Universiteit Leiden
The Netherlands

Opleiding Informatica & Economie

Classification of Android applications using the Koodous data set

Arian Doek

Supervisors:
Olga Gadyatskaya &
Nathan Daniel Schiele

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

July 26, 2021

Abstract

The use of mobile smartphones have risen all over the world, and are increasingly used to store personal data. Due to the popularity, mobile malware has also been on the rise, and it is thus important that security researchers propose new techniques to detect mobile malware. In this thesis, we investigate how the features of an Android application can be used to predict the maliciousness of the application.

The application features are sourced from Koodous, a collaborative platform aimed at research towards Android malware. The maliciousness of an applications is determined by the rating it was given by users on Koodous.

First, the Koodous data set is investigated on how the features within it differ between applications. These features are later used in two machine learning models, to determine if it is possible to correctly predict whether an application will get a positive or negative rating.

Under the best circumstances, the models reach an accuracy of 98.8% and an F1 score of 98.8%. We can conclude from these results that it is possible to predict the ratings given to applications on Koodous. Under different circumstances (limited feature set or a smaller balanced data set), the accuracy is between 61.1% and 98.5%. Classifying using a neural network model always performed equal to or better than classifying with a random forest model.

The results from this thesis can be used to further improve the classification of malicious applications before the code of an application is analyzed for malicious behavior. Furthermore, classification using these features could also point towards malicious applications that have not been identified as such.

Contents

1 Introduction

The use of smartphones has been rising all around the world, with a report estimating that 70% of the global population will have access to mobile services by 2025 [GSM21]. Mobile devices have also increasingly been used to store important information, with half of American respondents storing personal data on their smartphones [Bit17].

Android is the most popular mobile operating system, with a market share of around 73% [O'D21]. Largely because of its size, Google's operating system is a big target for malicious actors. According to a report from F-Secure, more than 99% of all mobile malware are designed for Android, and around 8% of all malware detected across all operating systems target the Android operating system [F-S17].

When applications are stored on an app store that comes pre-installed on Android devices (like Google Play or Samsung's Galaxy Store), there are additional checks if the apps are in any way malicious of nature. The Android operating system allows users to install apps from outside pre-installed app stores, and the applications outside these app stores have not necessarily been checked for malicious behavior. The aim is therefore to help improve the classification of Android applications, by investigating if it is possible to discriminate between malicious and benign apps using the application its features.

Our source for the application features is Koodous, a platform for collaborative research into malware on Android, where users can rate applications positively (indicating a benign app) or negatively (indicating a malicious app). Koodous itself could also benefit from this information, prioritizing the (manual or automatic) examination for these applications. More information about Koodous can be found in section 2.1.

1.1 Research questions

For this research, we want to find out how the features of an application can be used to identify malicious apps. In this paper, we aim to answer the following research questions:

Research questions

What are the app features that can be collected from the crowdsourcing mobile malware platform Koodous?

In what way can these features be used to predict if an application will have a positive or negative rating on Koodous?

1.2 Thesis overview

This chapter includes the introduction to the thesis. Section 2 explains the background of this topic. Section 4 talks about some of the features available in the data set that will be used. Section 5 gives more information about the model that will be used for classification. Section 7 gives a conclusion to this thesis. Section 7.1 gives possible topics for further research.

2 Background

2.1 Koodous

Koodous is a collaborative platform for Android malware research.[1] Platform users can upload new applications to the website, resulting in an extensive application data set. Koodous lists a set of features for each platform hosted on the platform (see 2.2).

Users can rate applications either positively ($+1$) or negatively (-1), with the total rating of an application being the cumulative of all ratings. Users can also leave comments to share insights about a specific application.

It is unlikely that the applications stored on Koodous have the same proportion of benign and malicious apps compared to other Android markets, like Google Play. Koodous defines itself as a platform for malware research, where anybody can submit applications to receive a community verdict whether the app is malicious or not. Most Android users install applications from secured app stores on their smartphones [KCB20], so a user is interested in the information on Koodous when they have doubts about the safety of an application that they got from an unreliable source.

2.1.1 Koodous API

Koodous offers an API (Application Programming Interface), which makes it easy to retrieve information about APKs (the file type for Android applications) stored on Koodous[2]. For this thesis, this API was used to retrieve all the features stored of applications.

Table 1 lists the different application features available in the data set, and their respective data types and descriptions.

[1] https://koodous.com/
[2] https://docs.koodous.com/rest-api/getting-started/

Application feature	**Data type**	**Description**	**Source**
Creation timestamp	Integer	Timestamp when the application was uploaded to Koodous (Epoch time).	Assigned by Koodous.
Rating	Integer	The rating given to an application.	Assigned by users on Koodous.
App icon	Image	Icon shown on the installed device.	Property of the application.
MD5	Hash key	Key resulting from hashing with MD5	Property of the application.
SHA1	Hash key	Key resulting from hashing with SHA-1	Property of the application.
SHA256	Hash key	Key resulting from hashing with SHA-256	Property of the application.
App name	Text	Name shown on the installed device.	Property of the application.
Package name	Text	Package name used for the application.	Property of the application.
Organization name	Text	Organization name stored in the certificate of the application.	Property of the application.
Version	Text	This is the version shown to the user, and has no naming convention.	Property of the application.
App size	Integer	Size of the application in bits.	Property of the application.
`analyzed`-flag	Boolean	Shows if Koodous has created an analysis report of the application.	Assigned by Koodous.
`trusted`-flag	Boolean	Shows if the application has been whitelisted by Koodous.	Assigned by Koodous.
`detected`-flag	Boolean	Shows if the application has been detected to be malicious by Koodous.	Assigned by Koodous.
`corrupted`-flag	Boolean	Shows if the APK will be able to be executed correctly.	Assigned by Koodous.
`on_devices`-flag	Boolean	Shows if the application is (or was) installed on some device.	Assigned by Koodous.

Table 1: All features available in the Koodous API

2.2 Application features

Application features describe a certain characteristic of an application. Some of the information in table 1 are properties of the application itself, and some are assigned by Koodous (e.g. the rating or the `analyzed`-flag). The last column in each row indicates this.

Applications (or their APK files) do not have to be downloaded to get their application features using the Koodous API. Koodous stores the application features separately from the APKs, allowing us to only retrieve this information using the API.

The rating given to an application is assigned by users, and we will use it as a measure of potential maliciousness of an app (the lower the rating, the more malicious the app). Users can use any source of information they want to give a rating. They could use other online information sources or platforms, or download the application and analyze it locally.

