques. *Informatica*, *31*(3), 249-268.
- Kursa, M. B., & Rudnicki, W. R. (2011). The all relevant feature selection using random forest. *5112 ArXiv Preprint ArXiv*.
- Langford, E. (2006). Quartiles in elementary statistics. *Journal of Statistics Education*, *14*(3).
- Lee, R., & Sumiya, K. (2010). Measuring geographical regularities of crowd behaviors for Twitter-based geo-social event detection. *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks*, (pp. 1-10).
- Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., & Liu, H. (2017). Feature selection: A data perspective. *ACM Computing Surveys (CSUR)*, *50*(6), 1-45.
- Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., & Ye, H. (2018). Significant Permission Identification for Machine Learning Based Android Malware Detection. *IEEE Transactions on Industrial Informatics*, *14*(7), 3216-25.

- Li, W., Ge, J., & Dai, G. (2015). Detecting malware for android platform: An svm-based approach. *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing* (pp. 464-469). IEEE.
- Lin, X., Yang, F., Zhou, L., Yin, P., Kong, H., Xing, W., . . . Xu, G. (2012). A support vector machine-recursive feature elimination feature selection method based on artificial contrast variables and mutual information. *Journal of Chromatography B*, *910*, 149-155.
- Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., & Das, K. (2000). The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, *34*(4), 579-595.
- Lison, P. (2015). An introduction to machine learning. *Language Technology Group (LTG)*, 35.
- Liu, C.-H., Zhang, Z.-J., & Wang, S.-D. (2016). An android malware detection approach using Bayesian inference. *2016 IEEE International Conference on Computer and Information Technology (CIT)* (pp. 476-483). IEEE.
- Liu, H., Dougherty, E. R., Dy, J. G., Torkkola, K., Tuv, E., Peng, H., . . . Parsons, L. (2005). Evolving feature selection. *IEEE Intelligent Systems*, *20*(6), 64-76.
- Liu, N., Yang, M., & Zhang, S. (2017). Detecting applications with malicious behavior in Android device based on GA and SVM. *2017 2nd International Conference on Electrical, Control and Automation Engineering (ECAE 2)*. Atlantis Press.
- Liu, N., Yang, M., Zhang, H., Yang, C., Zhao, Y., Gan, J., & Zhang, S. (2018). Detection of Android Applications with Malicious Behavior Based on Sparse Bayesian Learning Algorithm. *International Conference on Cloud Computing and Security* (pp. 266-275). Springer.
- Lu, Y.-F., Kuo, C.-F., Chen, H.-Y., Chen, C.-W., & Chou, S.-C. (2018). A SVM-Based Malware Detection Mechanism for Android Devices. *2018 International Conference on System Science and Engineering (ICSSE)* (pp. 1-6). IEEE.
- Machine Learning for Malware Detection*. (2019). Retrieved 11 10, 2019, from kaspersky.com: <https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf>
- Mahindru, A., & Singh, P. (2017). Dynamic permissions based Android malware detection using machine learning techniques. *Proceedings of the 10th Innovations in Software Engineering Conference* (pp. 202-210). ACM.

- Mahmood, R., Mirzaei, N., & Malek, S. (2014). Evodroid: Segmented evolutionary testing of android apps. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 599-609). ACM.
- Mariconti, E., Onwuzurike, L., Andriotis, P., De, C. E., Ross, G., & Stringhini, G. (2016). Mamadroid: Detecting android malware by building markov chains of behavioral models. *04433 ArXiv Preprint ArXiv:.*
- Martin, A., Fuentes-Hurtado, F., Naranjo, V., & Camacho, D. (2017). Evolving deep neural networks architectures for android malware classification. *2017 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1659-1666). IEEE.
- Martin, A., Menendez, H. D., & Camacho, D. (2017). MOCDroid: multi-objective evolutionary classifier for Android malware detection. *Soft Computing*, *21*(24), 7405-7415.
- Melis, M., Maiorca, D., Biggio, B., Giacinto, G., & Roli, F. (2018). Explaining Black-box Android Malware Detection. *ArXiv Preprint*.
- Moonsamy, V., Rong, J., & Liu, S. (2014). Mining permission patterns for contrasting clean and malicious android applications. *Future Generation Computer Systems*, *36*, 122-132.
- Morales-Ortega, S., Escamilla-Ambrosio, P. J., Rodriguez-Mota, A., & Coronado-De-Alba, L. D. (2016). Native malware detection in smartphones with android os using static analysis, feature selection and ensemble classifiers. *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)* (pp. 1-8). IEEE.
- Napoleon, D., & Pavalakodi, S. (2011). A new method for dimensionality reduction using k-means clustering algorithm for high dimensional data set. *International Journal of Computer Applications*, *13*(7), 41-46.
- Nauman, M., Tanveer, T. A., Khan, S., & Syed, T. A. (2018). Deep neural architectures for large scale android malware analysis. *Cluster Computing*, *21*(1), 569-588.
- Naway, A., & Li, Y. (2018). Using Deep Neural Network for Android Malware Detection. *International Journal of Advanced Studies in Computers, Science and Engineering*, *7*(12), 9-18.
- Nezhadkamali, M., Soltani, S., & Seno, S. A. (2017). Android malware detection based on overlapping of static features. *2017 7th International Conference on Computer and Knowledge Engineering (ICCKE)* (pp. 319-325). IEEE.

- Nix, R., & Zhang, J. (2017). Classification of Android apps and malware using deep neural networks. *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 1871-1878). IEEE.
- Number of Android applications.* (2020). Retrieved 5 1, 2020, from appbrain.com: <https://www.appbrain.com/stats/number-of-android-apps>
- Paja, W., & Pancerz, K. (2017). Feature selection methods applied to severe brain damages data. *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)* (pp. 199-202). IEEE.
- Papadopoulos, H., Georgiou, N., Eliades, C., & Konstantinidis, A. (2018). Android malware detection with unbiased confidence guarantees. *Neurocomputing*, 280, 3-12.
- Park, M., Seo, J., Han, J., Oh, H., & Lee, K. (2018). Situational Awareness Framework for Threat Intelligence Measurement of Android Malware. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 9(3), 25-38.
- Pauck, F., Bodden, E., & Wehrheim, H. (2018). Do android taint analysis tools keep their promises? Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, (pp. 331-341).
- Peiravian, N., & Zhu, X. (2013). Machine learning for android malware detection using permission and api calls. *2013 IEEE 25th International Conference on Tools With Artificial Intelligence* (pp. 300-305). IEEE.
- Permissions Overview.* (2019). Retrieved 4 1, 2018, from android.com: <https://developer.android.com/guide/topics/permissions/overview.html#permission-groups>
- Peterson, L. E., & Coleman, M. A. (2005). Comparison of gene identification based on artificial neural network pre-processing with k-means cluster and principal component analysis. *International Workshop on Fuzzy Logic and Applications* (pp. 267-276). Springer.
- Qiao, M., Sung, A. H., & Liu, Q. (2016). Merging permission and API features for Android malware detection. *2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)* (pp. 566-571). IEEE.

- Raphael, J. (2017). *The big secret behind Google Play Protect on Android*. Retrieved 12 1, 2018, from computerworld.com: <https://www.computerworld.com/article/3210587/android/google-play-protect-android.html>
- Rashidi, B., Fung, C., & Bertino, E. (2017). Android malicious application detection using support vector machine and active learning. *2017 13th International Conference on Network and Service Management (CNSM)* (pp. 1-9). IEEE.
- Reyhani, H. M., Shin, D., Lee, M., Cho, S.-J., & Hwang, C. (2018). AndroClass: An Effective Method to Classify Android Applications by Applying Deep Neural Networks to Comprehensive Features. *Wireless Communications and Mobile Computing, 2018*.
- Rousseeuw, P. J., & Hubert, M. (2011). Robust statistics for outlier detection. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 1*(1), 73-79.
- Rovelli, P., & Vigfusson, Y. (2014). Pmds: Permission-based malware detection system. *International Conference on Information Systems Security* (pp. 338-357). Springer.
- Sahs, J., & Khan, L. (2012). A machine learning approach to android malware detection. *2012 European Intelligence and Security Informatics Conference* (pp. 141-147). IEEE.
- Samani, R., & Davis, G. (2019). *McAfee Mobile Threat Report*. Retrieved 7 20, 2019, from mcafee.com: <https://www.mcafee.com/enterprise/en-us/assets/reports>
- Scarsella, A., Reith, R., Chau, M., & Shirer, M. (2018). *With Expectations of a Positive Second Half of 2018 and Beyond, Smartphone Volumes Poised to Return to Growth, According to IDC*. Retrieved 12 1, 2018, from idc.com: <https://www.idc.com/getdoc.jsp?containerId=prUS44240118>
- scikit-learn - Machine Learning in Python*. (2010). Retrieved 4 9, 2019, from scikit-learn.org: <https://scikit-learn.org/>
- Seo, S.-H., Gupta, A., Sallam, A. M., Bertino, E., & Yim, K. (2014). Detecting mobile malware threats to homeland security through static analysis. *Journal of Network and Computer Applications, 38*, 43-53.
- Shahriar, H., Islam, M., & Clincy, V. (2017). Android malware detection using permission analysis. *SoutheastCon* (pp. 1-6). IEEE.

- Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.
- Shang, F., Li, Y., Deng, X., & He, D. (2018). Android malware detection method based on naive Bayes and permission correlation algorithm. *Cluster Computing*, 21(1), 966-66.
- Sharma, A., & Dash, S. K. (2014). Mining API calls and permissions for Android malware detection. *International Conference on Cryptology and Network Security* (pp. 191-205). Springer.
- Shelke, C. J. (2017). Permission based malware detection by using k means algorithm in Android OS. *Asian Journal of Computer Science Engineering (AJCSE)*, 2(2).
- Shih, H.-C., & Liu, E.-R. (2016). New quartile-based region merging algorithm for unsupervised image segmentation using color-alone feature. *Information Sciences*, 342, 24-36.
- Smutz, C., & Stavrou, A. (2016). When a Tree Falls: Using Diversity in Ensemble Classifiers to Identify Evasion in Malware Detectors. *NDSS*. Internet Society.
- Su, X., Zhang, D., Li, W., & Zhao, K. (2016). A deep learning approach to android malware feature learning and detection. *2016 IEEE Trustcom/BigDataSE/ISPA* (pp. 244-251). IEEE.
- Sufatrio, M., Tan, D. J., Chua, T.-W., & Thing, V. L. (2015). Securing android: a survey, taxonomy, and challenges. *ACM Computing Surveys (CSUR)*, 47(4), 58.
- Sun, L., Li, Z., Yan, Q., Srisa-an, W., & Pan, Y. (2016). SigPID: significant permission identification for android malware detection. *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)* (pp. 1-8). IEEE.
- Sung, A. H., & Mukkamala, S. (2003). Identifying important features for intrusion detection using support vector machines and neural networks. *2003 Symposium on Applications and the Internet* (pp. 209-216). IEEE.
- Talal, M., Zaidan, A., Zaidan, B., Albahri, O., Alsalem, M., Albahri, A., . . . Alaa, M. (2019). Comprehensive review and analysis of anti-malware apps for smartphones. *Telecommunication Systems*, 72(2), 285-337.
- Tam, K., Feizollah, A., Anuar, N. B., Salleh, R., & Cavallaro, L. (2017). The evolution of android malware and android analysis techniques. *ACM Computing Surveys (CSUR)*, 49(4), 76.

- The best antivirus software for Android.* (2018). Retrieved 12 1, 2018, from av-test.org: <https://www.av-test.org/en/antivirus/mobile-devices/>
- Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1), 267-288.
- Tsang, C.-H., Kwong, S., & Wang, H. (2007). Genetic-fuzzy rule mining approach and evaluation of feature selection techniques for anomaly intrusion detection. *Pattern Recognition*, 40(9), 2373-2391.
- Tukey, J. W. (1977). *Exploratory data analysis* (Vol. 2). Reading, Mass.
- Tuv, E., Borisov, A., & Torkkola, K. (2006). Feature selection using ensemble based ranking against artificial contrasts. *The 2006 IEEE International Joint Conference on Neural Network Proceedings* (pp. 2181-2186). IEEE.
- Tuv, E., Borisov, A., Runger, G., & Torkkola, K. (2009). Feature selection with ensembles, artificial variables, and redundancy elimination. *The Journal of Machine Learning Research*, 10, 1341-1366.
- Ustebay, S., Turgut, Z., & Aydin, M. A. (2018). Intrusion detection system with recursive feature elimination by using random forest and deep learning classifier. *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)* (pp. 71-76). IEEE.
- Verma, S., & Muttoo, S. (2016). An Android Malware Detection Framework-based on Permissions and Intents. *Defence Science Journal*, 66(6), 618-623.
- VirusShare. (2020). Retrieved 12 1, 2019, from virusshare.com: <https://virusshare.com>
- Wang, W., Li, Y., Wang, X., Liu, J., & Zhang, X. (2018). Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers. *Future Generation Computer Systems*, 78, 987-994.
- Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., & Zhang, X. (2014). Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, 9(11), 1869-1882.
- Wang, W., Zhao, M., & Wang, J. (2018). Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. *Journal of Ambient Intelligence and Humanized Computing*, 10(8), 3035-3043.