2.3 Detailed information about application features

2.3.1 Package name

A package name is used to uniquely identify an app to the Android system. It is also used to uniquely identify an app in the Google Play Store.[3]

Formally, the identifier used to identify an application outside of the code is called an application ID. However, since most research uses the 'package name' term, this term will also be used in this thesis. The data set from Koodous used in this research also refers to the application IDs as package names.

Trying to install an app with the same package name on an Android device, will result in the system attempting to update the older app. This will fail if the developer of the already-installed app differs from the new app. The package name will need to stay the same between different versions of the same app, or else the app will not be able to be updated[4].

The first official programming language for creating apps for Android was Java. The package names used in Android apps therefore use the same naming convention as Java uses in naming their packages. The Java documentation advises companies to use the reversed domain name as a package name, so the top-level domain (`com`, `ru`, `org`) is at the front. This way, the Java package name gets more specific the more segments the name has, with the most specific function name at the end[5].

The Java package naming convention is most often also used for the package name. Applications do not necessarily need to use this naming convention, but there are some more restrictive rules Android applications are required to follow in their package names. First, the package name needs to consist of two or more segments, with one or more dots in between. Second, each segment needs to start with a letter. Third, all characters can only be letters, numbers, underscores or dots (indicating a segment)[6].

We will refer to the top-level domains as TLDs. We will investigate whether the TLD at the beginning of a package name can help to discriminate between apps rated positively and apps rated negatively on Koodous.

2.3.2 Organization name

The organization name of an application is retrieved from within the certificate each application is signed with. The information in this certificate is not necessarily complete, and is also not necessarily truthful. This results in some applications falsely claiming they are from Google, or applications that do not give any name for the organization at all. App stores can have additional requirements, requiring developers to have a certificate with genuine information.

[3] https://developer.android.com/studio/build/application-id

[4] https://developer.android.com/studio/build/application-id

[5] https://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html

[6] https://developer.android.com/studio/build/application-id.html

We will investigate whether the organization name of an application to investigate if it can help to discriminate between apps that have a positive rating and apps that have a negative rating on Koodous.

2.3.3 Version

The versions of the apps is not used in this thesis. This is mainly due to the variation between different app versions, since there is not any specified way these version should be given. The Android documentation further clarifies that the version name “has no purpose other than to be displayed to users.“[7]

2.3.4 App name

The application names are also unused in this thesis. Similar to the use of the version, the name of an application has little structure and there are too many possible interpretations and formats.

[7] https://developer.android.com/studio/publish/versioning

3 Related work

In this thesis, we focus on classifying whether an Android application has a positive or negative rating on Koodous. A positive rating (including zero) indicates a benign app, and a negative rating indicates a malicious app. In the following paragraphs, we look at papers that use different static classification techniques to classify Android applications into malicious or benign apps. We will also look at some studies that have used the same Koodous data set.

A paper by Arp et al. [ASH+14] uses a wide assortment of application features to detect if an application is likely to be malicious, ranging from application permissions to hardware components that are used in the app. The goal of the paper is to create a static analysis method that is able to correctly and efficiently classify applications as malicious or benign, and for it to explain its detection results. This method detects 94% of malware samples with a false positive rate of 1%. The authors used the MobileSandbox data set, and have published a version of this data set with the extracted application features.

A paper by Wang et al. [WCY+20] takes a more specified feature set, and only uses the URLs that are visited by the applications. The URLs are segmented and fed into a multi-view neural network. While the amount of features is limited, the model outperforms most virus scanners on a 'wild' data set (a data set with new applications collected from Android app markets)

Another application of static analysis is the use of requested permission names in Android applications, in a paper by Milosevic and Dehghantanha [MDC17]. Malicious applications tend to use certain permission to access files that a normal application would not need to access. This approach reached an accuracy of 89%, by using a combination ('ensemble learning') of support vector machines, logistic regression and random forest.

In a paper by Zhu and Dimitras [ZD16], a new system called 'FeatureSmith' is proposed for generating application features from scientific papers. Sentences from these documents containing the names from malware families are captured and the described behavior is extracted. Using these behavior patterns to see whether an app has actual malicious behavior achieves a true positive rate of 92.5%, and a false positive rate of 1%. This system can speed up the evaluation process of an application's code, by prioritizing the most-common indicators of malicious code.

A paper by Chakraborty, Pierazzi and Subrahmanian [CPS17] uses the Drebin data set mentioned earlier to create a classification and clustering algorithm called EC2. The algorithm aims to identify Android applications, and cluster them into families of known and unknown malware families. This research uses the Koodous data set to evaluate the results of the algorithm.

Lastly, a paper by Aktas and Sen [AS18] also makes use of the Koodous data set. The applications from Koodous (in combination with apps from APKPure) are used to create a new mobile malware data set focusing on update attacks, called UpDroid. Different familial classification techniques were utilized to correctly classify the malware family for each application.

The research on Android malware detection is very large, and it is therefore not possible to mention all the papers that contributed to this area. A literature review by Pan et al. [PGFF20] focuses on malware detection using static analysis, and notes that the most commonly used features in studies are permissions, API calls and intents, none of which are used in this thesis. Another survey by Odusami et al. [OAAM+18] also discusses Android malware detection, but analyzes dynamic and machine learning approaches too. A survey by Tam et al. [TFA+17] focuses more on tactics used by Android malware to avoid detection and analysis.

Finally, the topic of this thesis slightly differs from detecting malware, as we will use application features from the Koodous data set to predict the app rating given by users on Koodous. We make the assumption that the app rating has a strong correlation with the maliciousness of an application, but it is still a different metric. We will thus use the rating as a 'proxy variable' to the actual maliciousness of an application. To the best of our knowledge, no other researchers have looked at this specific topic.

4 Data exploration

A total of 54,819,127 apps are used in this research, approximately 69.9% of the total Koodous data set. These applications range in submission date between August 2017 and February 2021. As of writing, 24.8% of the applications on Koodous is older than this, while 5.1% is newer. The features of the applications were collected between February 2021 and June 2021.

The mean rating for all applications in the data set is `-0.612`, with a standard deviation of `1.054`. The mean rating for all applications is slightly negative. This indicates that users on the platform tend to vote negatively more often than positively.

The median rating is `0`. This makes sense, because an application starts without any ratings and thus begins at a cumulative rating of zero.

4.1 Package name

Table 2 shows the ten most common package names in the data set. The table is sorted with the most common package name at the top.

	Package name	**Occurrences (% of total)**	**Mean rating (st. dev.)**
1.	`ch.nth.android.contentabo_101_sim_univ`	550858 (1.005%)	-2.163 (0.689)
2.	`com.baidu.haokan`	230888 (0.421%)	-0.039 (0.281)
3.	`cm.aptoide.pt`	213740 (0.390%)	-0.128 (0.512)
4.	`com.tencent.mobileqq`	83311 (0.152%)	-0.279 (0.724)
5.	`pob.xyz`	81388 (0.148%)	-1.823 (1.195)
6.	`com.anaconda.brave`	66292 (0.121%)	-1.989 (0.385)
7.	`com.lushi.zhuanbao`	63314 (0.115%)	-2.252 (1.291)
8.	`com.android.systemui`	62120 (0.113%)	-0.008 (0.130)
9.	`com.android.xq.noiconads`	59786 (0.109%)	-1.946 (0.510)
10.	`com.fanhua.box`	58229 (0.106%)	-0.814 (1.020)

Table 2: Ten most common package names

Table 3 lists the amount of package names with and without using top-level domains (TLDs) at its beginnings.

	Occurrences (% of total)	**Mean rating (st. dev.)**
TLDs	49,203,841 (89.8%)	-0.580 (1.038)
No TLDs	5,615,286 (10.2%)	-0.891 (1.151)
Total	54,819,127 (100.0%)	-0.635 (1.053)

Table 3: Use of TLDs at the start of package names

Tables 4 and 5 show the ten most common package names for package names starting with TLDs and non-TLDs respectively, with the amount of occurrences and mean ratings for each package name beginning.

	Package name beginning	Occurrences (% of total)	Mean rating (st. dev.)
1.	`com`	17,600,697 (32.107%)	-0.577 (1.038)
2.	`org`	280,684 (0.512%)	-0.523 (1.007)
3.	`net`	244,205 (0.445%)	-0.655 (1.065)
4.	`cn`	158,174 (0.289%)	-0.814 (1.130)
5.	`de`	139,164 (0.254%)	-0.199 (0.685)
6.	`kr`	127,915 (0.233%)	-0.047 (0.323)
7.	`br`	118,956 (0.217%)	-0.340 (0.863)
8.	`jp`	115,311 (0.210%)	-0.328 (0.845)
9.	`io`	113,934 (0.208%)	-0.245 (0.700)
10.	`bs`	109,375 (0.200%)	-0.002 (0.060)

Table 4: Ten most common TLDs used as package name beginnings

	Package name beginning	Occurrences (% of total)	Mean rating (st. dev.)
1.	`appinventor`	219,620 (0.401%)	-0.654 (0.967)
2.	`air`	121,086 (0.221%)	-0.640 (1.094)
3.	`trustgo`	55,524 (0.101%)	-2.346 (0.942)
4.	`aire`	51,945 (0.095%)	0.000 (0.010)
5.	`theme`	34,233 (0.062%)	-0.113 (0.474)
6.	`aimoxiu`	32,289 (0.059%)	-0.360 (0.885)
7.	`minuhome`	31,940 (0.058%)	0.000 (0.000)
8.	`wxbit`	29,566 (0.054%)	-0.022 (0.218)
9.	`seC`	26,927 (0.049%)	-2.206 (0.636)
10.	`ThemeZilla`	26,082 (0.048%)	0.000 (0.000)

Table 5: Ten most common non-TLDs used as package name beginnings

While the package names shown in table 2 represent only a small part of the whole data set (only a few percent), it shows that there exists some difference in mean ratings between different package names. These differences are to be expected, because a specific package name tends to refer to one specific application. For instance, a paper from Li et al. (2017) uses the package name to identify applications that have been used for 'piggybacking' ("unpack a benign, preferably popular, app and then graft some malicious code on it before repackaging it and distributing it for free") [LLB+17].

Package names starting with a valid top-level domain (TLD) occur more often (around 89.0% of the total data set) than package names not starting with a TLD. Package names starting with a TLD also have a higher average rating (`-0.580` compared to `-0.891`).

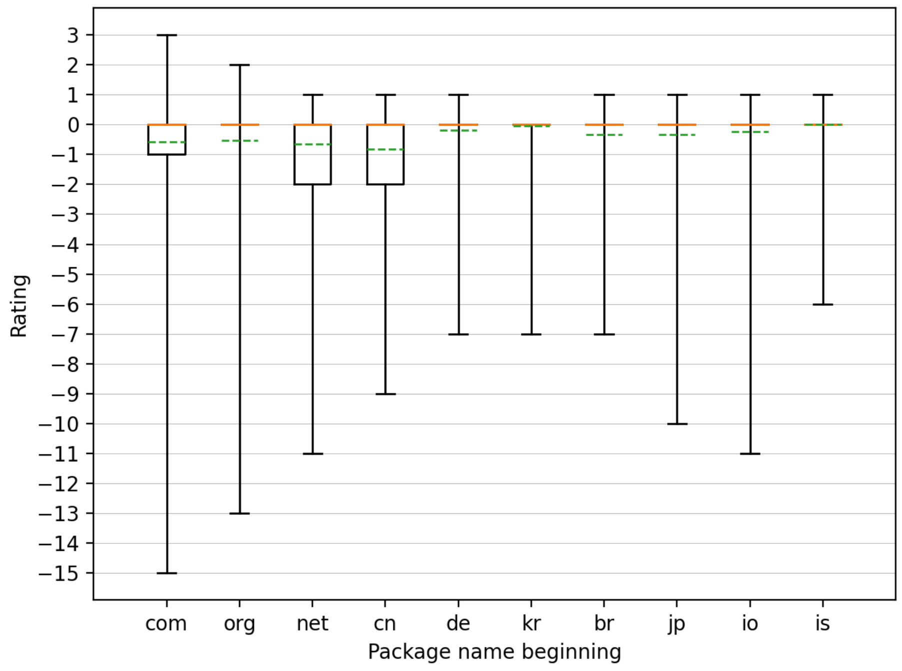

Figure 1: Box plots showing rating distribution of package name beginnings (from table 4). The whiskers indicate the 0th and 100th percentile, while the box shows the 25th, 50th (orange median line) and 75th percentile. The green horizontal line is the mean rating for a category. Due to the values being integers, some elements of the box plots are likely to overlap.

Furthermore, some top-level domains indicate an application with a lower mean rating. As we can see in table 4, the two lowest rated TLDs out of the ten most common are `cn` and `net`. These TLDs have a mean rating of respectively `-0.869` and `-0.655`.

There are also some TLDs with a higher rating, namely `bs` and `de`. These TLDs have a mean rating of respectively `-0.002` and `-0.239`. The TLD `.bs` has a very low standard deviation, while the TLD `.de` has a higher standard deviation.

In a paper from Messabi et al. [MAY+18], a list was created with suspicious top-level domains. Only one of the TLDs mentioned in that paper is present in our ten most common TLDs, namely `cn`. Limited to the ten most common TLDs used as package name beginnings in table 4, applications with this package name beginning have the lowest mean rating.