- Wang, X., Wang, J., & Zhu, X. (2016). A Static Android Mal-ware Detection Based on Actual Used Permissions Combination and API Calls. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 10(9), 1547-1554.
- Weka 3: Data Mining Software in Java*. (2018). Retrieved 12 1, 2018, from cs.waikato.ac.nz: <https://www.cs.waikato.ac.nz/ml/weka/>
- Welcome to the definitive Android Central take on the history of Google's OS*. (2015). Retrieved 4 1, 2018, from androidcentral.com: <https://www.androidcentral.com/android-history>
- Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., & Wu, K.-P. (2012). Droidmat: Android malware detection through manifest and api calls tracing. *2012 Seventh Asia Joint Conference on Information Security (Asia JCIS)* (pp. 62-69). IEEE.
- Xu, K., Li, Y., Deng, R. H., & Chen, K. (2018). DeepRefiner: Multi-layer Android Malware Detection System Applying Deep Neural Networks. *2018 IEEE European Symposium on Security and Privacy (EuroS&P)* (pp. 473-487). IEEE.
- Xu, W., Zhang, F., & Zhu, S. (2013). Permlyzer: Analyzing permission usage in android applications. *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)* (pp. 400-410). IEEE.
- Yang, M., Wang, S., Ling, Z., Liu, Y., & Ni, Z. (2017). Detection of malicious behavior in android apps through API calls and permission uses analysis. *Concurrency and Computation: Practice and Experience*, 29(19), e4172.
- Yerima, S. Y., Sezer, S., & Muttik, I. (2015). High accuracy android malware detection using ensemble learning. *IET Information Security*, 9(6), 313-320.
- Yerima, S. Y., Sezer, S., McWilliams, G., & Muttik, I. (2013). A new android malware detection approach using bayesian classification. *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)* (pp. 121-128). IEEE.
- Zhang, Y., Yang, Y., & Wang, X. (2018). A Novel Android Malware Detection Approach Based on Convolutional Neural Network. *Proceedings of the 2nd International Conference on Cryptography, Security and Privacy* (pp. 144-149). ACM.

- Zhao, M., Ge, F., Zhang, T., & Yuan, Z. (2011). Antimaldroid: An efficient svm-based malware detection framework for android. *International Conference on Information Computing and Applications* (pp. 158-166). Springer.
- Zhao, M., Zhang, T., Ge, F., & Yuan, Z. (2012). RobotDroid: a lightweight malware detection framework on smartphones. *Journal of Networks*, 7(4), 715.
- Zhao, X., Fang, J., & Wang, X. (2014). Android malware detection based on permissions. IET.
- Zhou, Y., & Jiang, X. (2012). Dissecting android malware: Characterization and evolution. *2012 IEEE Symposium on Security and Privacy* (pp. 95-109). IEEE. Retrieved from <http://malgenomeproject.org>
- Zhou, Y., Wang, Z., Zhou, W., & Jiang, X. (2012). Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. *NDSS*. 25, pp. 50-52. Internet Society.
- Zhu, H.-J., Jiang, T.-H., Ma, B., You, Z.-H., Shi, W.-L., & Cheng, L. (2018). HEMD: a highly efficient random forest-based malware detection framework for Android. *Neural Computing and Applications*, 30(11), 3353-61.
- Zhu, H.-J., You, Z.-H., Zhu, Z.-X., Shi, W.-L., Chen, X., & Cheng, L. (2018). DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing*, 272, 638-646.