There is the possibility of geograhical biases in the data set, based on the country the top-level domains refer to. For example, the main Koodous website is only available in the English languge, which could negatively bias applications from countries without a large enough population that speak English. More research is needed using other data sets to get more insight about the presence of (geographical) bias.

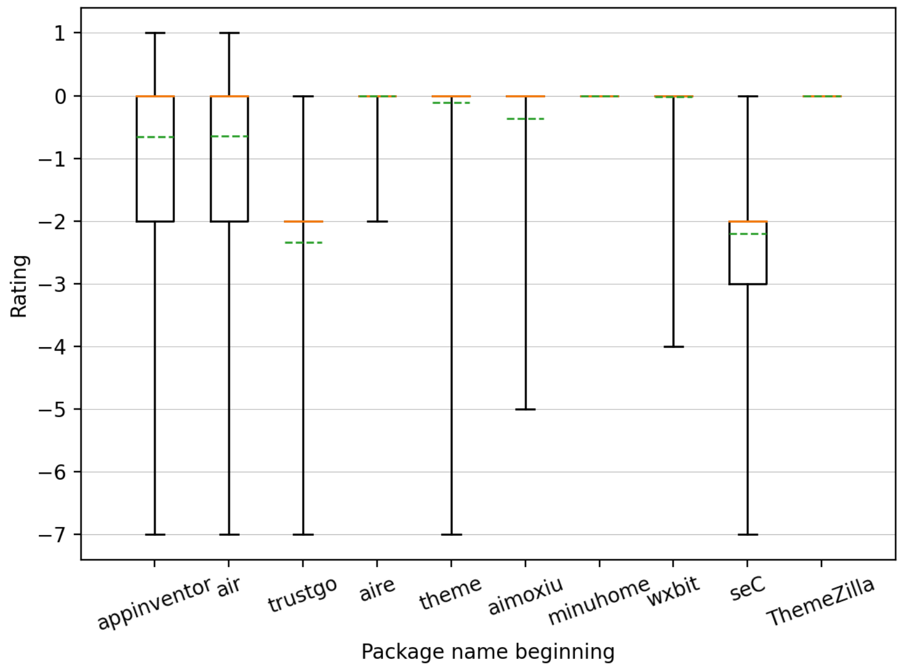

Figure 2: Box plots showing rating distribution of package name beginnings (from table 5).

Three of the ten most common non-TLDs used at the beginning of package names all have a mean rating and standard deviation at (or very close to) zero. A manual check on the website of Koodous (where the amount of users who have voted on the rating of an application is shown) confirms that this is most of the applications do not have any votes. This is probably because these non-TLDs refer to a specific type of application (e.g. `theme` and `ThemeZilla` referring to device themes), with some types of application being more accessible to users than other. Thus, no conclusions can necessarily be drawn based on these package name begins, because the applications they belong to have not been rated enough.

4.2 App size

The average size of applications in the data set is 12.390 Mb, with a standard deviation of 19.149 Mb.

Table 6 shows the amount of applications in different size ranges, and their mean ratings in that range.

App size	Occurrences (%)	Mean rating (st. dev)
0 - 2.5 Mb	18,956,457 (34.580%)	-0.848 (1.174)
2.5 - 5 Mb	9,498,510 (17.327%)	-0.888 (1.151)
5 - 10 Mb	8,789,243 (16.033%)	-0.422 (0.906)
10 - 15 Mb	4,896,989 (8.933%)	-0.454 (0.967)
15 - 20 Mb	3,248,423 (5.926%)	-0.333 (0.829)
20 - 25 Mb	2,201,620 (4.016%)	-0.243 (0.704)
25 - 50 Mb	4,771,238 (8.704%)	-0.234 (0.695)
>50 Mb	2,456,647 (4.481%)	-0.155 (0.581)

Table 6: Amount of applications and mean rating per app size

The mean rating of app sizes is the lowest of applications between 2.5 and 5 Megabits, whereas the highest mean rating is of applications over 50 Megabits. The standard deviation is the smallest for the biggest applications (larger than 50 Megabits), and biggest for the applications smaller than 2.5 Megabits.

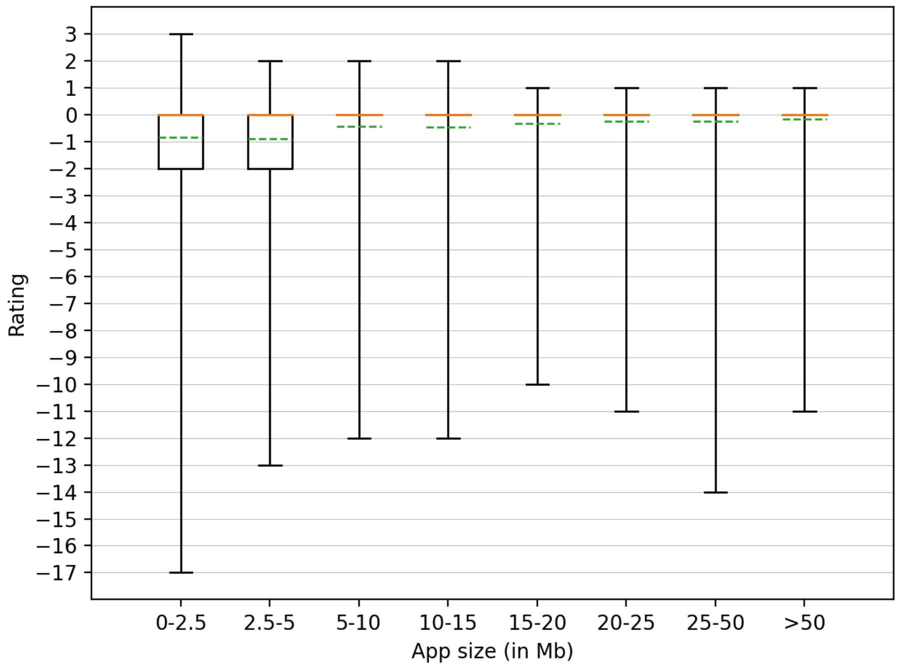

Figure 3: Box plots showing rating distribution of app sizes (from table 6).

First, it is possible that the larger an application becomes, the less votes it will get. This would mean that the rating would stay nearer the zero than smaller applications. The data set unfortunately does not include the amount of votes on an application, so we can not conclusively state whether this is the case.

Second, the size of an application might be an indication of another variable that is not in the data set (a 'proxy variable'). Unfortunately, the data set does not have any comparable or more specific variables relating to the app size, so it can not be said that the app size is a proxy variable based on the current data set.

4.3 Certificates

Table 7 shows the ten most common organization names used in the certificates of the applications.

	Organization name	**Occurrences (% of total)**	**Mean rating (st. dev.)**
1.	`Google Inc.`	5,572,252 (10.165%)	-0.021 (0.217)
2.	`Android`	5,376,286 (9.807%)	-0.917 (1.161)
3.	`Samsung Corporation`	4,247,683 (7.749%)	-0.001 (0.051)
4.	`<empty organization name>`	3,665,931 (6.687%)	-0.732 (1.064)
5.	`Unknown`	1,660,298 (3.029%)	-1.429 (1.266)
6.	`AppInventor for Android`	585,948 (1.069%)	-0.418 (0.825)
7.	`Huawei`	453,149 (0.827%)	-0.004 (0.094)
8.	`CN`	390,019 (0.711%)	-2.042 (1.190)
9.	`Andromo.com L=Winnipeg`	287,427 (0.524%)	-0.004 (0.099)
10.	`www.hao123.com`	233,208 (0.425%)	-0.042 (0.299)

Table 7: Ten most common organization names

The mean ratings in table 7 vary between -0.001 and -2.042. The API does not give any information about the amount of votes for an application, but a manual check on the website of Koodous showed that most of the applications for the organization names with mean ratings around zero did not have any votes. This makes it likely that these applications have a rating and standard deviation around zero because they have not been voted on (instead of the application deliberately being rated higher).

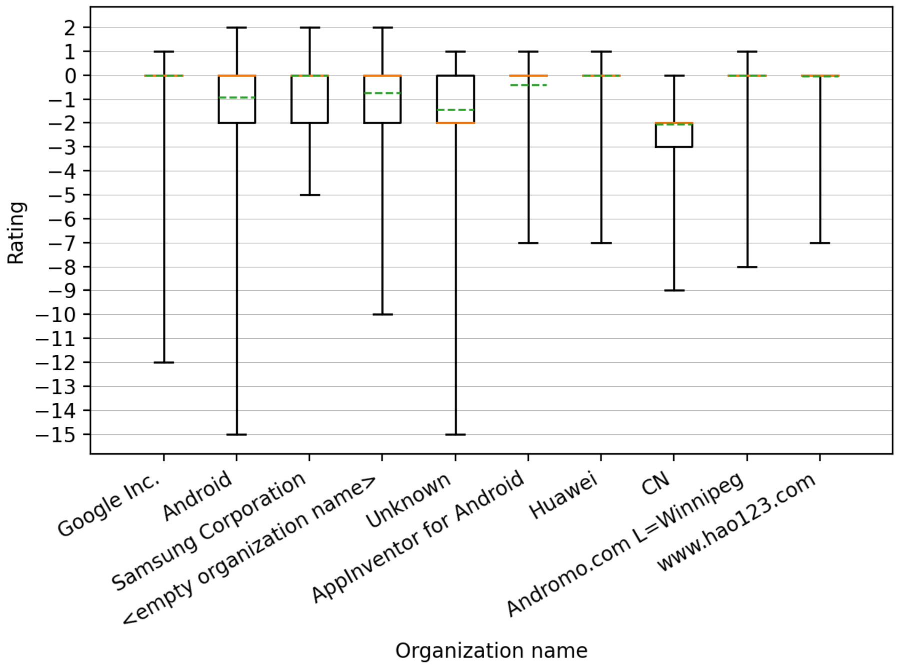

Figure 4: Box plots showing rating distribution of organizations (from table 7).

The fourth most common organization name used in the certificates of the applications is an empty value (indicated as `<empty company name>`). It is not mandatory to fill in any value to create a certificate for an application (as mentioned earlier in section 2.3.2). Some app stores do require applications to have a valid organization name in their certificate.[8]

“CN“, the eight most common name organization name used is likely to be the result of a wrongly formatted certificate for the application. It refers to an earlier identifier of the certificate called the ’Common Name’.

The ninth most common organization name in table 7 is “Andromo.com L=Winnipeg“. This organization name refers to a website used to create Android applications without having to write code.[9] Due to a single missing comma, the ’Locality’ identifier is also included in the organization name.

4.4 App icon

Table 8 lists the amount of applications that do and do not have an icon stored for it.

Applications that do have an icon have a lower mean rating (−0.759 with a standard deviation of 1.125) than applications that do not have an icon (−0.113 with a standard deviation of 0.516). Applications with an icon are however more common (77.3% compared to 22.7%).

There are multiple possible explanations for this difference in the mean ratings between the

[8] https://developer.android.com/studio/publish/app-signing

[9] andromo.com

	Occurrences (% of total)	**Mean rating (st. dev.)**
App icon	42,375,706 (77.3%)	-0.759 (1.125)
No app icon	12,443,421 (22.7%)	-0.113 (0.516)
Total	54,819,127 (100.0%)	-0.635 (1.053)

Table 8: Use of app icons

two categories. First, applications without an icon might be work-in-progress, and thus might not have an icon yet. The app still being developed could indicate that there currently is not any malicious behavior.

Second, it is possible that applications that do not have an icon are less likely to be voted on. Apps without icons can also be widgets, or have been packaged incorrectly. In these cases, users have less information to use when giving a rating, and thus might withhold to vote. This would also explain the lower standard deviation for the category of applications without an icon.

5 Classification models

5.1 Task

The task for the model is to predict the rating of an application, based on its features. Since users are likely to only care whether the application is rated positively or negatively, the model will only have to predict whether the rating of an application is positive (rating of zero or higher) or negative (rating lower than zero). In the context of this thesis, the negative class indicates if an application is malicious and the positive class indicates if an application is benign. This is a classification task.

5.2 Features

The features that are used in the model are listed in table 9.

Application feature	**Data type**	**Description**	**Source**
Creation timestamp	Integer	Timestamp when the application was uploaded to Koodous (Epoch time).	Assigned by Koodous.
Availability of app icon	Boolean	Indicates if there is an icon available for the app.	Property of the application.
App size	Integer	Size of the application in bits.	Property of the application.
`TLD`-flag	Boolean	Indicates if the first segment of the application's package name is a top-level domain.	Property of the application.
Organization name variables	Boolean	One-hot encoded variables for the ten most common organization names (see section 5.2.2).	Property of the application.
Package name beginnings variables	Boolean	One-hot encoded variables for the ten most common TLDs and non-TLDs used as package name beginnings (see section 5.2.2).	Property of the application.
`analyzed`-flag	Boolean	Shows if Koodous has created an analysis report of the application.	Assigned by Koodous.
`trusted`-flag	Boolean	Shows if the application has been whitelisted by Koodous.	Assigned by Koodous.
`detected`-flag	Boolean	Shows if the application has been detected to be malicious by Koodous	Assigned by Koodous.
`corrupted`-flag	Boolean	Shows if the APK will be able to be executed correctly.	Assigned by Koodous.
`on_devices`-flag	Boolean	Shows if the application is (or was) installed on some device.	Assigned by Koodous.

Table 9: All features used in the classification model

5.2.1 Feature sets

The last column in the table indicates if the value of a feature is inherent to the application, or if the value is assigned by Koodous (similar to table 1). We will use two different sets of features:

one with all features and one with only the features that are a property of the application.

Limiting the amount of application features the model is allowed to use can give more information about how accurately different combinations of features can predict the class of an application. Additionally, this will also help to evaluate the model when the application features are not determined by Koodous. This would help to know more about the maliciousness of an application before uploading it to Koodous.

5.2.2 One-hot encoded variables

To use the information from the most common package name beginnings (TLD and non-TLD) and most common organization names (tables 4, 5 and 7), we decided to add them as categorical variables. With these categorical variables, the model will be able to use the knowledge it has about these variables to improve the classification of all applications.

For each category there are ten variables, making a total of thirty variables. One-hot encoding is used for these variables, meaning that the column that corresponds with the same package name beginning or organization name as the application will have a 1 to indicate. For example, if an application has the same organization name as one of the ten most common organization names, that column will show a 1 and a 0 is shown for all other one-hot encoded organization columns.

5.2.3 Normalization

The timestamp and the app size features in the data set are normalized before they are used in the neural network model or in the random forest model. This is to prevent adverse effects during the convergence of the neural network. The features are scaled using the `normalize` function in `sklearn`.

5.3 Balancing the data set

As noted earlier, the applications are skewed towards positive applications (73.1% of the total data set). To prevent the model from being biased in giving the correct classification, we will run the same models with a balanced data set and show these results alongside the unbalanced data set. The data set is balanced by undersampling the larger (in this case: positive) class.

5.4 Classifiers

The classifiers used for the models are a neural network (using `Keras`, which is based on Tensorflow 2) and the random forest classifier (using the `scikit-learn` library).

The neural network used in this thesis is a multilayer perceptron, a type of artificial neural network. This neural network has three layers: an input layer, a hidden layer and an output layer. The input layer receives all the features, and the output layer. The neural network is optimized using the Adam optimizer (described in [KB14]), and uses the `SparseCategoricalCrossentropy` as the loss function.

The hyperparameters used for the random forest classifier are `n_estimators` = 15 and `max_depth` = 6. These parameters were chosen to lower the processing time needed to obtain results. Additionally, since there are continuously applications being added and rated on Koodous, this would improve the time needed to create a new model using the information from the newer applications.

Using these two models simultaneously allows us to achieve high accuracies, but also to evaluate the classification better. In addition, the random forest model has the extra benefit of allowing us to retrieve the feature importances.

6 Results

Table 10 shows the accuracy and F1 scores from the neural network model. The table shows which feature set has been used (all features, or only the features that are a property of the application) and if the data set has been balanced beforehand.

The accuracy score is how many correct predictions the model made out of all predictions in percentage. The best accuracy score is 1 (all predictions are correct), and the worst accuracy score is 0 (none of the predictions are correct. The F1 score is a weighted average of the precision (percentage of correct class predictions across all predictions of that class) and the recall (percentage of how many instances of a class have been correctly predicted). The best possible value for the F1 score is also 1 (perfect precision and recall), with the worst value also being 0 (precision and recall are both zero).

Feature set	**Balanced**	**Unbalanced**
All features	Accuracy: `0.9878` F1 score: `0.9878`	Accuracy: `0.9850` F1 score: `0.9798`
Without features assigned by Koodous	Accuracy: `0.6231` F1 score: `0.5767`	Accuracy: `0.7723` F1 score: `0.5037`

Table 10: Accuracy and F1 score of the neural network model for each combination of feature set and balanced or unbalanced data set.

The neural network model with all features is more accurate when it is balanced than when it is unbalanced (98.8% and 98.5%). The F1 score is also higher with the balanced data set with all features (98.8% compared to 98.0%).

The opposite is true for the model using the limited data set (without the features assigned by Koodous), where the accuracy is lower for the balanced data set (62.3% compared to 77.3%). The F1 score is still higher for the balanced data set than for the unbalanced data set (57.7% compared to 50.1%).

Feature set	**Balanced**	**Unbalanced**
All features	Accuracy: `0.9878` F1 score: `0.9878`	Accuracy: `0.9850` F1 score: `0.9798`
Without features assigned by Koodous	Accuracy: `0.6105` F1 score: `0.5438`	Accuracy: `0.7624` F1 score: `0.4395`

Table 11: Accuracy and F1 score of the random forest model for each combination of feature set and balanced or unbalanced data set.

The random forest model with all features performs similarly to the neural network model network, with the same (rounded) values being reached for both the accuracy scores and the F1 scores. It is more accurate when it is balanced than when it is unbalanced (98.8% compared to 98.5%), and it also has a higher F1 score for the balanced data set (98.8% compared to 98.0%).

Without the features assigned by Koodous, the random forest achieves similar (but slightly lower) accuracy scores (61.1% for the balanced data set, 76.2% for the unbalanced dataset). There is a bigger difference between the F1 scores of the neural network model and the random forest model for the data set with a limited amount of features, but the F1 score for the balanced data set is still larger (54.4% compared to 44.0%).

6.1 Feature importances

In this section of the thesis, we will investigate the feature importances from the random forest model. Table 12 only includes the features that are not one-hot encoded.

The table shows which feature set has been used (all features, or only the features that are a property of the application) and if the data set has been balanced beforehand. The importance is shown in brackets behind each feature.

Feature set	**Balanced**	**Unbalanced**
All features	1. `detected` (0.76558) 2. `on_devices` (0.06329) 3. `analyzed` (0.00528) 4. `is_tld` (0.00098) 5. `corrupted` (0.00038) 6. `trusted` (<0.00010) 7. `app_size` (<0.00010) 8. `timestamp` and `has_icon` (0.00000)	1. `detected` (0.62180) 2. `on_devices` (0.10355) 3. `corrupted` (0.00930) 4. `analyzed` (0.00412) 5. `is_tld` (0.00113) 6. `trusted` (<0.00010) 7. `app_size` (<0.00010) 8. `timestamp` and `has_icon` (0.00000)
Only features that are properties of the application	1. `is_tld` (0.01620) 2. `app_size` (<0.00010) 3. `has_icon` (0.00000)	1. `is_tld` (0.03163) 2. `app_size` (<0.00010) 3. `has_icon` (0.00000)

Table 12: Feature importances for all features, excluding one-hot encoded features

The lower two features in the balanced and unbalanced data set for the limited set of features have importances of zero. As we can see later in the results, this is due to the one-hot encoded features being more important in identifying the correct class when there are less features.

In table 13, the most important one-hot encoded variables are listed. This list shows which companies are the most influential in determining the correct classifications for the applications in the data set.

Feature set	**Balanced**	**Unbalanced**
All features	1. `Google Inc.` (0.07764)	1. `Google Inc.` (0.07198)
	2. `Samsung Corporation` (0.02803)	2. `Samsung Corporation` (0.06640)
	3. `www.hao123.com` (0.01805)	3. `Unknown` (0.03362)
	4. `<empty organization name>` (0.00922)	4. `CN` (0.02184)
	5. `Huawei` (0.00905)	5. `<empty organization name>` (0.02030)
	6. `Unknown` (0.00828)	6. `www.hao123.com` (0.01130)
	7. `CN` (0.00565)	7. `Android` (0.00840)
	8. `Andromo.com L=Winnipeg` (0.00146)	8. `Andromo.com L=Winnipeg` (0.00443)
	9. `Android` (0.00061)	9. `Huawei` (0.00344)
	10. `AppInventor for Android` (<0.00010)	10. `AppInventor for Android` (<0.00010)
Only features that are properties of the application	1. `Samsung Corporation` (0.25862)	1. `Samsung Corporation` (0.24017)
	2. `Google Inc.` (0.19983)	2. `Google Inc.` (0.20053)
	3. `Unknown` (0.09460)	3. `Unknown` (0.12590)
	4. `Huawei` (0.08041)	4. `<empty organization name>` (0.09451)
	5. `www.hao123.com` (0.07002)	5. `Huawei` (0.06629)
	6. `<empty organization name>` (0.06517)	6. `CN` (0.05600)
	7. `CN` (0.05098)	7. `www.hao123.com` (0.03933)
	8. `Andromo.com L=Winnipeg` (0.03193)	8. `Android` (0.02907)
	9. `Android` (0.02812)	9. `Andromo.com L=Winnipeg` (0.02353)
	10. `AppInventor for Android` (0.00406)	10. `AppInventor for Android` (0.00069)

Table 13: Feature importances for one-hot encoded organization name features

Table 14 lists the most important package name beginnings for each combination of feature set and balanced or unbalanced.

Feature set	**Balanced**	**Unbalanced**
All features	1. `bs` (0.00279)	1. `com` (0.00719)
	2. `kr` (0.00146)	2. `seC` (0.00541)
	3. `com` (0.00101)	3. `net` (0.00176)
	4. `appinventor` (0.00062)	4. `bs` (0.00163)
	5. `net` (0.00030)	5. `jp` (0.00106)
	6. `SeC` (0.00023)	6. `air` (0.00071)
	7. `cn` (<0.00010)	7. `cn` (0.00018)
	8. `io` (<0.00010)	8. `aire` (0.00013)
	9. `org` (<0.00010)	9. `org` (0.00013)
	10. `jp` (<0.00010)	10. `appinventor` (0.00012)
	11. `de` (<0.00010)	11. `io` (<0.00010)
	12. `air` (<0.00010)	12. `de` (<0.00010)
	13. `br` (<0.00010)	13. `br` (<0.00010)
	14. `theme` (<0.00010)	14. `theme` (<0.00010)
	15. `minuhome` (<0.00010)	15. `kr` (<0.00010)
	16. `trustgo` (<0.00010)	16. `minuhome` (<0.00010)
	17. `aimoxiu` (<0.00010)	17. `trustgo`, `aimoxiu`, `wxbit` and `ThemeZilla` (0.00000)
	18. `wxbit` (<0.00010)	
	19. `aire` and `ThemeZilla` (0.00000)	
Only features that are properties of the application	1. `com` (0.02398)	1. `com` (0.04205)
	2. `bs` (0.01977)	2. `SeC` (0.01512)
	3. `jp` (0.01683)	3. `bs` (0.00966)
	4. `SeC` (0.01280)	4. `jp` (0.00664)
	5. `de` (0.00707)	5. `kr` (0.00408)
	6. `br` (0.00610)	6. `br` (0.00306)
	7. `kr` (0.00314)	7. `appinventor` (0.00254)
	8. `air` (0.00289)	8. `aimoxiu` (0.00184)
	9. `theme` (0.00275)	9. `aire` (0.00167)
	10. `io` (0.00142)	10. `org` (0.00142)
	11. `aimoxiu` (0.00091)	11. `net` (0.00116)
	12. `appinventor` (0.00066)	12. `theme` (0.00072)
	13. `net` (0.00057)	13. `io` (0.00070)
	14. `org` (0.00056)	14. `minuhome` (0.00060)
	15. `aire` (0.00050)	15. `cn` (0.00053)
	16. `cn` (0.00011)	16. `air` (0.00039)
	17. `minuhome` (<0.00010)	17. `wxbit` (0.00011)
	18. `trustgo` (<0.00010)	18. `de` (<0.00010)
	19. `wxbit` and `ThemeZilla` (0.00000)	19. `trustgo` and `ThemeZilla` (0.00000)

Table 14: Feature importances for one-hot encoded package name beginning features

The importance values of the package name beginnings are very low. This is likely due to the fact that most of these package name beginnings cover only a small percentage of the total data set, with the exception of the `com` TLD (as can be seen in tables 4 and 5).

6.2 Discussion and analysis

Looking at the scores in tables 10 and 11, we can conclude that it is possible to accurately predict whether a rating is positive or negative given all the application features. The neural network slightly outperforms or equals the random forest for all scores using the full data set.

Looking at the same results, balancing has a large impact on the models. The accuracy score of both the neural network model and the random forest model increase with 15%, but the F1 score lowers with 7% for the neural network model and 10% for the random forest model. This indicates that the models are less likely to be successful in balanced data sets, where it has relatively less information about the malicious applications.

To further investigate where the neural network model fails, we will use the confusion matrix of the neural network with the limited feature set without balancing (table 15) and with balancing (table 16).

	Predicted negative class	**Predicted positive class**
True negative class	2.94%	20.98%
True positive class	1.74%	74.34%

Table 15: Confusion matrix of unbalanced neural network model with limited feature set (without features assigned by Koodous)

	Predicted negative class	**Predicted positive class**
True negative class	47.73%	2.29%
True positive class	35.35%	14.63%

Table 16: Confusion matrix of balanced neural network model with limited feature set (without features assigned by Koodous)

As we can see from the first of the two confusion matrixes, the neural network model for the unbalanced data set mostly predicts that applications are part of the positive class (rating of zero or more). This makes sense, since most applications have a positive rating. The opposite is true for the neural network model when the data set is balanced, where the model mostly predicts that applications have a negative class (rating lower than zero). The neural network model using the limited data set is thus not accurate enough to confidently say whether an application has a positive or negative rating, but can definitely help with the prioritization of having application scanned for malicious behavior.

Looking at the results from the feature importances (table 12), the `detected`-flag was the most important to classify the rating of applications. This makes sense, because this flag shows if an application is found to be malicious by Koodous. The first feature that is not assigned by Koodous is `is_tld`, indicating whether the beginning of a package name is a top-level domain, with the other two features being of little or no importance.

All three most important features for the random forest model using the full data set are assigned by Koodous. This makes it likely that these features are the most valuable for correct classification, which reflects the difference found between the accuracy and F1 scores for the full and limited data sets in tables 10 and 11. Another possible explanation is that users voting on Koodous pay extra attention to the values given by Koodous instead of the values inherent to an application.

Continuing to the feature importances for the organization features, there is similar ranking across all combinations of feature sets and balanced or unbalanced. The only organization names to always be in the top three are `Google Inc.` and `Unknown`. These two organization names are thus the most important overall when classifying applications. `Google Inc.` is the most popular organization name in table 7, and the feature importance is therefore also consistent with the common the organization name is. Other organization names that are not as common in the data set, rank lower in the feature importances (table 13).

The organization name `Unknown` however is only the fifth most common organization name. It is likely that the low mean rating of `Unknown` (-1.429, around 0.8 lower than the overall average rating) increases the importance.

Next, we will discuss the feature importances for the package name beginnings. The only package name beginning that occurs in the top three for all model combinations is `com`. This is understandable, since it is by far the most common package name beginning, occurring with 32.1% of the total dataset (the next most-common package name beginning is only 0.5%). The feature importances of all package name beginnings however are not large enough, and are unlikely to have a sizeable impact on the classification of applications.

The first limitation of this research is the use of the data set. The data set itself is likely to be negatively skewed towards malicious Android applications compared to app markets like Google Play. Second, we have not used the whole data set available on Koodous. Both of these reasons could impact the classification.

Furthermore, as mentioned earlier, we do not have a way to distinguish applications that have not been rated and applications that are purposefully left at a rating of zero. By also including both these applications into the positive class, these might impact the classification of the apps. Only the Koodous API was used in this research, and the inclusion of a web scraper that retrieves the amount of votes from the website could allow us to only include apps that have received votes.

7 Conclusion

Our goal for this thesis was to investigate how the features of an application can help to correctly identify malicious Android apps. Our first research question therefore was: “What are the app features that can be collected from the crowdsourcing mobile malware platform Koodous?” During the data exploration phase of this research (section 4), the different features that are available in the Koodous API were investigated and how these features could be used to predict what the rating of an application would be.

The second research question was: “In what way can these features be used to predict if an application will have a positive or negative rating on Koodous?” We took a machine learning approach to this problem, using a neural network model and a random forest model to see if it is possible to predict the rating of an application based on its features. Using all features, an accuracy of 98.8% could be reached, with an F1 score of 98.8% (with both a balanced or an unbalanced data set). This shows us that it is possible to predict whether the rating of an application will be positive or negative using machine learning.

7.1 Further research

7.1.1 Bias in the Koodous data set

For any data set, it is possible that there are biases present. This thesis was restricted to the Koodous data set, a data set where effectively everyone can vote on an application, without looking at the data from other sources. A suggestion for further research is to look into the possible biases that are available in the data set.

7.1.2 Additional application features

The data set from Koodous used has a limited amount of features available, and differs from the possible features that the end-user will see when installing from other sources, like pre-installed app stores. As a further research suggestion, a model with more features from different sources is likely to improve the classification of apps.

7.1.3 Use of image classification

In table 8, the mean ratings between apps with and without icons are shown. Since the focus of this thesis was on explainable application features, no deep learning methods were used to analyze the app icons. Further research could use the icons in a more qualitative way, using deep learning to predict the ratings of applications on Koodous.

References

[AS18] Kursat Aktas and Sevil Sen. Updroid: Updated android malware and its familial classification. In *Nordic Conference on Secure IT Systems*, pages 352–368. Springer, 2018.

[ASH$^+$14] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[Bit17] Bitdefender. US: Personal data stored on smartphones by 50 percent of users. Technical report, September 2017.

[CPS17] Tanmoy Chakraborty, Fabio Pierazzi, and VS Subrahmanian. Ec2: Ensemble clustering and classification for predicting android malware families. *IEEE Transactions on Dependable and Secure Computing*, 17(2):262–277, 2017.

[F-S17] F-Secure. The State of Cyber Security 2017. Technical report, 2017.

[GSM21] GSM Association. The Mobile Economy 2021. Technical report, 2021.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KCB20] Platon Kotzias, Juan Caballero, and Leyla Bilge. How did that get in my phone? unwanted app distribution on android devices. *arXiv preprint arXiv:2010.10088*, 2020.

[LLB$^+$17] L. Li, D. Li, T. F. Bissyandé, J. Klein, Y. Le Traon, D. Lo, and L. Cavallaro. Understanding android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics and Security*, 12(6):1269–1284, 2017.

[MAY$^+$18] Khulood Al Messabi, Monther Aldwairi, Ayesha Al Yousif, Anoud Thoban, and Fatna Belqasmi. Malware detection using dns records and domain name features. In *Proceedings of the 2nd International Conference on Future Networks and Distributed Systems*, pages 1–7, 2018.

[MDC17] Nikola Milosevic, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Machine learning aided android malware classification. *Computers & Electrical Engineering*, 61:266–274, 2017.

[OAAM$^+$18] Modupe Odusami, Olusola Abayomi-Alli, Sanjay Misra, Olamilekan Shobayo, Robertas Damasevicius, and Rytis Maskeliunas. Android malware detection: A survey. In *International conference on applied informatics*, pages 255–266. Springer, 2018.

[O'D21] S. O'Dea. Mobile os market share 2021, June 2021.

[PGFF20] Ya Pan, Xiuting Ge, Chunrong Fang, and Yong Fan. A systematic literature review of android malware detection using static analysis. *IEEE Access*, 8:116363–116379, 2020.

[TFA$^+$17] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. The evolution of android malware and android analysis techniques. *ACM Comput. Surv.*, 49(4), January 2017.

[WCY$^+$20] Shanshan Wang, Zhenxiang Chen, Qiben Yan, Ke Ji, Lizhi Peng, Bo Yang, and Mauro Conti. Deep and broad url feature mining for android malware detection. *Information Sciences*, 513:600–613, 2020.

[ZD16] Ziyun Zhu and Tudor Dumitraş. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 767–778, 2016.